FUJITSU

FUJITSU Software Technical Computing Suite V4.0L20

Job Operation Software API user's Guide for Command API

J2UL-2544-01ENZ0(01)
September 2020

Preface

Purpose of This Manual

This manual describes the command API of the job operation management function provided by the Job Operation Software of Technical Computing Suite.

Intended Readers

This manual is intended for the administrators who operate and manage jobs using the Job Operation Software, and the end users who actually perform operations with jobs.

The manual assumes readers have the following knowledge:

- Basic Linux knowledge
- Knowledge of job operations (submit a job, delete a job, etc.) described in "Job Operation Software End-user's Guide"
- Knowledge of job operation control described in "Job Operation Software Administrator's Guide for Job Management"

Organization of This Manual

This manual is organized as follows.

Chapter 1 Command API Overview

This chapter provides an overview of the command API.

Chapter 2 Using the Command API

This chapter describes how to use the command API.

Appendix A Command API Common Reference

This appendix describes common command API operations and an API for referencing information.

Appendix B Job Operation API Reference

This appendix describes the API for job operations.

Appendix C Information Acquisition API Reference

This appendix describes the API for getting job and resource information.

Appendix D Job Operation Control API Reference

This appendix describes the API for permitting job submission and job execution.

Appendix E Sample Programs

This appendix provides simple sample programs using the command API.

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.

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 2020, Version 1.1	J2UL-2544-01ENZ0(01)
February 2020, First version	J2UL-2544-01ENZ0(00)

Copyright

Update history

Changes	Location	Version
Added "no-io-exclusive" and " io-exclusive" to the possible values for the parameter "node" of the function pjcmd_submit_put_job_resource().	B.1.10	1.1

Contents

Chapter 1 Command API Overview

This chapter provides an overview of the command API.

1.1 What is the Command API?

The user interface required for job operations varies depending on the operation system.
For example, some users may want to change or disable a command option name according to the job operation policy. Some users may want to change the format of a command message and information output by a command to the original format.

To help create commands with the user interface preferred by the user, the job operation management function provides an interface to call the same functions (job operation and information acquisition) as commands provided by the job operation management function. The interface is called the command API (Application Programming Interface). The command API enables creating of commands with the user interface preferred by the administrator or end user.

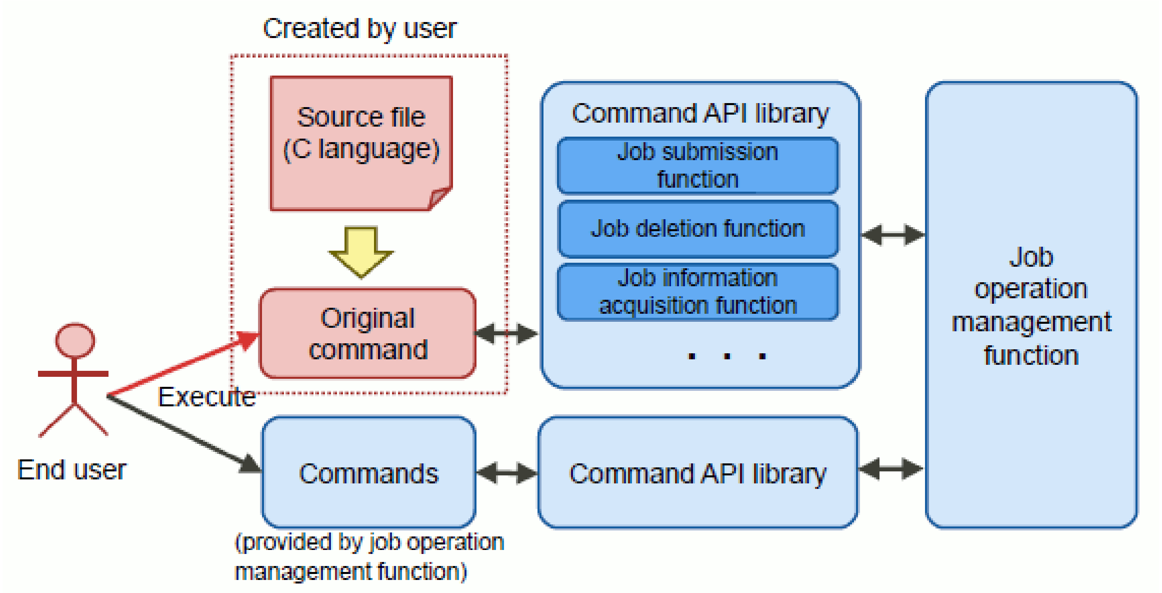
The command API consists of C language functions. They are provided as a library. This library can be used on the login node, compute cluster management node, and system management node.

The command API supports the functions listed in the following table.

Table 1.1 Functions Supported by the Command API

Function	Command Equivalent to Function
Submitting a job	pjsub command
Deleting a job	pjdel command
Changing a job parameter	pjalter command, pmalter command
Sending a signal to a job	pjsig command
Holding a job and releasing a job hold	pjhold command, pjrls command
Waiting for a job to complete	pjwait command
Getting job information	pjstat command (-s and -S options)
Getting resource unit and resource group information	pjstat command (--rsc option)
Getting job limit value information	pjstat command (--limit option)
Getting job ACL function information	pjacl command
Getting the status of job resource usage	pjshowrsc command
Changing the job operation and getting the job operation status (setting permissions for job submission or execution and getting the setting status)	pmpjmopt command (--set-rsc-ru option, --show-rsc-ug option)

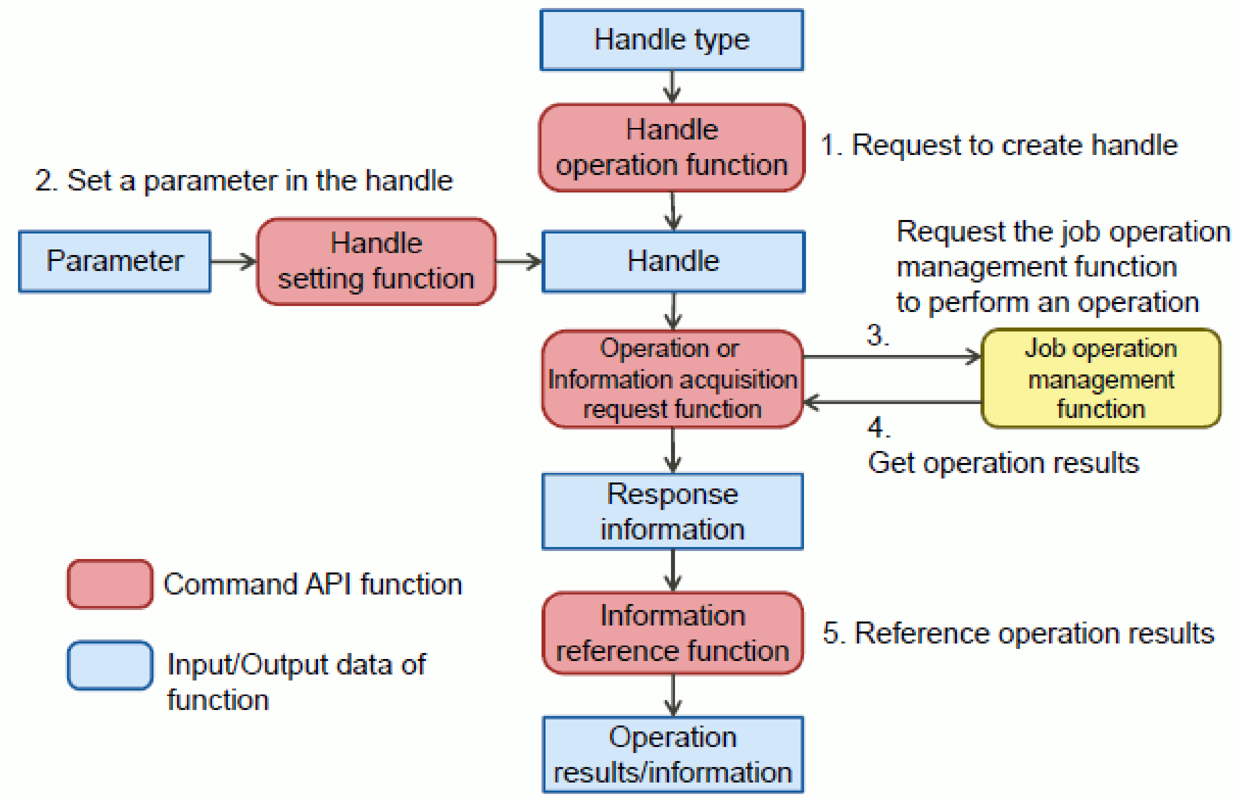
Figure 1.1 Conceptual Diagram



1.2 Operation Flow

The following figure shows the flow of operations using the command API. For details on how to use the command API, see "Chapter 2 Using the Command API."

Figure 1.2 Flow of Operations Using the Command API


Handle type
Handle operation function
1. Request to create handle
2. Set a parameter in the handle
Parameter
Handle setting function
Handle
Request the job operation management function to perform an operation
3.
Operation or Information acquisition request function
Job operation management function
4.
Get operation results
Response information
Command API function
Input/Output data of function
Information reference function
5. Reference operation results
Operation results/information

1. Request generation of a handle

   To use the command API, a handle is first generated using the handle operation function. The handle is information for holding a parameter to make a request for job operation and information acquisition to the job operation management function. The handle type to be generated is specified for the job operation management function based on the request type.

2. Set a parameter in the handle

   A parameter is set in the generated handle by the handle setting function. The parameter is information indicating the contents of a request to the job operation management function, such as the amount of resources allocated to a job to be submitted or a job ID to be deleted.

3. Request the job operation management function to perform an operation

   After the parameter is set in the handle, an operation request is made to the job operation management function by the request function. The job operation management function processes the operation based on the handle contents.

4. Get operation results

   Response information about the results of the request to the job operation management function is returned as a return value for the request function.

5. Reference the operation results

   The response information includes information regarding the success/failure of the operation and obtained information. The information is referenced with the information reference function. When getting information, a structure with more detailed information may also be obtained from the response information in order to reference it.

1.3 Function Types

The following table lists the names of the command API functions by operation type.

Table 1.2 Command API Functions

Operation	Function Name
Submitting a job	pjcmd_submit_*action*()
Deleting a job	pjcmd_kill_*action*()
Changing a job parameter	pjcmd_alter_*action*()
Sending a signal to a job	pjcmd_signal_*action*()
Holding a job	pjcmd_hold_*action*()
Releasing a job hold	pjcmd_release_*action*()
Waiting for a job to complete	pjcmd_wait_*action*()
Getting job information	pjcmd_jobinfo_*action*()
Getting resource unit and resource group information	pjcmd_rscinfo_*action*()
Getting limit value information when submitting a job	pjcmd_limitinfo_*action*()
Getting job ACL function information	pjcmd_jacl_*action*()
Getting the status of job resource usage	pjcmd_rscstat_*action*()
Changing the job operation and getting the job operation status (setting permissions for job submission or execution and getting the setting status)	pjcmd_setpjmstat_*action*() pjcmd_getpjmstat_*action*()
Utility function (supplementary function to use the command API, such as for a handle operation and response information, and reference results)	pjcmd_*action*()

"*action*" in a function name indicates the process type. The following table lists processes.

Table 1.3 Process Types of Command API Functions

Process	Description	Function Example

Handle operation	Generates, initializes, replicates, or releases a handle. A specific function is prepared for each operation.	pjcmd_create_handle()
		pjcmd_reset_handle()
		pjcmd_clone_handle()
		pjcmd_destroy_handle()

Process	Description	Function Example

Parameter setting and reference	Sets a job operation, information acquisition, or other parameter in a handle, and references a setting value.
	A specific function is prepared for each function and parameter type.
		pjcmd_submit_put_param()
		pjcmd_submit_get_param()

Argument analysis	Analyzes the provided command line argument based on the command option specification of the job operation management function, and sets a parameter in a handle.
	These functions are used when command arguments are analyzed by the command API based on the command option system of the Job Operation Software rather than by the user.
	An option that is equivalent to the pjsub command can be customized (changing and disabling an option name) using a function called parser.
		pjcmd_submit_parse_pjsub_args()
		pjcmd_getopt_long()
		pjcmd_renameopt_in_parser()
		pjcmd_delopt_in_parser()

Operation request	Makes a request for job operation and information acquisition to the job operation management function based on the set information in a handle. A specific function is provided for each function type.	pjcmd_submit_execute()
Result reference	References the success/failure of a request and obtained information (job ID, job information, etc.) from response information about operation results and information acquisition results. A specific function is provided for each operation type and individual reference information.	pjcmd_get_jobresult_info()
Display	Outputs the contents of response information based on the standard command display specifications. This function can be used when there is no need to customize information display.	pjcmd_jobinfo_print_resp()
Response information operation	Releases response information. Response information is held in the command API. The user releases the information when it becomes unnecessary.	pjcmd_destroy_resp()

See

For details on the functions, see "Appendix A Command API Common Reference" to "Appendix D Job Operation Control API Reference."

Chapter 2 Using the Command API

This chapter describes how to use the command API.

2.1 Header File

The header file of the command API is located at the following location on each login node, compute cluster management node, and system management node.

```
/usr/include/FJSVtcs/pjm/pjcmd.h
```

This header file must be included in a source file that uses the command API.

```
#include <FJSVtcs/pjm/pjcmd.h>
...
main()
{
    // Process using the command API
    ...
}
```

2.2 Operations Using the Command API

This section describes how to use typical functions based on the flow of operations using the command API.

2.2.1 Generating a Handle

The command API sets an operation-related parameter in a handle and makes a request to the job operation management function. For that purpose, the handle needs to be created first.

A handle is created using the pjcmd_create_handle() function.

```
PjcmdHandle_t *handle_p;

handle_p = pjcmd_create_handle(PJCMD_SUBMIT);  // Generate handle for job submission
```

The pjcmd_create_handle() function creates a handle (PjcmdHandle_t). When the function succeeds, a pointer to the area is returned.

Note

A handle is prepared for each operation type. For example, a handle generated for job submission cannot be used for job deletion.

2.2.2 Setting a Parameter

A parameter indicating operation details is set in a handle. The parameter is equivalent to a command option (pjsub command, etc.) of the job operation management function.

For example, when a bulk job is submitted, an argument is specified in the pjsub command as follows.

```
pjsub -L "node=8x8" --bulk --sparam "1-10" ./job.sh
```

To realize this using the command API, parameters that are equivalent to these options are set in a handle as follows.

```
char *jobscript_p = "./job.sh";                  // Job script ./job.sh
char *node_p = "8x8";                            // Node shape 8x8
pjcmd_jobmodel_t jobmodel = PJCMD_JOBMODEL_BULK; // Job model: Bulk job
uint32_t bulk_sno = 1, bulk_eno = 10;            // Bulk start number 1, bulk end number 10

pjcmd_submit_put_param(handle_p, PJCMD_SUBMIT_SCRIPTFILE, &jobscript_p);
```

```
pjcmd_submit_put_job_resource(handle_p, "node", &node_p);
pjcmd_submit_put_param(handle_p, PJCMD_SUBMIT_JOBMODEL, &jobmodel);
pjcmd_submit_put_param(handle_p, PJCMD_SUBMIT_BULK_STARTNO, &bulk_sno);
pjcmd_submit_put_param(handle_p, PJCMD_SUBMIT_BULK_ENDNO, &bulk_eno);

* Determination of the function return value is omitted.
```

Note

- Note that, in the above example, instead of a parameter value, the address of the storage area of the parameter value is passed to a handle when setting a parameter value (pointer or numerical value) in order to handle various data-type parameters using one function.
  For example, the pjcmd_submit_put_param() function specifies a parameter type in the second argument, and specifies the address of the storage area of the parameter in the third argument (void * type). The function determines a parameter type based on the parameter type indicated by the second argument.

```
Incorrect example:
char *jobscript_p = "./job.sh";
char *node_p = "8x8";
pjcmd_jobmodel_t jobmodel = PJCMD_JOBMODEL_BULK;

pjcmd_submit_put_param(handle_p, PJCMD_SUBMIT_SCRIPTFILE, jobscript_p); // Pass value (pointer)
pjcmd_submit_put_job_resource(handle_p, "node","8x8");                  // Pass value (pointer)
pjcmd_submit_put_param(handle_p, PJCMD_SUBMIT_JOBMODEL, jobmodel);      // Pass value (numerical
value)
```

- A parameter passed to a function is copied to a handle in the function. Therefore, if the function is successful, changing the original parameter does not affect the handle contents.

Similar to when deleting a job, a job ID is set in a handle using the pjcmd_put_job() function in an operation to specify a target job.

In the following example, a normal job whose job ID is 10 is set in a handle.

```
int64_t jobid[2] = {10, -1};  // Specify job ID as array that ends with -1

pjcmd_put_job(handle_p, jobid, 0, 0, NULL, 0, 0, PJCMD_JOBMODEL_NORMAL);
```

A job ID can also be set in a handle by selecting from job IDs that are specified using a character string.

```
char *jobid_str_p = "10[1-5]";  // Bulk jobs  10[1] to 10[5]

pjcmd_put_job_by_str(handle_p, jobid_str_p);
```

Thus, by using this function, a job ID that is specified in a command line argument can be directly set in a handle without converting it to a numerical value.

2.2.3 Processing a Command Line Option

The command API has a function to recognize the same options as a command of the job operation management function and set parameters in a handle.

2.2.3.1 Parameter Setting From a Command Argument

"2.2.2 Setting a Parameter" describes individual parameter settings made by the users. However, if a command that uses the same option system as a command provided by job operation management function is created, the arguments of the command can be batch analyzed by the command API and set in a handle.

For example, if the same specifications as the pjsub command are applied to the options of a command to be created, parameters can be set in a handle by analyzing the arguments with the pjcmd_submit_parse_pjsub_args() function.

```
main(int argc, char **argv_pp)
{
    PjcmdHandle_t *handle_p;
    char *jobscript_p;
    ...
    pjcmd_submit_parse_pjsub_args(handle_p, argc, argv_pp);
    if (pjcmd_optind >= argc) {
        Error processing;
    }
    jobscript_p = argv_pp[pjcmd_optind];
    pjcmd_submit_put_param(handle_p, PJCMD_SUBMIT_SCRIPTFILE, &jobscript_p);
    ...
```

When the pjcmd_submit_parse_pjsub_args() function ends successfully, the pjcmd_optind global variable of the command API indicates the first argument that is not an option of the argv_pp[] argument array. The caller needs to process the arguments that are not options. In the following example, if there are any remaining arguments after setting the options in a handle with the pjcmd_submit_parse_pjsub_args() function, argv_pp[pjcmd_optind] is regarded as a job script and set in the handle (the subsequent arguments are ignored).

```
    ...
    if (pjcmd_optind >= argc) { // No remaining argument
        ... // Error processing
    }
    jobscript_p = argv_pp[pjcmd_optind];
    pjcmd_submit_put_param(handle_p, PJCMD_SUBMIT_SCRIPTFILE, &jobscript_p);  // Set job script
                                                                              // in handle
    ...
```

2.2.3.2 Command Line Parser

The names of the options that are recognized by the command API can be changed or disabled. By doing so, while inheriting the options of a command of the job operation management function using a command to be created, an option name can be changed to another name or unnecessary options can be disabled.

The command API treats option specifications (option name, meaning, and information on whether or not to have arguments) as information called a command line parser. When an option name is changed or disabled, a command line parser is created first, and then the setting is made into the parser.

Note

- The command line parser is a function enabled in a program that uses the command API. The function does not operate the options of a command of the job operation management function.
- Currently, the command line parser supports only the options equivalent to the pjsub command that is related to the job submission operation.

The following example changes the -L option in the pjsub command, which is recognized by the command API, into the -R option, and disables the --fs and --appname options.

```
PjcmdHandle_t *handle_p;
PjcmdParser_t *parser_p;

handle_p = pjcmd_create_handle(PJCMD_SUBMIT);           // Generate handle for job submission
parser_p = pjcmd_submit_create_pjsub_parser(handle_p); // Generate command line parser related to job
submission

pjcmd_renameopt_in_parser(parser_p, 'L', 'R', NULL, NULL); // Change -L option into -R option
pjcmd_delopt_in_parser(parser_p, 0, "fs");                 // Disable --fs option
pjcmd_delopt_in_parser(parser_p, 0, "appname");            // Disable --appname option

pjcmd_submit_parse_pjsub_args(handle_p, argc, argv_pp);    // Analyze options using above settings
```

Note

- A parser is associated with a handle. Do not release the handle corresponding to an option before analyzing the option. If the handle is released first, operation is undetermined.
- When an option name is changed or disabled using a parser, it is also applied to analysis of arguments using the pjcmd_submit_parse_pjsub_args() function that specifies the handle associated with the parser.

2.2.3.3 Option Analysis

When a command with original options needs to be created, in addition to the options of a command provided by the Job Operation Software, only analysis of the original options is required as the argument analysis process if the pjcmd_getopt_long() function is used.
The pjcmd_getopt_long() function is similar to the C language library function getopt_long(). However, if the command options that are recognized by the command API are detected, the pjcmd_getopt_long() function does not return but analyzes them internally. Then, the function sets parameters in the handle associated with the command line parser.
If an option that cannot be recognized by the command API, such as an original option, is detected, the pjcmd_getopt_long() function operates the same way as the pjcmd_getopt_long() function and returns.

In the following example, some of the options that are equivalent to the pjsub command of the job operation management function are changed or disabled, and an original -Q option is processed during the job submission process.

```
main(int argc, char **argv_pp)
{
    int c, Q_flag = 0;
    char *myopts_p = "Q";  // Original -Q option
    PjcmdHandle_t *handle_p;
    PjcmdParser_t *parser_p;
    ...

    handle_p = pjcmd_create_handle(PJCMD_SUBMIT);               // Generate handle
    parser_p = pjcmd_submit_create_pjsub_parser(handle_p);      // Create command line parser
    pjcmd_renameopt_in_parser(parser_p, 'L', 'R', NULL, NULL); // Change -L option into -R option
    pjcmd_delopt_in_parser(parser_p, 0, "fs");                 // Disable --fs option
    pjcmd_delopt_in_parser(parser_p, 0, "appname");            // Disable --appname option

    // Analyze argv command line argument and set it in handle
    // Recognized options are same as pjsub command option (parser_p) and -Q (myopts_p) option
    while ((c = pjcmd_getopt_long(parser_p, argc, argv_pp, myopts_p, NULL, NULL) != -1) {
        // pjcmd_getopt_long() does not return upon detecting the options recognized
        // by the command line parser.
        if (c == -1)
            break;
        switch (c) {
        case 'Q':  // Original -Q option
            Q_flag = 1;
            break;
        default:   // This is neither an option recognized by the command line parser
                   // nor an original option.
                   // Disabled options are treated as unknown options.
            fprintf(stderr, "%c: Unknown option\n", c);
            break;
    }
    ...
    if (Q_flag) {
        // Process when specifying -Q option
    }
    ...
}
```

Note

Currently, the command line parser supports only the pjsub command options. Therefore, only the pjsub command options can be analyzed by the pjcmd_getopt_long() function that requires the command line parser.

2.2.4 Requesting the Job Operation Management Function

2.2.4.1 Operation Request Function Call

After setting parameters in a handle, an operation is requested of the job operation management function by the operation request function.

For example, job submission is requested as follows.

```
PjcmdHandle_t *handle_p
PjcmdResp_t *resp_p;
...
resp_p = pjcmd_submit_execute(handle_p);
```

Response information (PjcmdResp_t) indicating operation results is returned as the function return value.

Note

- An operation must be requested after setting all necessary parameters in a handle. If the necessary parameters for the operation are not set, an error occurs.
  For example, if a bulk start number is not set when submitting a bulk job, or if a job ID is not set when deleting a job, an error occurs.
- If the operation request function is called in a series, the requests to the job operation management function are held in the function until a certain period of time has passed after the first call. During the period, the function does not return. The reason is to prevent increasing the load of the job operation management function. The administrator can change the length of time. For details on how to set parameters, see "Command API settings" in "Chapter 3 Job Operation Management Function Settings" in "Job Operation Software Administrator's Guide for Job Management."
- The command API communicates with the job operation management function every time an operation is requested. Therefore, a new operation request cannot be accepted until the operation requests to the job operation management function are completed in one program.
  - While the operation request function is being processed in another thread, calling a new operation request function results in an error, and PJCMD_ERROR_BUSY is set in pjcmd_errcode. In this case, a new operation request function needs to be called again after the operation request function being processed is completed.
  - When the operation request function returns, the command API may internally communicate with the job operation management function regarding the operation contents until response information is released. For this reason, if the operation request function is called another time after calling the operation request function, the response information must be released first.
- The operation request function operates as follows against some signals.

  ```
  SIGHUP, SIGQUIT, SIGINT, SIGALRM, SIGTERM: Process is interrupted
  SIGUSR1, SIGUSR2, SIGPOLL, SIGPROF, SIGVTALRM, SIGIO, SIGPWR: Ignore
  ```

  If the operation request function process is interrupted by a signal, the pjcmd_errcode error code becomes PJCMD_ERR_SIGNAL. After the function returns, the signal setting returns back to the previous state that was used before the function was called.

2.2.4.2 Job Submission Operation

A job may be submitted by following a special procedure, which is not used for other operations, depending on the job model. This section describes the precautions when submitting a job with the command API.

- Step jobs
  When the sub jobs of a step job are submitted, one handle is prepared for each sub job to submit them.

When the same step job is submitted, all that needs to be done is to replicate the handle of the previous sub job and update only the necessary parameters.

When multiple sub jobs of one step job are batch submitted, the pjcmd_submit_executev() function can be used.
For example, when multiple sub jobs of a step job are batch submitted by the pjsub command, they are executed as follows.

```
$ pjsub --step -N myjob stepjob0.sh stepjob1.sh stepjob2.sh stepjob3.sh stepjob4.sh
```

The same can be performed using the command API as follows.

```
#define SUBJOB_NUM 5

PjcmdHandle_t *handle_p[SUBJOB_NUM];
PjcmdResp_t *resp_p;
int i;
pjcmd_jobmodel_t model = PJCMD_JOBMODEL_STEP;
char *jobname_p = "myjob";
char *jobscript_p[SUBJOB_NUM] = {"stepjob0.sh", "stepjob1.sh", "stepjob2.sh",  "stepjob3.sh",
"stepjob4.sh"};

handle_p[0] = pjcmd_create_handle(PJCMD_SUBMIT);
pjcmd_submit_put_param(handle_p[0], PJCMD_SUBMIT_JOBMODEL, &model);
pjcmd_submit_put_param(handle_p[0], PJCMD_SUBMIT_JOBNAME, &jobname_p);
pjcmd_submit_put_param(handle_p[0], PJCMD_SUBMIT_SCRIPTFILE, jobscript_p[0]);
...

for (i = 1; i < subjob_num; i++) {
    handle_p[i] = pjcmd_clone_handle(&handle_p[0]); // Replicate handle of first sub job
    pjcmd_submit_put_param(handle_p[i], PJCMD_SUBMIT_SCRIPTFILE, jobscript_p[i]); // Job script
}

resp_p = pjcmd_submit_executev(handle_p, SUBJOB_NUM); // Batch specify and submit handles
                                                      // of sub jobs
```

- Interactive jobs
  When the submission of an interactive job is requested by the pjcmd_submit_execute() function, the function does not return until the input of the interactive job is closed (until the interactive job is completed).

  An interactive job is submitted through the following four steps.

  1. Acceptance of an interactive job is completed.
  2. The interactive job enters the wait state.
  3. Execution of the interactive job begins.
  4. The interactive job is completed.

  The pjsub command of the job operation management function outputs a message indicating the processing progress in each step. The command API provides a callback mechanism to call a function prepared by the user in each step. When the same message as when using the pjsub command needs to be output by a command that uses the command API, a job submission operation must be requested after registering a function to output the message using the pjcmd_submit_set_callback() function. A sub job ID structure (PjcmdSubjobid_t) is passed to an argument of the callback function.

  The following examples show the callback function used to output messages similar to the messages shown below (error processing and other processes are omitted), displayed when submitting an interactive job with the pjsub command.

```
$ pjsub --interact
[INFO] PJM 0000 pjsub Job 405916 submitted.             (*1)
[INFO] PJM 0081 .connected.                             (*2)
[INFO] PJM 0082 pjsub Interactive job 405916 started.   (*3)
$                                                       (*4)
...
$ exit                                                  (*5)
[INFO] PJM 0083 pjsub Interactive job 405916 completed. (*6)
```

```
(*1) Message indicating submission of interactive job
(*2) Message indicating interactive job being prepared
(*3) Message indicating start of interactive job
(*4) Shell prompt in interactive job
(*5) End of shell
(*6) Message indicating completion of interactive job
```

a. Message indicating the submission of an interactive job
A message is displayed by the callback function that is called when acceptance of an interactive job is completed. A job ID consisting of a character string is created ("405916" in the above example) based on the sub job ID structure (PjcmdSubjobid_t) that is provided as an argument of the callback function.

```
void interact_job_accept_msg(const PjcmdSubjobid_t *subjobid_p)
{
    char buf[PJCMD_MAX_SUBJOBID_STR_LEN];

    pjcmd_subjobid_to_str(subjobid_p, buf);
    fprintf(stdout, "[INFO] PJM 0000 mypjsub Job %s submitted.\n", buf);
}
```

b. Message indicating an interactive job is being prepared
A message is displayed by the callback function that is called at a regular interval (every three seconds) while waiting for the execution of an interactive job. In the following example, "." is displayed every time the callback function is called.

```
int interact_job_wait_msg_called = 0;
void interact_job_wait_msg(const PjcmdSubjobid_t *subjobid_p)
{
    if (interact_job_wait_msg_called != 0) {
        // Only beginning of message is displayed when called for first time
        fprintf(stdout, "[INFO] PJM 0081 ");
        interact_job_wait_msg_called = 1;
    }
    fprintf(stdout, ".");
    fflush(stdout);
}
```

c. Message indicating the start of an interactive job
A message is displayed by the callback function that is called when an interactive job starts. A job ID consisting of a character string is created ("405916" in the above example) based on the sub job ID structure (PjcmdSubjobid_t) that is provided as an argument of the callback function.

```
void interact_job_start_msg(const PjcmdSubjobid_t *subjobid_p)
{
    char buf[PJCMD_MAX_SUBJOBID_STR_LEN];

    // Displayed to show end of message indicating job being prepared
    fprintf(stdout, "connected\n");
    pjcmd_subjobid_to_str(subjobid_p, buf);
    fprintf(stdout, "[INFO] PJM 0082 mypjsub Interactive job %s started.\n", buf);
}
```

d. Message indicating the end of an interactive job
A message is displayed by the callback function that is called when an interactive job is completed. A job ID consisting of a character string is created ("405916" in the above example) based on the sub job ID structure (PjcmdSubjobid_t) that is provided as an argument of the callback function.

```
void interact_job_end_msg(const PjcmdSubjobid_t *subjobid_p)
{
    char buf[PJCMD_MAX_SUBJOBID_STR_LEN];

    pjcmd_subjobid_to_str(subjobid_p, buf);
    fprintf(stdout, "[INFO] PJM 0082 mypjsub Interactive job %s completed.\n", buf);
}
```

The callback functions prepared above are registered as follows before requesting submission of an interactive job.

```
if (jobtype == PJCMD_JOBTYPE_INTERACTIVE) {              // When interactive job submitted
    pjcmd_submit_set_callback(handle_p,                  // Handle
                              &interact_job_accept_msg,  // When accepting job
                              &interact_job_wait_msg,    // When waiting for job to start
                              &interact_job_start_msg,   // When job starts
                              &interact_job_end_msg);    // When job ends
}
...
resp_p = pjcmd_submit_execute(handle_p);
...
```

By doing so, a callback function is called at each step of the internal processing of the pjcmd_submit_execute() request function.

- Job from the standard input

  The pjsub command can provide job details from the standard input without specifying a job script. When providing job details in the same way by using the command API, the contents of the standard input are saved in a temporary file in a program and set in a handle as a job script.

  The command API has the pjcmd_submit_create_scriptfile_from_stdin() function that is used to create the contents of the standard input as a temporary job script. The following example uses this function.

```
char *tmp_jobscript_p;

// Input stored in automatically generated file
tmp_jobscript_p = pjcmd_submit_create_scriptfile_from_stdin(NULL, NULL);
if (tmp_jobscript_p == NULL) {
    // To error processing
}

// Set job script name
pjcmd_submit_put_param(handle_p, PJCMD_SUBMIT_SCRIPTFILE, &tmp_jobscript_p);
...
pjcmd_submit_execute(handle_p);
...
// Delete temporarily created job script
unlink(tmp_jobscipt_p);
```

2.2.4.3 Job Operation

The following job operations may take a long time to receive a request for processing by the job operations management function and return a response.

- Deleting a job
- Holding a job
- Release job hold
- Sending a signal to a job
- Changing job parameters

The job operation command of the job operation management function that corresponds to the job operation function above outputs a message indicating the progress of processing when a request takes time to be accepted. The command API provides a callback mechanism to call a user-provided function if the request takes a long time to be accepted.

If you want a command using the command API to process message output similar to various operation commands, register a function that outputs messages with the pjcmd_*xxxx*_set_callback() function and request a job operation.

The following is an example of a callback function. It prints the following message, similar to the pjdel command (Error handling is omitted).

```
$ pjdel 405916
[INFO] PJM 0181 ...done.                       (*)
[INFO] PJM 0100 pjdel Accepted job 405916.

(*) "[INFO] PJM 0181 ..." : A message indicating that a request to delete a job is
                            waiting to be accepted.
    "done." : A message indicating that the request to delete the job has been accepted.
```

a. A message indicating that a request to delete a job is waiting to be accepted.

   A message is output in a callback function called periodically (Every 3 Seconds) while waiting for a delete job request to be accepted. This example outputs "." each time it is called.

```
int kill_wait_msg_called = 0;
void kill_wait_msg(void)
{
    if (kill_wait_msg_called == 0) {
        fprintf(stdout, "[INFO] PJM 0181 "); // First call outputs the beginning of the message.
        kill_wait_msg_called = 1;
    }
    fprintf(stdout, ".");
    fflush(stdout);
}
```

b. A message indicating that the request to delete the job has been accepted.

   A message is output in a callback function that is called when a request to delete a job has been accepted.

```
void kill_accept_msg(void)
{
    fprintf(stdout, "done.\n"); // The end of a message waiting for a response to
                                // a request to delete a job.
}
```

Register the prepared callback function as follows before requesting deletion of the job.

```
pjcmd_kill_set_callback(handle_p,          // Handle
                        &kill_wait_msg,    // Called at waiting acceptance of request
                                           // for job deletion.
                        &kill_accept_msg); // Called when request for job deletion has been accepted.
}
...
resp_p = pjcmd_kill_execute(handle_p);
```

That way, the registered callback function will be called at each stage in the pjcmd_kill_execute() function.

2.2.5 Referencing Results

2.2.5.1 Referencing Response Information

Operation request results are found by referencing response information.

```
PjcmdHandle_t *handle_p;
PjcmdResp_t *resp_p;
int code, subcode;
char *detail_p;
...
resp_p = pjcmd_submit_execute(handle_p);               // Request job submission operation
if (resp_p == NULL) {
    // To error processing
}
```

```
pjcmd_get_result(resp_p, &code, &subcode, &detail_p);  // Get operation request results
if (code != 0) {
    //To error processing
}
...
```

a. When response information is NULL
   If the response information returned as the return value of the operation request function is NULL, it indicates that a problem, such as insufficient contents of a handle, was found by a check before requesting the operation.

b. When response information is not NULL
   When response information is returned, whether or not an operation has been successfully requested is checked with the pjcmd_get_result() function first. If 0 (success) is returned in the second argument (code), detailed results as explained later in this document can be referenced.

The detailed information that can be obtained from response information is different from job operation contents and other operation contents.

- Job operation

  The following information can be obtained from response information about a job operation, such as submitting or deleting a job.

  - Number of operated jobs

    The total number of operation target jobs and the number of successful jobs are obtained by the pjcmd_get_jobresult_num() function.

```
PjcmdHandle_t *handle_p;
PjcmdResp_t *resp_p;
int64_t num[2]; // Array storing total number of jobs and number of successful jobs
...
resp_p = pjcmd_submit_execute(handle_p);     // Request job submission operation
if (resp_p == NULL) {
    // To error processing
}
pjcmd_get_result(resp_p, &code, &subcode, &detail_p);  // Get operation request results
if (code != 0) {
    // To error processing
}
pjcmd_get_jobresult_num(resp_p, num);        // Get number of successfully submitted jobs
```

    In the above example, the total number of submitted jobs is stored in num[0], and the number of successfully submitted jobs is stored in num[1].

  - Results of individual jobs

    The operation results of individual jobs are found by the pjcmd_get_jobresult_info() function.

```
int code, subcode;
char *detail_p;
PjcmdSubjobid_t *subjobid_p;
...
resp_p = pjcmd_submit_execute(handle_p);               // Request job submission operation
if (resp == NULL) {
    // To error processing
}
pjcmd_get_result(resp_p, &code, &subcode, &detail_p);  // Obtain operation request results
if (code != 0) {
    // To error processing
}
pjcmd_get_jobresult_num(resp_p, num);          // Get total number of submitted jobs and
                                               // number of successfully submitted jobs

for (i = 0; i < num[0]; i++) {
    pjcmd_get_jobresult_info(resp_p, PJCMD_JOBRESULT_ANY,
                                     i, PJCMD_JOBRESULT_CODE, &code);
    pjcmd_get_jobresult_info(resp_p, PJCMD_JOBRESULT_ANY,
                                     i, PJCMD_JOBRESULT_SUB_CODE, &subcode);
    pjcmd_get_jobresult_info(resp_p, PJCMD_JOBRESULT_ANY,
                                     i, PJCMD_JOBRESULT_DETAIL, &detail_p);
    pjcmd_get_jobresult_info(resp_p, PJCMD_JOBRESULT_ANY,
                                     i, PJCMD_JOBRESULT_SUBJOBID, &subjobid_p);

    printf("....<Displaying obtained information and other operations> ...");
}
```

In the above example, the following information on all jobs to be operated are obtained sequentially from response information by the pjcmd_get_jobresult_info() function: result code (code), detailed result code (subcode), detailed result (character string) (detail) and sub job ID (subjobid). Only jobs that were successfully submitted (PJCMD_JOBRESULT_OK) or only jobs that that failed in submission (PJCMD_JOBRESULT_ERR) can also be specified in the second argument.

Information

The pjcmd_get_jobresult_info() function specifies a job index to reference results. Therefore, the number of jobs must be obtained by the pjcmd_get_jobresult_num() function beforehand. If an index that exceeds the number of target jobs is specified, an error occurs.

- Other than job operations

As for operations other than job operations (information acquisition and other operations), operation-specific information can be referenced from response information. The reference method varies depending on the information type. The following example references job information (equivalent to the pjstat command).

```
pjcmd_result_t ret;
PjcmdResp_t *resp_p;

...
resp_p = pjcmd_jobinfo_execute(handle_p);              // Request job submission operation
if (resp_p == NULL) {
    // To error processing
}
pjcmd_get_result(resp_p, &code, &subcode, &detail_p);  // Acquire operation request results
if (code != 0) {
    // To error processing
}
do (ret = pjcmd_jobinfo_read_infogrp(resp_p)) {        // Move to one information group
    if (ret == PJCMD_ERR) {
        if (pjcmd_errcode == PJCMD_ERROR_NODATA) {
            break;
        }
        fprintf(stderr, "%s: Cannot read infogrp\n", CMD_NAME);
        pjcmd_destroy_resp(resp_p);
        pjcmd_destroy_handle(handle_p);
        exit(EXIT_FAILURE);
    }

    pjcmd_jobinfo_print_resp(resp_p, PJCMD_JOBINFO_PRINT_JOBINFO); // Display information
                                                                   // group contents
}
```

Multiple information groups are stored in response information. An information group is a unit for obtaining information. It is equivalent to the display of information for each resource unit or resource group by using the --ru or -rg option, respectively, in the pjstat command. The pointer pointing to the information group that is currently being referenced is included in response information. The pjcmd_jobinfo_read_infogrp() function updates the pointer so that it points to the next information group every time the function is called.
Then, the function displays the summary information and job information for the current information group by using the pjcmd_jobinfo_print_resp() function.

Information

- The above example is equivalent to the information that is displayed when executing the pjstat command as follows.

```
$ pjstat --ru --rg
[ RSCUNIT: unit1 ]     (*) Display unit per resource unit
[ RSCGRP: group1 ]         or resource group is equivalent to information group
JOB_ID     JOB_NAME   ...
2927       jobA       ...

[ RSCUNIT: unit1 ]
[ RSCGRP: group2 ]
JOB_ID     JOB_NAME   ...
2928       jobB       ...
...
```

- The pjcmd_jobinfo_read_jobinfo() function is used to confirm job information one piece at a time. This function enables information on jobs belonging to the current information group to be referenced one piece at a time.

2.2.5.2 Error Information

The following steps are used to detect a command API error.

- Function return value (pjcmd_result_t type or pointer type)
  Whether a function is successful or failed is found by referencing a function return value.
- pjcmd_errcode error code
  If a function fails, a code indicating the details is set.
- Response information
  Operation request results are found by referencing response information details (see "2.2.5.1 Referencing Response Information").
- Detailed error information
  Error information that is more detailed than the above information is found.

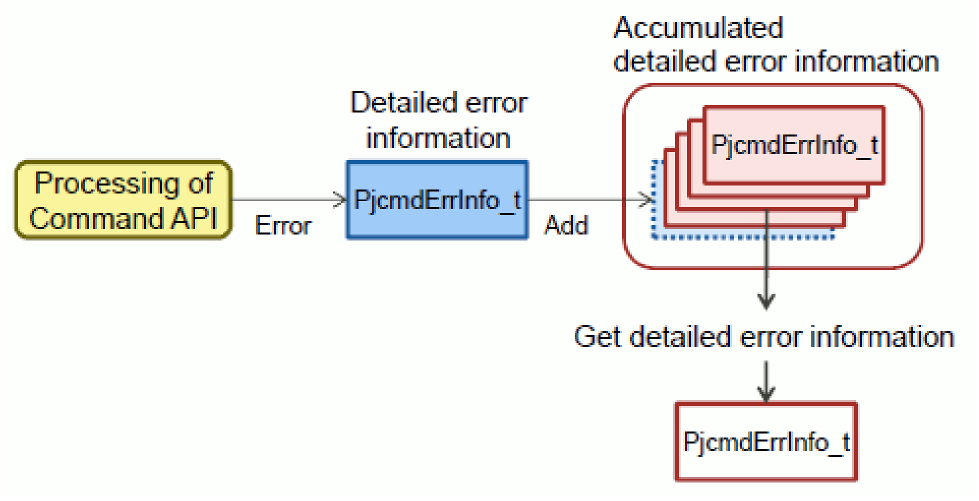
Information

If information that is similar to the information included in a standard command message is required, it is not sufficient to reference only a function return value and error code. Detailed error information also needs to be referenced.

This section describes detailed error information.

Detailed error information (PjcmdErrInfo_t) is accumulated when an error occurs in the command API. Multiple items of detailed error information may be accumulated at one time depending on the error.

Figure 2.1 Conceptual Diagram of Detailed Error Information


Processing of Command API
Error
Detailed error information
PjcmdErrInfo_t
Add
Accumulated detailed error information
PjcmdErrInfo_t
Get detailed error information
PjcmdErrInfo_t

The following information is included in detailed error information.

Table 2.1 Information Included in Detailed Error Information

Information	Description
Detailed error code	This code indicates an error type (pjcmd_suberrcode_t).
Detailed code 1 to 3	Error detailed code (numerical value). A maximum of 3 codes are set based on the error type.
Detailed information (character string) 1 to 5	Detailed error information (character string). A maximum of 5 pieces of information are set based on the error type.
Number of job script line where error occurred	A line number is displayed when an error occurs in a job script.
Sub job ID	If an error related to a specific job occurs, this sub job ID indicates the job ID, bulk number, and step number of the job.

See

The obtained detailed code and detailed information (character string) are different for each detailed error code. For details, see "A.7.2 Detailed Error Code" in "Appendix A Command API Common Reference."

When the user of the command API detects a function error, the user references the detailed error information and, if necessary, displays it as a message.

The following table lists functions for referencing detailed error information.

Table 2.2 Functions for Referencing Detailed Error Information

Function	Description
pjcmd_error_read_errinfo()	This function retrieves a piece of detailed error information accumulated in the command API. The function returns the next piece of detailed error information every time it is called.
pjcmd_error_read_errinfo_by_sjid()	If the detailed error information accumulated in the command API is contents corresponding to a job, the detailed error information corresponding to the specified sub job ID structure is returned.
pjcmd_error_get_info()	This function references information (other than detailed information (character string)) in detailed error information.
pjcmd_error_get_detail_info()	This function references detailed information (character string) in detailed error information.

Function	Description
pjcmd_error_destroy_errinfo()	This function releases (deletes) some detailed error information.
pjcmd_error_clear_errinfo()	This function releases (deletes) all detailed error information accumulated in the command API.

Note

Detailed error information keeps accumulating in memory used by a process. Therefore, normally, detailed error information must be released (deleted) after outputting error information or performing other operations by referencing the detailed error information when an error occurs.
In many cases, a program is designed to terminate when an error occurs. In that case, the OS releases detailed error information when a program ends without intentionally releasing detailed error information.

[Example of Referencing Detailed Error Information]

The following example shows code to reference detailed error information in cases where an error occurs in a function of the command API.

```
main(int argc, char **argv)
{
    PjcmdHandle_p *handle_p;
    PjcmdResp_p *resp_p;
    int code, subcode;
    char *detail_p;
    ...
    ret = pjcmd_submit_parse_pjsub_args(handle_p, argc, argv);
    if (ret != PJCMD_OK) { // Parameter setting error
        goto l_err;
    }
    ...
    resp_p = pjcmd_submit_execute(handle_p);
    if (resp_p == NULL) { // Submission request failure
        goto l_err;
    }

    if (pjcmd_get_result(resp_p, &code, &subcode, &detail_p) != PJCMD_OK) {
        goto l_err;  // Failure in getting results
    }

    if (code != 0) { // Requested operation does not end successfully
        goto l_err;
    }

...

l_err:
    mycmd_print_error();  // Display error message
    pjcmd_error_destroy_errinfo();
    pjcmd_destroy_resp(resp_p);
    pjcmd_destroy_handle(handle_p);
    ...
    exit(1);
}

void mycmd_print_error(void)
{
    PjcmdErrInfo_t *einfo_p;
    int suberrcode, code1, code2, code3, line;
    char *detail_p[5];
    PjcmdSubjobid_t *subjobid_p;
43
44      while ((errinfo_p = pjcmd_error_read_errinfo()) != NULL) {
45          pjcmd_error_get_info(einfo_p, PJCMD_ERRINFO_SUBERRCODE, &suberrcode);
46          switch (suberrcode) {
47          case PJCMD_SUBERR_UNKNOWN_OPT: // Detect unknown option
48              fprintf(stderr, "[ERR.] 0001 mycmd Unknown option '%s'\n",
49                              pjcmd_error_get_detail_info(einfo_p, 0)); // Unknown option
50              break;
51          case PJCMD_SUBERR_COMBINATION: // Incorrect option combination
52              detail_p[0] = pjcmd_error_get_detail_info(einfo_p, 0); // Option 1
53              detail_p[1] = pjcmd_error_get_detail_info(einfo_p, 1); // Option 2
54             fprintf(stderr, "[ERR.] 0002 mycmd Invalid combination of options: '%s' and '%s'\n",
55                               detail_p[0], detail_p[1]);
56              break;
57          ...
58          case PJCMD_SUBERR_UPPER_LIMIT: // Specified resource amount exceeds upper limit
59              pjcmd_error_get_info(einfo_p, PJCMD_ERRINFO_SJID, &subjobid_p);
60              detail_p[0] = pjcmd_error_get_detail_info(einfo_p, 0); // Resource name
61              detail_p[1] = pjcmd_error_get_detail_info(einfo_p, 1); // Specified resource amount
62              detail_p[2] = pjcmd_error_get_detail_info(einfo_p, 2); // Upper limit value
63              fprintf(stderr, "[ERR.] 0057 mycmd %s=%s is greater than the upper limit (%s).\n",
64                              detail_p[0], detail_p[1], detail_p[2]);
65              break;
66          ...
67          case PJCMD_SUBERR_DAEMON_ISNOT_PRESENT: // Job manager function
68                                                  // not working/cannot communicate
69              fprintf(stderr, "[ERR.] 0090 mycmd Job manager does not work\n");
70              break;
71          ...
72          }
73      }
74  }
```

- Lines 10, 15, 19, and 23
  The occurrence of an error in a function of the command API causes a jump to an error process.

- Line 29
  The error message display function is called based on detailed error information.

- Lines 30 to 34
  Terminate once detailed error information, response information, and handles are released after displaying an error message.

- Lines 44 to 73
  Detailed error information PjcmdErrInfo_t is read one piece at a time.

- Line 45
  A detailed error code is obtained from detailed error information.

- Lines 46 to 72
  Information related to all detailed error codes are obtained to display them as a message.
  For details on detailed error code types and related information types, see "A.7.2 Detailed Error Codet" in "Appendix A Command API Common Reference."

  For example, in lines 58 to 65, an error is processed when the specified resource amount exceeds the upper limit defined by the job ACL function. The resource name, specified resource amount, and upper limit value are obtained from detailed error information by the pjcmd_error_get_detail_info() function and displayed by the fprintf() function.

2.2.6 Releasing Handles, Command Line Parsers, and Response Information

Generated handles, command line parsers, and response information must be released when a program ends after processing is completed.

```
PjcmdHandle_t *handle_p;
PjcmdResp_t *resp_p;
PjcmdParser_t *parser_p;
...
parser_p = pjcmd_submit_create_pjsub_parser(handle_p);
...
resp_p = pjcmd_submit_execute(handle_p);
...
pjcmd_submit_destroy_pjsub_parser(parser_p);
pjcmd_destroy_resp(resp_p);
pjcmd_destroy_handle(handle_p);
```

Note

When releasing handles, response information, and command line parsers, the handles must be released last. Response information and command line parsers are associated with handles. Therefore, if response information and command line parsers are operated after the release of the handles, operation is undetermined. Since these areas are part of the data area of a program, if the program ends without releasing the areas, the areas are released by the OS.

2.2.7 Precaution When Using the Command API

- The command API can be used for multithread programs. However, a single handle must not be updated on multiple threads simultaneously.

2.3 Creating a Command

The command API library is located at the following location on each login node, compute cluster management node, and system management node.

```
/usr/lib64/libpjcmd.so
```

A link to the libpjcmd library of the command API must be created when creating an executable file by compiling a created source file.

```
$ gcc -o mycmd mycmd.c -lpjcmd ...
```

Note

The standard gcc available on the OS needs to be the compiler used. Other compilers are not supported.

2.4 Setting the Command API

The administrator can configure settings to change the operation of the command API.
To change a default setting value, the administrator must edit the pmpjcmd.conf file on the login node, compute cluster management node, and system management node that can be used by the command API.

```
/etc/opt/FJSVtcs/pjm/pmpjcmd.conf
```

The contents of the pmpjcmd.conf configuration file are updated to the latest contents every time that a command that uses a function of the command API is executed. The command API reads this configuration file on the node from which it is called. If each node needs to have the same setting values, the configuration file on each node must have the same contents.

The pmpjcmd.conf file permissions must be set as follows:

- Owner/Group: root/root
- File mode: 0644

See

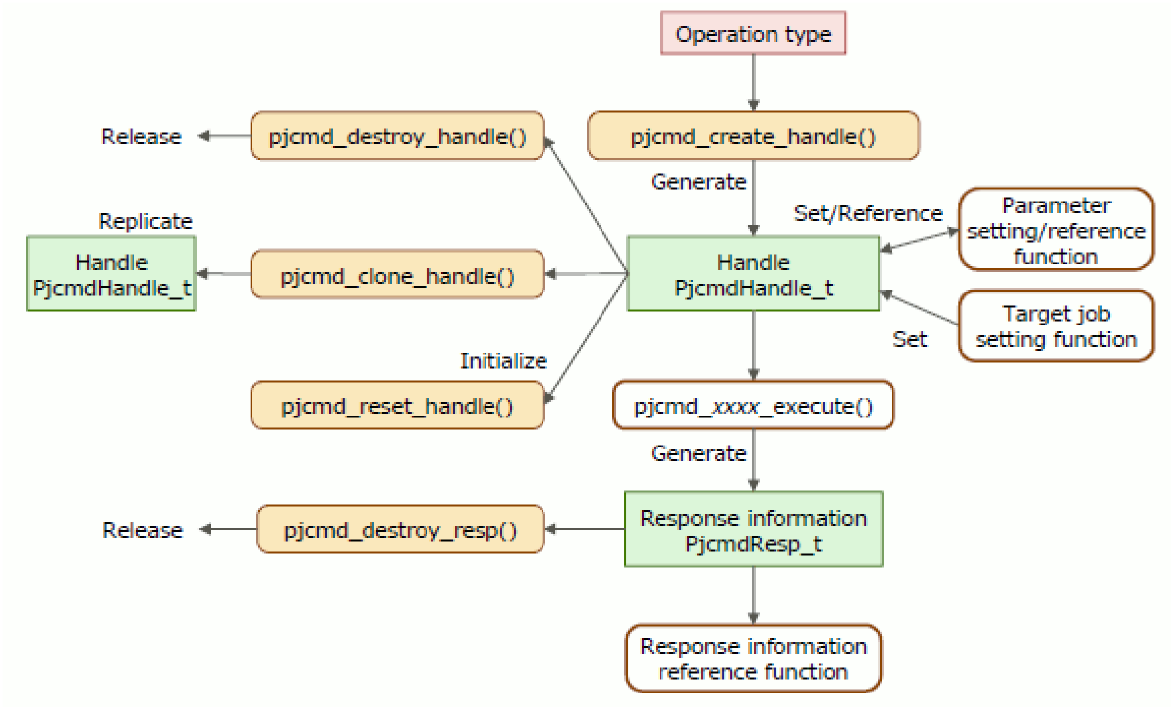
For details on the configuration file, see "Command API settings" in "Chapter 3 Job Operation Management Function Settings" in "Job Operation Software Administrator's Guide for Job Management."

Appendix A Command API Common Reference

A.1 Handle Operations and Response Information

This section describes the functions for handle operations and response information.

Figure A.1 Handle Operations and Response Information



A.1.1 pjcmd_create_handle()

```
PjcmdHandle_t *pjcmd_create_handle(pjcmd_operation_t op)
```

This function generates a handle.

[ARGUMENTS]

op

Identifier indicating an operation type

Identifier	Description
PJCMD_SUBMIT	Submitting a job
PJCMD_KILL	Delete a job
PJCMD_SIGNAL	Sending a signal
PJCMD_HOLD	Holding a job
PJCMD_RELEASE	Releasing a job hold
PJCMD_ALTER	Changing a job parameter
PJCMD_WAIT	Waiting for a job to complete
PJCMD_JOBINFO	Getting job information
PJCMD_RSCINFO	Getting resource information on a resource unit/resource group
PJCMD_LIMITINFO	Getting limit value information

Identifier	Description
PJCMD_RSCSTAT	Getting the resource status
PJCMD_JOBACL	Getting job ACL information
PJCMD_SETPJMSTAT	Changing the job operation (setting job submission/execution permission)
PJCMD_GETPJMSTAT	Getting the job operation setting status (getting the status of job submission/execution permission)

[RETURN VALUE]

The pointer to the generated handle is returned. If the function fails, NULL is returned, and the cause is set in pjcmd_errcode. The caller must release the generated handle by using the pjcmd_destroy_handle() function.

[pjcmd_errcode]

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *op.*

PJCMD_ERROR_NOMEM

Memory acquisition failed.

A.1.2 pjcmd_clone_handle()

```
PjcmdHandle_t *pjcmd_clone_handle(const PjcmdHandle_t *handle_p)
```

This function replicates the *handle_p* handle. The function is used in situations such as submitting a job using the parameters of another job.

[ARGUMENTS]

handle_p

Pointer to the handle to be replicated

[RETURN VALUE]

The pointer to the replicated handle is returned. If the function fails, NULL is returned, and the cause is set in pjcmd_errcode. The caller must release the replicated handle by using the pjcmd_destroy_handle() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

handle_p is invalid (NULL).

PJCMD_ERROR_NOMEM

Memory acquisition failed.

A.1.3 pjcmd_reset_handle()

```
pjcmd_result_t pjcmd_reset_handle(PjcmdHandle_t *handle_p)
```

This function initializes the handle contents. The handle then has the same contents as in the initial state with contents generated by the pjcmd_create_handle() function.

[ARGUMENTS]

handle_p

Pointer to the handle to be initialized

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

handle_p is invalid (NULL).

A.1.4 pjcmd_destroy_handle()

```
pjcmd_result_t pjcmd_destroy_handle(PjcmdHandle_t *handle_p)
```

This function releases a handle.

[ARGUMENTS]

handle_p

Pointer to the handle to be released

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

handle_p is invalid (NULL).

A.1.5 pjcmd_destroy_resp()

```
pjcmd_result_t pjcmd_destroy_resp(PjcmdResp_t *resp_p)
```

This function releases response information. Response information is result information returned by the pjcmd_*operation*_execute() function.

[ARGUMENTS]

resp_p

Pointer to the response information to be released

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

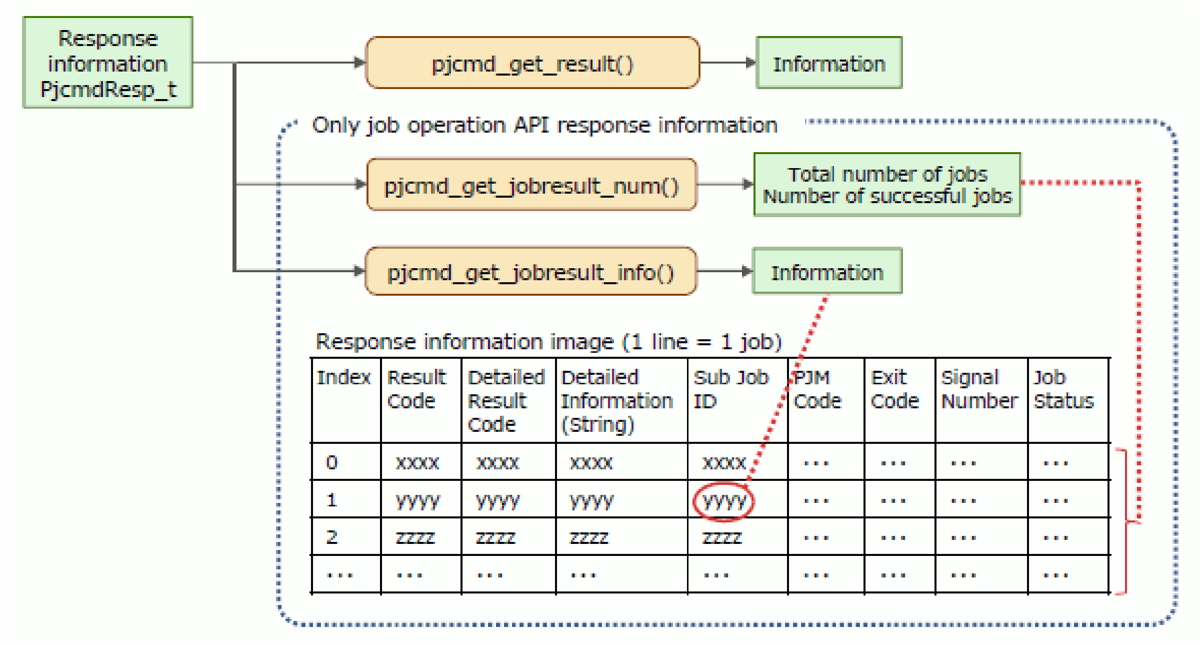
PJCMD_ERROR_INVALID_HANDLE

resp_p is invalid (NULL).

A.2 Referencing of Operation Results

This section describes the functions for referencing result information from response information about job operations and information acquisition.

Figure A.2 Referencing Operation and Information Acquisition Results


Response information PjcmdResp_t
pjcmd_get_result()
Information
Only job operation API response information
pjcmd_get_jobresult_num()
Total number of jobs
Number of successful jobs
pjcmd_get_jobresult_info()
Information
Response information image (1 line = 1 job)

Index	Result Code	Detailed Result Code	Detailed Information (String)	Sub Job ID	PJM Code	Exit Code	Signal Number	Job Status
0	xxxx	xxxx	xxxx	xxxx	...	...	...	...
1	yyyy	yyyy	yyyy	yyyy	...	...	...	...
2	zzzz	zzzz	zzzz	zzzz	...	...	...	...
...	...	...	...	...	...	...	...	...

A.2.1 pjcmd_get_result()

```
pjcmd_result_t pjcmd_get_result(const PjcmdResp_t *resp_p, int *code_p, int *subcode_p, char
**detail_pp);
```

This function gets operation result codes and detailed information from response information.
The information that is obtained by this function is the results of operation requests and information acquisition requests. The pjcmd_get_jobresult_info() function must be used to reference the individual results of job operations (submit a job, delete a job, hold a job, release a job hold, send signals, wait for job completion, and change job parameters).

[ARGUMENTS]

resp_p

Pointer to response information

code_p

A result code is stored in **code_p*. If **code_p* is 0, it indicates that the operation completed successfully. If the result code is not 0, it indicates that the operation did not complete successfully. In that case, detailed investigation data is stored in (int)**subcode_p* and (char *) **detail_pp*.

subcode_p

A result code for detailed investigation data is stored in **subcode_p*.

detail_pp

A pointer (character string) to detailed investigation data is stored in **detail_pp*.

[RETURN VALUE]

PJCMD_OK

Result information has been successfully obtained.

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

resp_p is invalid (NULL).

PJCMD_ERROR_INVALID_ARGUMENT

code_p, *subcode_p*, or *detail_pp* is invalid (NULL).

A.2.2 pjcmd_get_jobresult_num()

```
pjcmd_result_t pjcmd_get_jobresult_num(const PjcmdResp_t *resp_p, int64_t *num_p)
```

This function gets the total number of target jobs and the number of successful jobs from response information about job operations (*).
(*) Job submission, job deletion, job hold, releasing job hold, signal transmission, changing parameters, and waiting for a job to complete (see "Appendix B Job Operation API Reference.")

[ARGUMENTS]

resp_p

Pointer to response information

num_p

The total number of jobs and the number of operation-succeeded jobs are stored in the *num_p*[] array.
Num_p[0]:[0]: Total number of jobs
num_p[1]: Number of successful jobs

The caller needs to prepare these array areas.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- The response information is not response information for the job operation API.

PJCMD_ERROR_INVALID_ARGUMENT

num_p is invalid (NULL).

A.2.3 pjcmd_get_jobresult_info()

```
pjcmd_result_t pjcmd_get_jobresult_info(const PjcmdResp_t *resp_p, pjcmd_jobresult_type_t result,
int64_t indx,
 pjcmd_jobresult_info_t type, void *val_p)
```

This function gets result information of specific jobs from response information about job operations (*).
(*) Submit a job, delete a job, hold a job, release a job hold, send signals, change parameters, and wait for job completion (See "Appendix B Job Operation API Reference.")

[ARGUMENTS]

resp_p

Pointer to response information

result

Conditions for target jobs whose results are obtained

PJCMD_JOBRESULT_ANY

All jobs are targeted.

PJCMD_JOBRESULT_OK

Only successful jobs are targeted.

PJCMD_JOBRESULT_ERR

Only failed jobs are targeted.

indx

Index of jobs whose result information is obtained. The value must be in a range of 0 to a value that is calculated by subtracting 1 from the total number of jobs that meet the conditions indicated by the *result* argument.

type

Identifier of result information to be obtained (See the following table.)

val_p

Result information is stored in **val_p*. The caller needs to prepare an area of a sufficient size based on *type*. For example, if **val_p* type is int, the caller needs to prepare an int type area and specify a pointer (int *) to the area in *val_p*.

type	**val_p*	Type of **val_p*
PJCMD_JOBRESULT_SUBJOBID	Sub job ID structure of a job If the obtained value is referenced after the release of response information, operation is undetermined.	PjcmdSubjobid_t *
PJCMD_JOBRESULT_CODE	Job operation result code 0: Success Other than 0: Failure	int
PJCMD_JOBRESULT_SUB_CODE	Detailed result code when a job operation fails This is investigation data from the error occurrence time.	int
PJCMD_JOBRESULT_DETAIL	Detailed information (character string) when a job operation fails This is investigation data from the error occurrence time.	char *
PJCMD_JOBRESULT_PJM_CODE	Job completion code (PJM code) This identifier has meaning only when waiting for a job to complete.	int32_t
PJCMD_JOBRESULT_EXIT_CODE	Job script exit code This identifier has meaning only when waiting for a job to complete.	int32_t
PJCMD_JOBRESULT_SIGNAL_NUM	Signal number when a job script ends with a signal This identifier has meaning only when waiting for a job to complete.	int32_t
PJCMD_JOBRESULT_JOB_STATUS	Job status while waiting 0: Completed step job or bulk job 1: ACCEPT, QUEUED, READY, RUNNING-A, RUNNING-P, RUNNING, or RUNNING-E state 2: HOLD state 3: ERROR state	int32_t

type	**val_p*	Type of **val_p*
	This identifier has meaning only when waiting for a job to complete.	
PJCMD_JOBRESULT_ACCEPT_DATE	Job acceptance time This identifier has meaning only when waiting for a job to complete. If the job type is sub job, the time when the summary job is accepted is returned.	strcut timespec
PJCMD_JOBRESULT_NODE_MODEL	Compute node type 0: FX server 1: PRIMERGY server	uint32_t

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- The response information is not response information for the job operation API.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *result* or *type.*

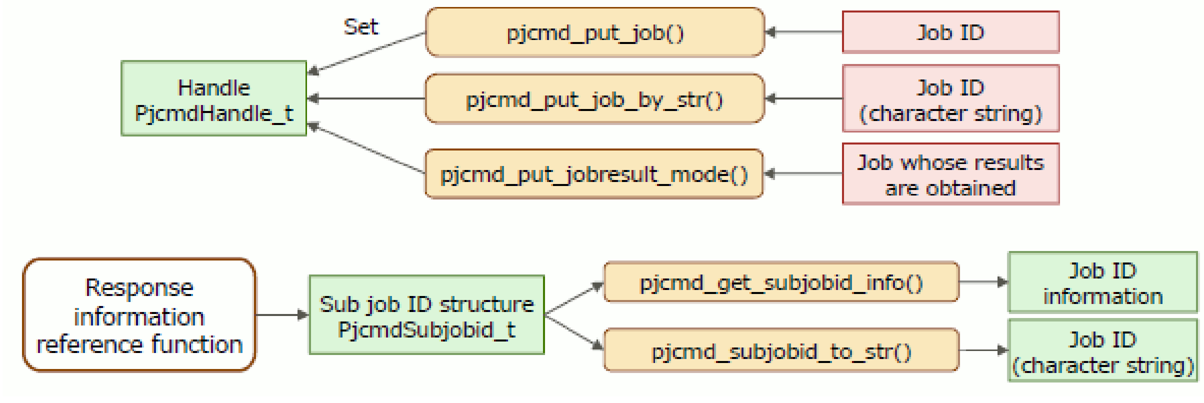
PJCMD_ERROR_NODATA

The *indx* value is out of range.

A.3 Setting and Acquisition of Job IDs

This section describes the functions for operating and referencing job IDs.

Figure A.3 Setting and Referencing Job IDs



A.3.1 pjcmd_put_job()

```
pjcmd_result_t pjcmd_put_job(PjcmdHandle_t *handle_p, const int64_t *jobid_p, int64_t jobid_s,
int64_t jobid_e,
 const int64_t *no_p, int64_t no_s, int64_t no_e, pjcmd_jobmodel_t model)
```

This function sets a job, which is specified in an argument, in a handle. If the job is already set in the handle, an additional setting is made.

[ARGUMENTS]

handle_p

Pointer to a handle

jobid_p

Jobid_p[] is an array composed of one or more job IDs. The last element in the array must be -1. If the array does not end with -1, operation is undetermined. If *jobid_p* is NULL and no job ID is specified in the array elements (*jobid_p*[0] is -1), this argument is ignored.

jobid_s, *jobid_e*

This argument indicates the start (*jobid_s*) and end (*jobid_e*) of a job ID range.
If *jobid_s* is larger than *jobid_e,* the function returns the PJCMD_ERR error, and PJCMD_ERROR_INVALID_PARAM is set in pjcmd_errcode.
If a job ID is specified in *jobid_p* and a value outside the range (0 to 2147483647; where a job ID can be specified), this argument is ignored.

no_p

If the job model is a step job or bulk job, the step number or bulk number is specified as an array. The last element in the array must be -1. If the array does not end with -1, operation is undetermined. If *no_p* is NULL, more than two job IDs are specified in *jobid_p*, or PJCMD_JOBMODEL_NORMAL is specified in *model*, this argument is ignored.
If no step number or bulk number is specified in an array element (*no_p*[0] is -1), the function returns the PJCMD_ERR error, and PJCMD_ERROR_INVALID_PARAM is set in pjcmd_errcode.

no_s, *no_e*

If the job model is a step job or bulk job, the start (*no_s*) and end (*no_e*) of the step number or bulk number is specified.
If *no_s* is larger than *no_e,* the function returns the PJCMD_ERR error, and PJCMD_ERROR_INVALID_PARAM is set in pjcmd_errcode.
If -1 is specified in *no_s* and *no_e*, a step number or bulk number is indicated by *no_p*, two or more job IDs are specified in *jobid_p*, or PJCMD_JOBMODEL_NORMAL is specified in *model*, this argument is ignored.

model

If the *no_p*, *no_s*, and *no_e* arguments are specified, *model* is specified in order to differentiate that the argument is a step number or bulk number.

PJCMD_JOBMODEL_STEP

Valid values that are specified in the *no_p*, *no_s*, and *no_e* arguments are regarded as step numbers.

PJCMD_JOBMODEL_BULK

Valid values that are specified in the *no_p*, *no_s*, and *no_e* arguments are regarded as bulk numbers.

If a job model other than the above (PJCMD_JOBMODEL_NORMAL, PJCMD_JOBMODEL_MSWK) is specified in the *model* argument, or if invalid values are set in the *no_p*, *no_s*, and *no_e* arguments, it is regarded that sub job IDs are not specified, and only the *jobid_p, jobid_s*, and *jobid_e* arguments that specify job IDs are valid.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This handle type cannot specify a job ID.

PJCMD_ERROR_INVALID_PARAM

A parameter specified in an argument is invalid.

- *jobid_s* > *jobid_e*
- *no_p*[0] is -1.
- *no_s* > *no_e*

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified to the argument *model*.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

Information

The arguments shown below need to be set based on the job type to be set. Argument that are not shown below are ignored.

- Specifying one job ID (100)

```
jobid_p[0]=100;
jobid_p[1]=-1;
no_p=NULL;
no_s=-1;
```

- Specifying N job IDs (100,101, ...)

```
jobid_p[0]=100;
jobid_p[1]=101;
...
jobid_p[N-1]=100+N-1;
jobid_p[N]=-1;
```

- Specifying a job ID range (100 to 110)

```
jobid_p=NULL;
jobid_s=100;
jobid_e=110;
```

- Specifying one sub job ID (100_1 or 100[1]) of a step job or bulk job

```
jobid_p[0]=100;
jobid_p[1]=-1;
no_p[0]=1;
no_p[1]=-1;
model=PJCMD_JOBMODEL_STEP or PJCMD_JOBMODEL_BULK
```

- Specifying N sub job IDs of step jobs or bulk jobs (100_0,100_1,... or 100[0],100[1],...)

```
jobid_p[0]=100;
jobid_p[1]=-1;
```

```
no_p[0]=0;
no_p[1]=1;
...
no_p[N-1]=N-1;
no_p[N]=-1;
model=PJCMD_JOBMODEL_STEP or PJCMD_JOBMODEL_BULK
```

- Specifying the sub job ID range of step jobs or bulk jobs (100_0 to 10 or 100 [0 to 10])

```
jobid_p[0]=100;
jobid_p[1]=-1;
no_p=NULL;
no_s=0;
no_e=10;
model=PJCMD_JOBMODEL_STEP or PJCMD_JOBMODEL_BULK
```

A.3.2 pjcmd_put_job_by_str()

```
pjcmd_result_t pjcmd_put_job_by_str(PjcmdHandle_t *handle_p, const char *str_p)
```

This function sets a job ID, which is specified using a character string, in a handle. If a job is already set in the handle, the job ID is added to it.

[ARGUMENTS]

handle_p

Pointer to a handle

str_p

Job ID or sub job ID (represented by a character string) to be set
A job ID can be specified in the following character string formats;

- Single job ID (Example: "100")
- Single sub job ID (Example: "200_1" or "300[5]")
- Specifying a job ID range (Example: "10 to 50")
- Specifying a sub job ID range (Example: "200_1 to 200_5" or "300 [5 to 10]")

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This handle type cannot specify a job ID.

PJCMD_ERROR_INVALID_ARGUMENT

str_p is NULL.

PJCMD_ERROR_INVALID_PARAM

The format of a job ID specified in *str_p* is invalid.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

A.3.3 pjcmd_put_jobresult_mode()

```
pjcmd_result_t pjcmd_put_jobresult_mode(PjcmdHandle_t *handle_p, pjcmd_jobresult_mode_t mode)
```

This function sets a target job range for getting operation results from the job operation management function in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

mode

Mode to get job operation results

PJCMD_JOBRESULT_MODE_ALL

If job IDs are specified as a range, jobs that do not exist are also included in the results to obtain.

PJCMD_JOBRESULT_MODE_BASIC

If job IDs are specified as a range, jobs that do not exist, jobs without the operation privilege, and jobs that cannot be operated are not included.

If this function is not called, PJCMD_JOBRESULT_MODE_ALL is applied as the mode to get job operation results.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job operations.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is set in *mode.*

PJCMD_ERROR_INTERNAL

Internal error

A.3.4 pjcmd_get_subjobid_info()

```
pjcmd_result_t pjcmd_get_subjobid_info(const PjcmdSubjobid_t *subjobid_p, pjcmd_subjobid_info_t
info, void *val_p)
```

This function obtains the job ID, step number, and bulk number of a sub job ID structure.

[ARGUMENTS]

subjobid_p

Pointer to a sub job ID structure

info

Identifier indicating the information to be obtained (See the following table.)

val_p

The obtained values are stored in **val_p*. The caller needs to prepare an area of a sufficient size based on *info.* For example, if * *val_p* type is int64_t, the caller needs to prepare an int64_t type area and specify a pointer (int64_t *) to the area in *val_p*.

info	**val_p*	Type of **val_p*
PJCMD_SUBJOBID_JOBID	Job ID	int64_t
PJCMD_SUBJOBID_STEPNO	Step number If the job is not a step job, -1 is stored.	int64_t
PJCMD_SUBJOBID_BULKNO	Bulk number If the job is not a bulk job, -1 is stored.	int64_t

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

subjobid_p or *val_p* is NULL.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *info*.

A.3.5 pjcmd_subjobid_to_str()

```
pjcmd_result_t pjcmd_subjobid_to_str(const PjcmdSubjobid_t *subjobid_p, char *str_p)
```

This function converts a sub job ID structure to a character string for display.

[ARGUMENTS]

subjobid_p

Pointer to a sub job ID structure

str_p

A converted character string is stored in the area indicated by *str_p*. The caller needs to reserve the area. The maximum length for a stored character string is PJCMD_MAX_SUBJOBID_STR_LEN bytes including the NULL character at the end.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

subjobid_p or *str_p* is NULL.

[EXAMPLES]

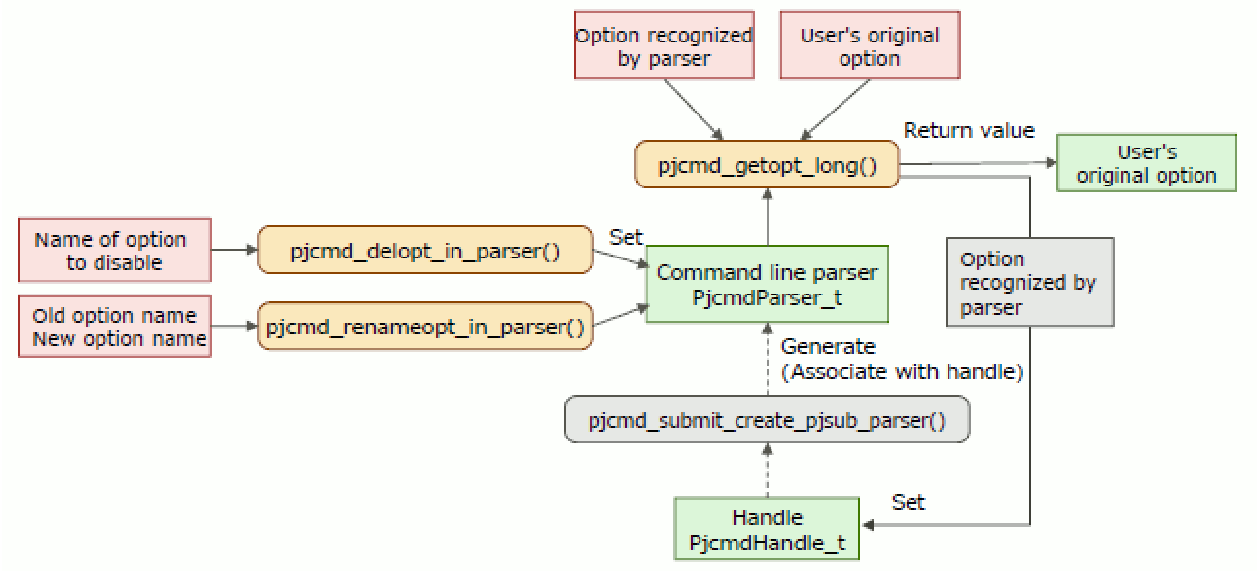
A sub job ID structure is converted as follows based on the job model:

- Normal job: "*<JobID>*" (Example: "123")
- Step job: "*<JobID>*_*<Step Number>*" (Example: "123_5")
- Bulk job: "*<JobID>*[*<Bulk Number>*]" (Example: "123[5]")

A.4 Analysis of Command Line Arguments

This section describes the functions for analyzing command line arguments.

Figure A.4 Analysis of Command Line Arguments


Option recognized by parser
User's original option
pjcmd_getopt_long()
Return value
User's original option
Name of option to disable
pjcmd_delopt_in_parser()
Set
Command line parser PjcmdParser_t
Option recognized by parser
Old option name New option name
pjcmd_renameopt_in_parser()
Generate (Associate with handle)
pjcmd_submit_create_pjsub_parser()
Handle PjcmdHandle_t
Set

A.4.1 pjcmd_getopt_long()

```
int pjcmd_getopt_long(const PjcmdParser_t *parser_p, int argc, char *const argv[], const char
*optstring_p, const struct option *longopts_p, int *longindex_p)
```

This function analyzes command line arguments based on the command line parser.
The operation of this function is the same as for the getopt_long() function, except for the following.

- This function recognizes the following options:
  - *optstring_p* (short option) and *longopts_p* (long option) that are specified by the caller
  - Options recognized by the command line parser *parser_p*
- The function returns every time that one option (*optstring_p* or *longopt_p*) specified by the caller is detected. When detecting an option that is recognized by the command line parser *parser_p*, the function does not return. However, a parameter is internally set in the handle (the handle specified when generating the command line parser) corresponding to the command line parser.

If NULL is specified in *parser_p*, the operation is the same as for the getopt_long() function.

[ARGUMENTS]

parser_p

Pointer to a command line parser

argc

Number of arguments to be analyzed

argv

Array of arguments to be analyzed

optstring_p

Short option character string (see getopt_long(3))

longopts_p

Long option to accept (see getopt_long(3))

longindex_p

Index to a recognized long option (see getopt_long(3))

[RETURN VALUE]

- If a short option is detected, a character is returned.
- If a long option is detected, a value is returned based on the corresponding member flag in *longopts_p*.
- When analysis of a command line option is completed, -1 is returned.
- If an unidentified option is detected, "?" is returned.

A.4.2 pjcmd_delopt_in_parser()

```
pjcmd_result_t pjcmd_delopt_in_parser(PjcmdParser_t *parser_p, char opt, const char *longopt_p)
```

This function disables the specified options among the options that are recognized by a command line parser. Calling this function once can disable either the short or long option. The caller needs to call this function multiple times to disable multiple options.

[ARGUMENTS]

parser_p

Pointer to a command line parser

opt

Short option to be disabled (character string excluding "-" in the option). If the NULL character "\0" is specified, it is regarded that a disabling short option is not specified.

longopt_p

Name of the long option to be disabled (option name excluding "--"). If NULL is specified, it is regarded that a disabling long option is not specified.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

parser_p is NULL.

PJCMD_ERROR_UNKNOWN_OPTION

A specified option does not exist in the command line parser.

A.4.3 pjcmd_renameopt_in_parser()

```
pjcmd_result_t pjcmd_renameopt_in_parser(PjcmdParser_t *parser_p, char old_opt, char new_opt, const
char *old_longopt_p, const char *new_longopt_p)
```

This function changes option names that are recognized by a command line parser.

[ARGUMENTS]

parser_p

Pointer to a command line parser

old_opt

Short option before the change (character string excluding "-" in the option). If a short option is not changed, the NULL character "\0" must be specified.

new_opt

Short option after the change (character string excluding "-" in the option). If *old_opt* is a NULL character, it is ignored.

old_longopt_p

Long option after the change (option name excluding "--"). If a long option is not changed, NULL must be specified.

new_longopt_p

Long option after the change (option name excluding "--"). If *old_longopt_p* is NULL, it is ignored.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

parser_p is NULL.

PJCMD_ERROR_UNKNOWN_OPTION

A specified option does not exist in the command line parser.

PJCMD_ERROR_INVALID_OPTION

The specification of the changed option is invalid.

- It duplicates other options that are not changed.
- When specifying the option *old_opt* or *old_longopt_p* before the change, a NULL character is specified in the changed option *new_opt*, or NULL is specified in *new_longopt_p*.

Note

- This function just changes an option name. The function cannot change the specification that defines whether an argument is necessary or not for an option. To disable the function, the pjcmd_delopt_in_parser() function must be used.
- A short option cannot be changed to a long option, and vice versa.

A.5 Display of Usage

This section describes the functions for displaying usage of a command provided by the job operation management function.

Figure A.5 Displaying Usage of a Command Provided by the Job Operation Management Function

Command type
pjcmd_print_stdcmd_usage()
Displays usage

A.5.1 pjcmd_print_stdcmd_usage()

```
pjcmd_result_t pjcmd_print_stdcmd_usage(pjcmd_stdcmd_t cmd, const char *cmdname_p)
```

This function outputs usage of a command provided by the job operation management function.

[ARGUMENTS]

cmd

Identifier of a command provided by the job operation management function

Identifier	Descripton
PJCMD_STDCMD_PJSUB	pjsub command
PJCMD_STDCMD_PJDEL	pjdel command
PJCMD_STDCMD_PJHOLD	pjhold command
PJCMD_STDCMD_PJRLS	pjrls command
PJCMD_STDCMD_PJSIG	pjsig command
PJCMD_STDCMD_PJWAIT	pjwait command
PJCMD_STDCMD_PJALTER	pjalter command
PJCMD_STDCMD_PMALTER	pmalter command
PJCMD_STDCMD_PJSTAT	pjstat command
PJCMD_STDCMD_PJACL	pjacl command
PJCMD_STDCMD_PMPJMOPT	pmpjmopt command
PJCMD_STDCMD_PJSHOWRSC	pjshowrsc command

cmdname_p

Command name displayed in usage. If NULL is specified, the name of a command provided by the job operation management function is displayed.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

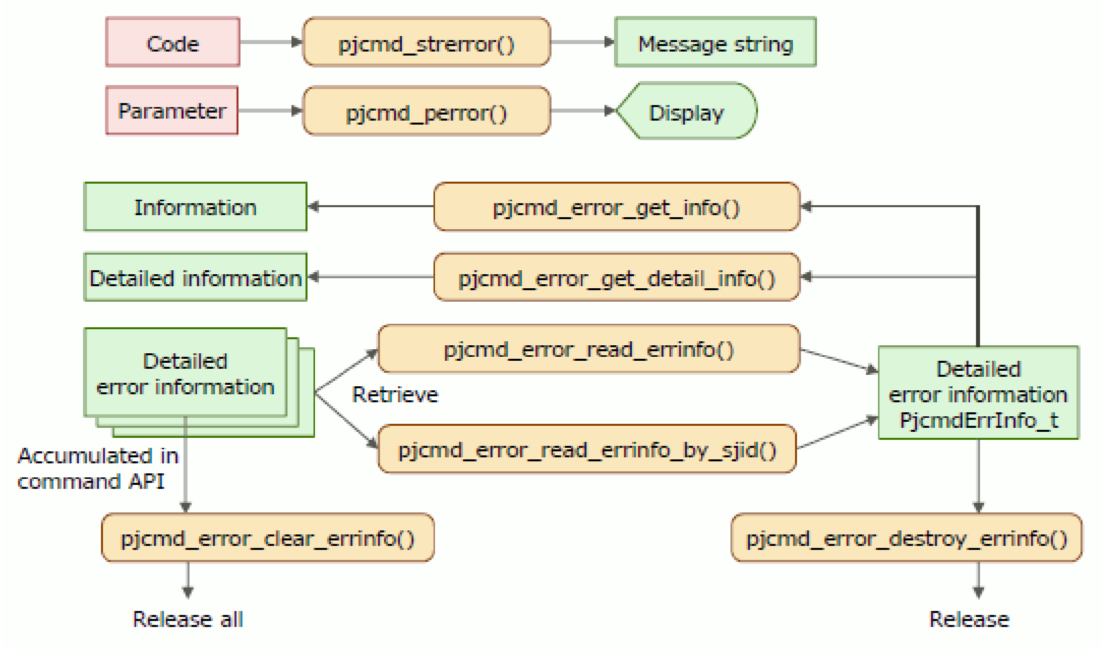
PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *cmd*.

A.6 Error-related Functions

This section describes the functions for handling error information from the command API.

Figure A.6 Referencing Error Information


Code
pjcmd_strerror()
Message string
Parameter
pjcmd_perror()
Display
Information
pjcmd_error_get_info()
Detailed information
pjcmd_error_get_detail_info()
Detailed error information
pjcmd_error_read_errinfo()
Retrieve
pjcmd_error_read_errinfo_by_sjid()
Detailed error information PjcmdErrInfo_t
Accumulated in command API
pjcmd_error_clear_errinfo()
pjcmd_error_destroy_errinfo()
Release all
Release

A.6.1 pjcmd_strerror()

```
char *pjcmd_strerror(pjcmd_error_t code)
```

This function returns a standard error message of the command API corresponding to the error code specified in an argument.

[ARGUMENTS]

code

Error code. For details on the values that can be specified, see "A.7.3 pjcmd_errcodejcmd_errcode Variable."

[RETURN VALUE]

The pointer to one of the following fixed character strings is returned based on *code.*

Error code	Message
PJCMD_SUCCESS	Succeeded
PJCMD_ERROR_INVALID_HANDLE	Invalid handle
PJCMD_ERROR_INVALID_RESP	Invalid response data
PJCMD_ERROR_INVALID_ARGUMENT	Invalid argument
PJCMD_ERROR_UNKNOWN_OPTION	Unknown option
PJCMD_ERROR_INVALID_OPTION	Invalid option
PJCMD_ERROR_UNKNOWN_PARAM	Unknown parameter
PJCMD_ERROR_INVALID_PARAM	Invalid parameter
PJCMD_ERROR_NODATA	No data
PJCMD_ERROR_OPEN	Open failed
PJCMD_ERROR_INVALID_NODE	Invalid node
PJCMD_ERROR_NOPERM	No permission
PJCMD_ERROR_CONNECT	Connection failed
PJCMD_ERROR_NOMEM	Not enough memory

Error code	Message
PJCMD_ERROR_BUSY	Operation is busy
PJCMD_ERROR_TOO_LONG	Too long
PJCMD_ERROR_NOENT	No such file or directory
PJCMD_ERROR_ACCESS	Permission denied
PJCMD_ERROR_SIGNAL	Interrupted system call
PJCMD_ERROR_INTERNAL	Internal error occurred
Other value *n*	Unknown code *n*

A.6.2 pjcmd_perror()

```
void pjcmd_perror(pjcmd_msg_level_t level, const char *compo_p, int id, const char *cmdname_p, const
char *addmsg_p)
```

This function displays a message at the standard error output in the following format in response to the pjcmd_errcode error code value.

```
[level] compo_p id cmdname_p message addmsg_p
```

"message" in the above character string is the character string corresponding to the pjcmd_errcode error code (see "A.6.1 pjcmd_strerror().")

[ARGUMENTS]

level

Message level. One of the following character strings corresponding to a level is output in a message.

level	Corresponding Character String
PJCMD_MSG_INFO	INFO
PJCMD_MSG_NOTICE	NOTE
PJCMD_MSG_WARN	WARN
PJCMD_MSG_ERROR	ERR. (A dot is added to the end.)
PJCMD_MSG_EMERG	EMRG

compo_p

Component name. The name must be five characters or less excluding the NULL character at the end. Any character string can be specified for a function name, etc.

id

Message ID. Any four-digit decimal number can be specified. If the number has fewer than four digits, the high-order digits are padded with zeros.

cmdname_p

Command name displayed in a message. Any character string, such as a program name that is used to call the command API, can be specified.

addmsg_p

Additional message. If NULL is specified, this argument is ignored.

[EXAMPLE]

```
pjcmd_perror(PJCMD_MSG_ERROR, "MYCMD", 1, "mycommand", "(-x)");
```

When pjcmd_errcode is PJCMD_ERROR_UNKNOWN_OPTION, the following message is output.

```
[ERR.] MYCMD 0001 mycommand Unknown option (-x)
```

A.6.3 pjcmd_error_read_errinfo()

```
PjcmdErrInfo_t *pjcmd_error_read_errinfo(void)
```

This function retrieves the following detailed error information from the list of detailed error information accumulated in the command API. The following detailed error information is returned every time the function is called.

[ARGUMENTS]

None

[RETURN VALUE]

Detailed error information.
When there is no detailed error information to be retrieved, NULL is returned, and the PJCMD_ERROR_NODATA error code is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_NODATA

There is no next piece of detailed error information.

A.6.4 pjcmd_error_read_errinfo_by_sjid()

```
PjcmdErrInfo_t *pjcmd_error_read_errinfo_by_sjid(const PjcmdSubjobid_t *sjid_p)
```

This function retrieves detailed error information on the job corresponding to the specified sub job ID structure from the list of detailed error information accumulated in the command API. The function returns the next piece of applicable detailed error information every time it is called.

[ARGUMENTS]

sjid_p

Pointer to a sub job ID structure. This argument specifies a sub job ID structure that is obtained by the pjcmd_get_jobresult_info() function when an error occurs during the job operation.

[RETURN VALUE]

Detailed error information.
If the function fails, NULL is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_NODATA

There is no next detailed error information. There is no detailed error information corresponding to *sjid_p.*

PJCMD_ERROR_INVALID_ARGUMENT

sjid_p is invalid (NULL).

A.6.5 pjcmd_error_get_info()

```
pjcmd_result_t pjcmd_error_get_info(const PjcmdErrInfo_t *errinfo_p, pjcmd_errinfo_type_t type, void
*val_p)
```

This function references a specified information item value from detailed error information.

[ARGUMENTS]

errinfo_p

Pointer to detailed error information. This argument specifies information that is obtained by the pjcmd_error_read_errinfo() function.

type

Identifier of an information item to be referenced (See the following table.)

val_p

A value is stored in **val_p*. The caller needs to prepare an area of a sufficient size based on *type*. For example, if **val_p* type is int, the caller needs to prepare an int type area and specify a pointer (int *) to the area in *val_p*.

type	**val_p*	Type of **val_p*
PJCMD_ERRINFO_SUBERRCODE	Detailed error code This error code indicates an error type provided by detailed error information. For details on a value (PJCMD_SUBERR_*xxxx*) to be stored and its meaning, see "A.7.2 Detailed Error Code."	int
PJCMD_ERRINFO_CODE1 PJCMD_ERRINFO_CODE2 PJCMD_ERRINFO_CODE3	Detailed error information (numerical value) The detailed error code (PJCMD_SUBERR_*xxxx*) that is obtained by specifying PJCMD_ERRINFO_SUBERRCODE in *type* may include up to 3 additional pieces of detailed information (numerical values) ("[CODE *n*]" shown in "A.7.2 Detailed Error Code"). PJCMD_ERRINFO_CODE*n* is specified in *type* when referencing the detailed information. If detailed information (numerical value) is not included in the detailed error code, the value to be obtained does not have any meaning.	int
PJCMD_ERRINFO_SCRIPTNAME	Name of the job script where an error occurred in a job script If the error was not caused by the job script, NULL is set in **val_p*. If the obtained value is referenced after the release of detailed error information, operation is undetermined.	char *
PJCMD_ERRINFO_SCRIPTLINE	Line number of the job script where an error occurred If the error was not caused by the job script, 0 is specified in **val_p*.	int
PJCMD_ERRINFO_SJID	Sub job ID structure of a job with an error The job ID, bulk number, and step number of the job can be known. If the error is not related to the sub job ID, NULL is specified in **val_p*. If the obtained value is referenced after the release of detailed error information, operation is undetermined.	PjcmdSubjobid_t *
PJCMD_ERRINFO_PLACEID	Location where an error occurred (for troubleshooting)	uint64_t

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

errinfo_p or val_p is NULL.

PJCMD_ERROR_UNKNOWN_PARAM

type is invalid.

A.6.6 pjcmd_error_get_detail_info()

```
const char *pjcmd_error_get_detail_info(const PjcmdErrInfo_t *errinfo_p, int indx)
```

This function returns detailed information on the specified index among multiple pieces of detailed information (character strings) included in one piece of detailed error information.

[ARGUMENTS]

errinfo_p

Pointer to detailed error information

indx

Index of detailed information (character strings) included in detailed error information. A maximum of five pieces of detailed information are included depending on the detailed error code. A value from 0 to 4 is specified. For detailed information corresponding to a detailed error code, see "A.7.2 Detailed Error Code."

[RETURN VALUE]

Detailed information (character string).
If detailed information corresponding to *indx* does not exist or if the function fails, this function returns NULL. pjcmd_errcode is used to distinguish between them.

[pjcmd_errcode]

PJCMD_SUCCESS (or RETURN VALUE is NULL)

Detailed information corresponding to *indx* is not included in this detailed error information.

PJCMD_ERROR_UNKNOWN_PARAM

indx exceeds the range that can be specified.

PJCMD_ERROR_INVALID_ARGUMENT

errinfo_p is NULL.

A.6.7 pjcmd_error_destroy_errinfo()

```
void pjcmd_error_destroy_errinfo(PjcmdErrInfo_t *errinfo_p)
```

This function releases specified detailed error information.

[ARGUMENTS]

errinfo_p

Pointer to detailed error information

[RETURN VALUE]

None

A.6.8 pjcmd_error_clear_errinfo()

```
void pjcmd_error_clear_errinfo(void)
```

This function releases all detailed error information accumulated in the command API.

[ARGUMENTS]

None

[RETURN VALUE]

None

A.7 Error Codes, Global Variables, and Constants

This section describes the error codes, global variables, and constants (macro) of the command API.

A.7.1 Result Codes

A result code is the result of calling a command API function.

Value	Description
PJCMD_OK	Succeeded
PJCMD_ERR	Failed

A.7.2 Detailed Error Code

A detailed error code is a value indicating a detailed error information type that is obtained by the pjcmd_error_read_errinfo() function.

Information

Any information accompanying a detailed error code is listed in the following table using the following symbols:

[CODE *n*] Detailed codes 1 to 3 can be obtained by the pjcmd_error_get_info() function.
[DETAIL *n*] Detailed information (character string) 0 to 4 can be obtained by the pjcmd_error_get_detail_info() function.
[JOBID] A job ID (sub job ID structure) can be obtained by the pjcmd_error_get_info() function.

Detailed error code.	Description
PJCMD_SUBERR_UNKNOWN_OPT	An unknown option is detected. [DETAIL 0] Unknown option name
PJCMD_SUBERR_COMBINATION	The combination of options is invalid. [DETAIL 0] Option name 1 [DETAIL 1] Option name 2
PJCMD_SUBERR_UNKNOWN_OPTARG	An option argument is an unknown value. [DETAIL 0] Option name [DETAIL 1] Argument with unknown value
PJCMD_SUBERR_INVALID_OPTARG	A value that cannot be specified in an option argument is specified. [DETAIL 0] Option name [DETAIL 1] Argument or argument = value
PJCMD_SUBERR_NO_PARAM	A parameter is not specified.
PJCMD_SUBERR_ARG_FORMAT_QUOTA	A double quotation mark (") or single quotation mark (') does not match.
PJCMD_SUBERR_INVALID_ARG	An invalid argument is specified. [DETAIL 0] Argument
PJCMD_SUBERR_NO_JOBID	A job ID is not specified.
PJCMD_SUBERR_JOBID_SYNTAX_ERROR	The format of a job ID is invalid. [DETAIL 0] Invalid job ID
PJCMD_SUBERR_JOBID_NOT_EXIST	The job corresponding to the specified job ID does not exist. [JOBID]
PJCMD_SUBERR_JOB_STATE_ERROR	A command cannot be executed in the specified job state. [JOBID]

Detailed error code.	Description
PJCMD_SUBERR_JOB_TYPE_ERROR	There was an attempt to change a job parameter for an interactive job. [JOBID]
PJCMD_SUBERR_JOB_MODEL_ERROR	A sub job ID is specified in a step job when changing a resource unit name.
PJCMD_SUBERR_JOBNAME_MISMATCH	Job names do not match when the sub jobs of a step job are batch submitted. [DETAIL 0] Job script name
PJCMD_SUBERR_DUP_REQUEST	All operation requests have been accepted. [JOBID]
PJCMD_SUBERR_FILE_OPEN	A file failed to open. [CODE 1] errno when file failed to open [DETAIL 0] File name
PJCMD_SUBERR_FILE_NAME_TOO_LONG	A file name is too long. [DETAIL 0] File name
PJCMD_SUBERR_FILE_FORMAT	The format of a file is invalid. [CODE 1] Line number [DETAIL 0] File name
PJCMD_SUBERR_MULTIPLE_SCRIPT	Multiple job scripts are specified.
PJCMD_SUBERR_TOO_MANY_ARG_SCRIPT	Too many options are written in a job script. [DETAIL 0] Names of the options that could not be analyzed
PJCMD_SUBERR_LINE_LENGTH	The number of words exceeds the maximum number of words that can be written in one job script line.
PJCMD_SUBERR_CURRENT_ACCESS	Information on the current directory cannot be obtained.
PJCMD_SUBERR_CURRENT_PATH	The current directory name is invalid. Alternatively, a linefeed code is included in the directory name.
PJCMD_SUBERR_FILE_CREAT_FAIL	The specified file cannot be created. [CODE 1] errno when file could not be created [DETAIL 0] File name
PJCMD_SUBERR_NOT_FOUND_UNAME	The specified user name does not exist. [DETAIL 0] User name
PJCMD_SUBERR_GID_FAILED	A group name cannot be obtained from the specified group ID. [DETAIL 0] Group ID (character string)
PJCMD_SUBERR_CANNOT_USED_OPT	An option that cannot be used is specified. [DETAIL 0] Option name
PJCMD_SUBERR_RESOURCE_ERROR	The specified resource is invalid for the target job. [JOBID] [DETAIL 0] Resource information specified during job parameter change operation
PJCMD_SUBERR_OPERATION_BUSY	A request cannot be executed because another request is being processed. [JOBID]
PJCMD_SUBERR_LOWER_LIMIT	The specified resource amount is below the lower limit. [JOBID] (Information may not exist) [DETAIL 0] Specified resource name [DETAIL 1] Specified resource amount [DETAIL 2] Lower limit resource amount that can be specified
PJCMD_SUBERR_TIME_ERROR	The scheduled time to start executing the specified job is earlier than the current time.

Detailed error code.	Description
	[DETAIL 0] Time ("*YYYYMMDDhhmm*") *YYYY*: Year, *MM*: Month, *DD*: Day, *hh*: Hour, *mm*: Minute
PJCMD_SUBERR_UPPER_LIMIT	The specified resource amount exceeds the upper limit. [JOBID] (Information may not exist) [DETAIL 0] Specified resource name [DETAIL 1] Specified resource amount [DETAIL 2] Upper limit resource amount that can be specified
PJCMD_SUBERR_RSC_DOES_NOT_EXIST	The specified resource does not exist. [DETAIL 0] Specified resource name [DETAIL 1] Specified resource amount
PJCMD_SUBERR_RSC_IS_DISABLED	The specified resource cannot be used. [DETAIL 0] Specified resource name [DETAIL 1] Specified resource amount
PJCMD_SUBERR_STEPNO_OVERFLOWED	The specified step number exceeds the defined range.
PJCMD_SUBERR_ARCH_ERROR	There was an attempt to change the resource unit to which a job was submitted, to a resource unit of a different model.
PJCMD_SUBERR_POLICY_M [PG]	When specifying a virtual node placement policy, the value *m* specified in the *opt1* option must be a multiple of the value *n* specified in the *opt2* option. [DETAIL 0] Option name *opt1* [DETAIL 1] Value *m* [DETAIL 2] Option name *opt2* [DETAIL 3] Value *n* This is an error when specifying a value that is equivalent to -L vnode in the pjsub command = *m* and -P vn-policy = unpack/abs-unpack = *n*.
PJCMD_SUBERR_SYSTEM_CONF_CHANGED	The system configuration changed while processing job acceptance.
PJCMD_SUBERR_RSCNAME_NOT_SPECIFIED	A resource name is not specified.
PJCMD_SUBERR_NO_EXECUTE_PERMISSION	There is no authority to operate.
PJCMD_SUBERR_NO_JOBID_PERMISSION	Operating the specified job is not permitted. [JOBID]
PJCMD_SUBERR_ACCEPT_LIMIT	The number of accepted jobs exceeds the upper limit. [JOBID] (Information may not exist) [DETAIL 0] Name of item that exceeds upper limit "ru-accept": Number of simultaneously accepted batch jobs in resource unit "ru-accept-allsubjob": Number of simultaneously accepted sub jobs of bulk jobs and step jobs in resource unit "ru-accept-bulksubjob": Number of simultaneously accepted sub jobs of bulk jobs in resource unit "ru-accept-stepsubjob": Number of simultaneously accepted sub jobs of step jobs in resource unit "ru-interact-accept": Number of simultaneously accepted interactive jobs in resource unit "rg-accept" : Number of simultaneously accepted batch jobs in resource group "rg-accept-allsubjob": Number of simultaneously accepted sub jobs of bulk jobs and step jobs in resource group "rg-accept-bulksubjob": Number of simultaneously accepted sub jobs of bulk jobs in resource group "rg-accept-stepsubjob": Number of simultaneously accepted sub jobs of step jobs in resource group

Detailed error code.	Description
	"rg-interact-accept": Number of simultaneously accepted interactive jobs in resource group
PJCMD_SUBERR_OUT_OF_RANGE	The specified value exceeds the range defined by the job ACL function. [JOBID] (Information may not exist) [DETAIL 0] Option name
PJCMD_SUBERR_RSCNAME_ILLEGAL	A combination with the default value defined by the job ACL function for the resources that can be specified when submitting a job is invalid. [DETAIL 0] Specified resource name [DETAIL 1] Detailed message
PJCMD_SUBERR_RSC_CANNOT_BE_SPECIFIED	The specified custom resource value is not included in the custom resource types that can be specified. [JOBID] (Information may not exist) [DETAIL 0] Specified custom resource name [DETAIL 1] Specified value [DETAIL 2] Value that can be specified
PJCMD_SUBERR_TOO_MANY_CUSTOMRSC	The number of specified custom resources is excessive.
PJCMD_SUBERR_NOT_FOUND_CSTMRSC	A custom resource corresponding to the specified resource value is not defined. [DETAIL 0] Specified resource name [DETAIL 1] Specified value
PJCMD_SUBERR_GATE_CHECK_ERROR	Job acceptance is denied by the exit function configured by the administrator. [CODE 1] Message ID set at job manager exit (Information may not exist) [DETAIL 0] Message set at job manager exit (Information may not exist)
PJCMD_SUBERR_JOB_TIMEOUT	A job was canceled because a compute resource allocated to a job could not be determined within the specified time in an interactive job. [JOBID]
PJCMD_SUBERR_NOT_LOGIN_NODE	An interactive job was executed on a node other than the login node.
PJCMD_SUBERR_JOB_FAIL	Execution of an interactive job has failed. [JOBID]
PJCMD_SUBERR_JOB_CANCEL	Execution of an interactive job has been canceled. [JOBID]
PJCMD_SUBERR_NOT_SUPPORTED	The specified functions or a combination of the functions is not currently supported. [DETAIL 0] Detailed message
PJCMD_SUBERR_DAEMON_ISNOT_PRESENT	The job operation management function is not operating, or communication with the job operation management function cannot be established.
PJCMD_SUBERR_INTERNAL_ERROR	Internal error [CODE 1 to 3] Investigation data [DETAIL 0 to 4] Investigation data
PJCMD_SUBERR_PERMIT	There is no authority to operate.
PJCMD_SUBERR_NODE_ERROR	The operation is not possible on this node.
PJCMD_SUBERR_SYSFUNC_STOP	The system management function is not operating, or information cannot be obtained from the system management function.
PJCMD_SUBERR_NO_CLSTNAME	A cluster name is not specified.

Detailed error code.	Description
PJCMD_SUBERR_REJECT_OPT	The operation for the specified job has been rejected. [JOBID] [DETAIL 0] Option name [DETAIL 1] Detail message
PJCMD_SUBERR_NOT_EXIST_RSCUNIT	The specified resource unit or the default resource unit defined by the job ACL function does not exist. [DETAIL 0] Resource unit name
PJCMD_SUBERR_NOT_EXIST_RSCGRP	The specified resource group or the default resource group defined by the job ACL function does not exist. [DETAIL 0] Resource group name
PJCMD_SUBERR_NO_SIGNO	A signal is not specified by the signal transmission operation.
PJCMD_SUBERR_NOT_FOUND_GROUP	The specified group does not exist in the job ACL information acquisition operation. [DETAIL 0] Group name
PJCMD_SUBERR_NO_USER_PERMISSION	Displaying information for the specified user or group is not permitted by the job ACL information acquisition API. [DETAIL 0] User name [DETAIL 1] Group name
PJCMD_SUBERR_GROUP_NOT_AUTHORIZED	The job submission API does not permit job submission using the privileges of this group. [DETAIL 0] Group name
PJCMD_SUBERR_NO_EXEC_PERM_OPT	Specifying an option is not permitted. [DETAIL 0] Option name This error occurs when performing an operation that is equivalent to the indicated option name.
PJCMD_SUBERR_NO_EXEC_PERM_JIDOPT	Operating a job or specifying an option is not permitted. [JOBID] [DETAIL 0] Option name This error occurs when performing an operation that is equivalent to the indicated option name.
PJCMD_SUBERR_NO_EXEC_PERM_JID	Operating a job is not permitted. [JOBID]
PJCMD_SUBERR_DAEMON_INTERNAL	Daemon internal error of the job operation management function [CODE 1] Investigation data 1 [CODE 2] Investigation data 2
PJCMD_SUBERR_DAEMON_INTERNAL_RETVAL	Daemon internal error of the job operation management function [CODE 1] Investigation data
PJCMD_SUBERR_SIGNAL	A signal has been received.
PJCMD_SUBERR_CMD_UNAVAILABLE	The requested operation cannot be performed.
PJCMD_SUBERR_NODATA	There is no applicable information. [DETAIL 0] Character string indicating specified information item and other information
PJCMD_SUBERR_DUP_OPT	The same option is specified multiple times. [DETAIL 0] Option name
PJCMD_SUBERR_FEW_OPTION	An option that should be specified at the same time is not set. [DETAIL 0] Option name 1 [DETAIL 1] Option name 2 This error occurs when an operation that is equivalent to the two indicated option names is not performed.

Detailed error code.	Description
PJCMD_SUBERR_CONFLICT	An option that cannot be specified at the same time is specified.
PJCMD_SUBERR_INVAL_RUNITNAME	The specified resource unit name is invalid. [DETAIL 0] Resource unit name
PJCMD_SUBERR_INVAL_ID	The specified node group ID, boot group ID, or node ID is invalid. [DETAIL 0] Specified ID
PJCMD_SUBERR_INVAL_RANGE	The specified parameter range is invalid. [DETAIL 0] Specified parameter range
PJCMD_SUBERR_INVAL_RGRPNAME	The specified resource group name is invalid. [DETAIL 0] Resource group name
PJCMD_SUBERR_SAME_ID	The specified node group ID, boot group ID, or node ID already exists. [DETAIL 0] Specified ID
PJCMD_SUBERR_SAME_RUNAME	The specified resource unit name already exists. [DETAIL 0] Resource unit name
PJCMD_SUBERR_SAME_RGNAME	The specified resource group name already exists. [DETAIL 0] Resource group name
PJCMD_SUBERR_NOT_FOUND_ID	A specified node group ID, boot group ID, or node ID does not exist. [DETAIL 0] Specified ID
PJCMD_SUBERR_NOT_FOUND_RUNAME	The specified resource unit does not exist. [DETAIL 0] Resource unit name
PJCMD_SUBERR_PERM	There is no authority to get the specified node state. [DETAIL 0] Node ID
PJCMD_SUBERR_PERM_RUNIT	There is no authority to get information on the specified resource unit. [DETAIL 0] Resource unit name
PJCMD_SUBERR_NOINFO_NODE	The specified node information does not exist. [DETAIL 0] Node ID
PJCMD_SUBERR_NOINFO_RUNIT	The specified resource unit information does not exist. [DETAIL 0] Resource unit name
PJCMD_SUBERR_NOINFO_RGRP	The specified resource group information does not exist. [DETAIL 0] Resource group name
PJCMD_SUBERR_CONNECT	Communication with the compute cluster management node has failed. [DETAIL 0] Cluster name
PJCMD_SUBERR_STANDBYNODE	This request cannot be made on a standby node.
PJCMD_SUBERR_WARN_CONNECT	Communication with a specific compute cluster management node has failed. However, communication with the other cluster management nodes was successful. [DETAIL 0] Cluster name
PJCMD_SUBERR_MEMORY	Acquisition of memory has failed. [CODE 1] errno at failure [CODE 2] Acquisition size
PJCMD_SUBERR_PSM_ERROR	Internal error [CODE 1] Investigation data [DETAIL 0] Investigation data
PJCMD_SUBERR_TRN_ERROR	Internal error [CODE 1] Investigation data [DETAIL 0] Investigation data
PJCMD_SUBERR_API_ERROR	Internal error

Detailed error code.	Description
PJCMD_SUBERR_SYSCALL_ERROR	Internal error [CODE 1] Investigation data [DETAIL 0] Investigation data
PJCMD_SUBERR_JACCTDB_ERROR	Internal error [CODE 1] Investigation data [CODE 2] Investigation data [DETAIL 0] Investigation data

A.7.3 pjcmd_errcodejcmd_errcode Variable

```
extern __thread pjcmd_error_t pjcmd_errcode
```

This is a global variable in which an error code of the command API is stored. The error of a command API function that is called at the end is set. The following table lists the meanings of the error codes.

Error Code	Main Meaning
PJCMD_SUCCESS	Success
PJCMD_ERROR_INVALID_HANDLE	The specified handle is invalid. A pointer to the handle is NULL. A handle whose operation type is different.
PJCMD_ERROR_INVALID_RESP	The specified response information is invalid. The pointer to response information is NULL. The response information differs in operation type.
PJCMD_ERROR_INVALID_ARGUMENT	An argument of the function is invalid.
PJCMD_ERROR_UNKNOWN_OPTION	An unknown option was specified. Mainly, this is an error in an argument analysis function.
PJCMD_ERROR_INVALID_OPTION	A method to specify an option is invalid. Mainly, this is an error in an argument analysis function.
PJCMD_ERROR_UNKNOWN_PARAM	An unknown parameter was specified.
PJCMD_ERROR_INVALID_PARAM	A parameter value is invalid. A specification method is incorrect. A required parameter is not set.
PJCMD_ERROR_NODATA	There is no applicable data. This error occurs when a specified index is out of range and there is no more data to be obtained.
PJCMD_ERROR_OPEN	The file has failed to open.
PJCMD_ERROR_INVALID_NODE	A node that is used to call a function is inappropriate. This error occurs when a node that is used to call the operation request function, pjcmd_*operation*_execute(), is inappropriate.
PJCMD_ERROR_NOPERM	There is no authority to call the function. The operator is not the administrator. The job ALC function limits calling the function.
PJCMD_ERROR_CONNECT	Communication with the job operation management function failed.
PJCMD_ERROR_NOMEM	The memory does not have enough space.

Error Code	Main Meaning
PJCMD_ERROR_BUSY	It is in busy state. For example, busy state occurs when another operation request function is called while the operation request function, pjcmd_*operation*_execute(), is being processed.
PJCMD_ERROR_TOO_LONG	The data size is too long.
PJCMD_ERROR_NOENT	The file or directory does not exist.
PJCMD_ERROR_ACCESS	There is no authority to access files or directories.
PJCMD_ERROR_SIGNAL	The process is interrupted by an interrupt, such as signal.
PJCMD_ERROR_INTERNAL	An internal error has occurred.

A.7.4 Variable pjcmd_optarg

```
extern __thread char *pjcmd_optarg
```

Pointer to an argument (*argv* element) that is being analyzed with the pjcmd_getopt_long() function
* It is equivalent to optarg of the getopt_long(3) function.

A.7.5 Variable pjcmd_optind

```
extern __thread int pjcmd_optind
```

An argument index that is processed next by the pjcmd_getopt() function
* It is equivalent to optind of the getopt_long(3) function.

A.7.6 Variable pjcmd_optopt

```
extern __thread int pjcmd_optopt
```

When an option that cannot be recognized by the pjcmd_getopt_long() function is detected, the option is stored.
* It is equivalent to optopt of the getopt_long(3) function.

A.7.7 PJCMD_UNLIMITED Constant

```
#define PJCMD_UNLIMITED (~0UL)
```

This function specifies a limit value. This value indicates that it is limitless (not limited).

A.7.8 PJCMD_UNDEFINED Constant

```
#define PJCMD_UNDEFINED (~1UL)
```

This function specifies a limit value or a function to get the setting information of the job ACL function. This value indicates that it is an invalid value (not specified).

A.7.9 PJCMD_MAX_SUBJOBID_STR_LEN Constant

```
#define PJCMD_MAX_SUBJOBID_STR_LEN 32
```

This is a necessary area size to store a sub job ID structure after converting it to a character string with the pjcmd_subjobid_to_str() function. It includes the NULL character at the end.

Appendix B Job Operation API Reference

B.1 Job Submission

This section describes the functions for submitting jobs.

Figure B.1 Requesting Job Submission

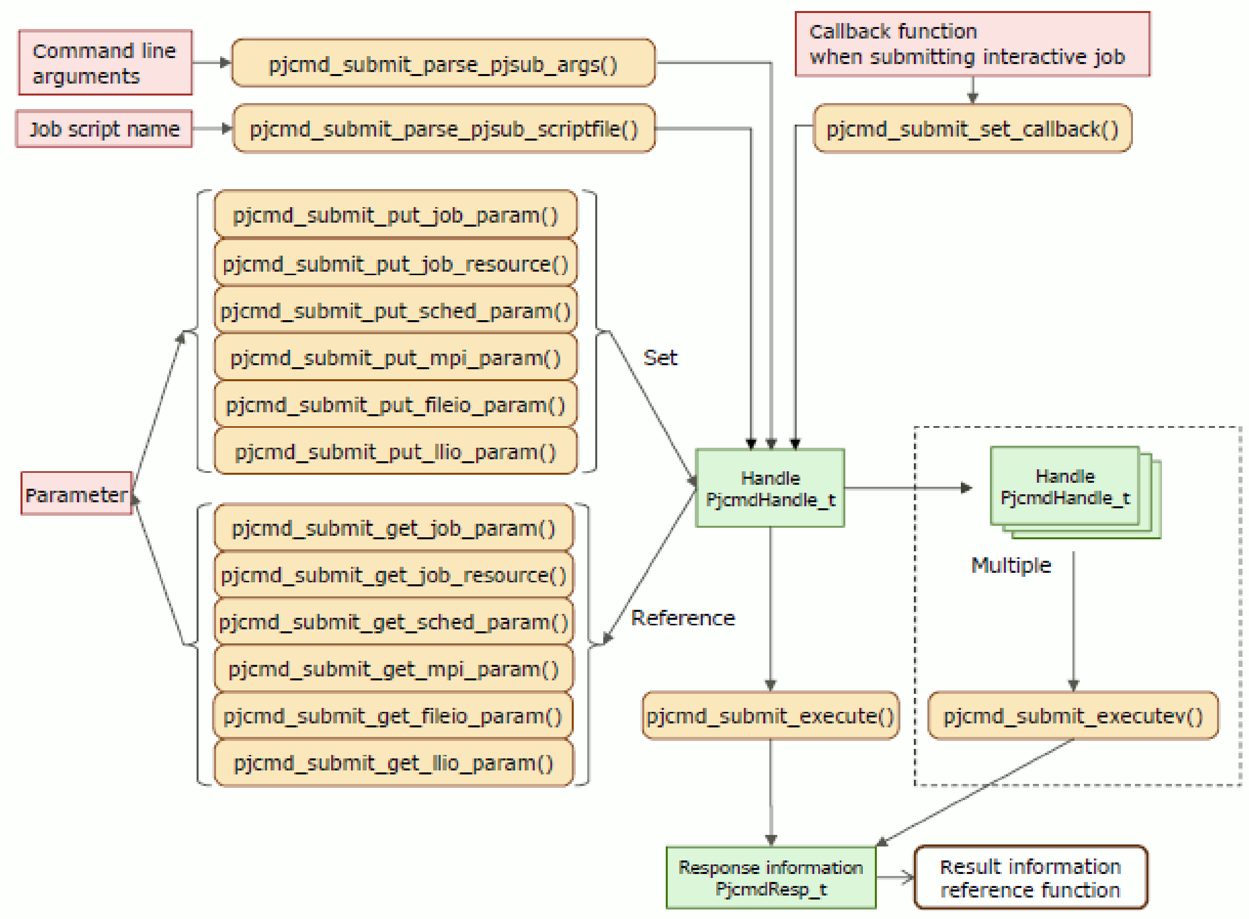
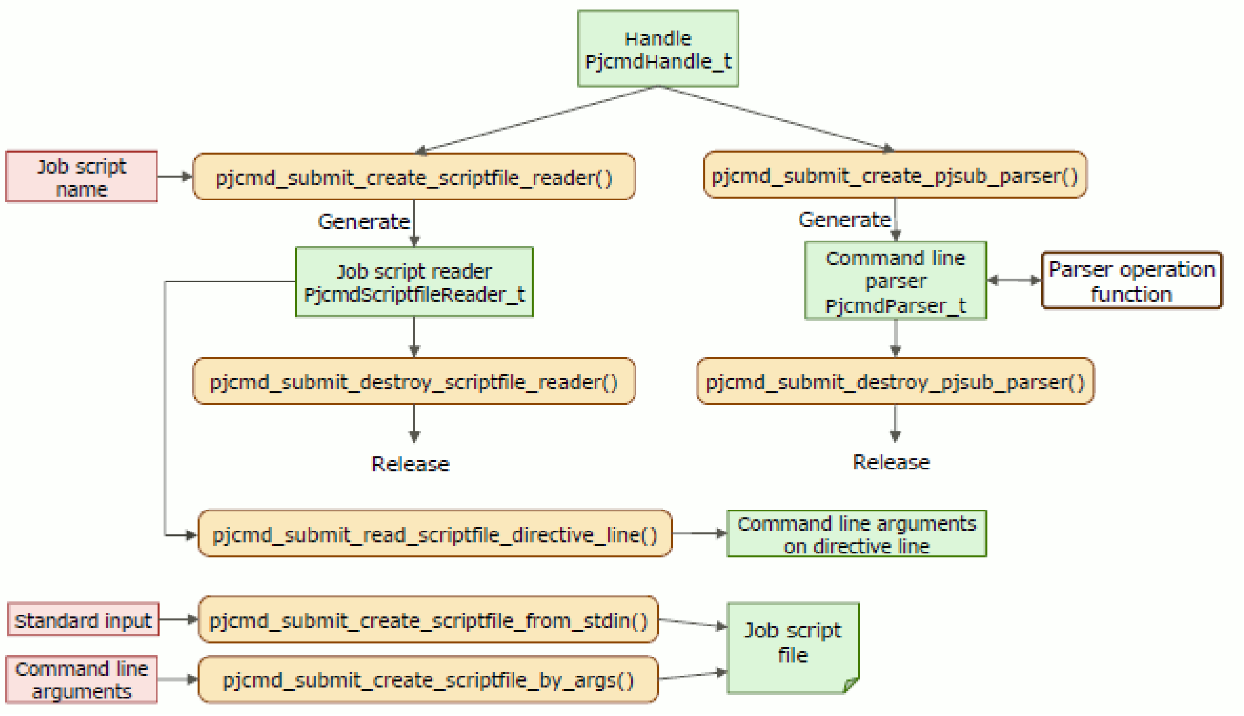

Command line arguments
pjcmd_submit_parse_pjsub_args()
Job script name
pjcmd_submit_parse_pjsub_scriptfile()
Callback function when submitting interactive job
pjcmd_submit_set_callback()
pjcmd_submit_put_job_param()
pjcmd_submit_put_job_resource()
pjcmd_submit_put_sched_param()
pjcmd_submit_put_mpi_param()
pjcmd_submit_put_fileio_param()
pjcmd_submit_put_llio_param()
Set
Parameter
Handle PjcmdHandle_t
Handle PjcmdHandle_t
Multiple
pjcmd_submit_get_job_param()
pjcmd_submit_get_job_resource()
pjcmd_submit_get_sched_param()
pjcmd_submit_get_mpi_param()
pjcmd_submit_get_fileio_param()
pjcmd_submit_get_llio_param()
Reference
pjcmd_submit_execute()
pjcmd_submit_executev()
Response information PjcmdResp_t
Result information reference function

Figure B.2 Operation of the Parser and Reader Related to Job Submission


Handle
PjcmdHandle_t
Job script name
pjcmd_submit_create_scriptfile_reader()
pjcmd_submit_create_pjsub_parser()
Generate
Generate
Job script reader
PjcmdScriptfileReader_t
Command line parser
PjcmdParser_t
Parser operation function
pjcmd_submit_destroy_scriptfile_reader()
pjcmd_submit_destroy_pjsub_parser()
Release
Release
pjcmd_submit_read_scriptfile_directive_line()
Command line arguments on directive line
Standard input
pjcmd_submit_create_scriptfile_from_stdin()
Command line arguments
pjcmd_submit_create_scriptfile_by_args()
Job script file

B.1.1 pjcmd_submit_parse_pjsub_args()

```
pjcmd_result_t pjcmd_submit_parse_pjsub_args(PjcmdHandle_t *handle_p, int argc, char **argv_pp)
```

This function analyzes command line arguments based on the specification of a pjsub command option and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option is not specified.
- An exclusive option is specified.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

Calling this function moves an option and an argument that is not the parameter of the option, namely the job script name, to the end of the *argv_pp*[] array.
When the operation is successfully completed, the pjcmd_optind variable specifies the job script (the first argument other than the option). The caller needs to set the job script in a handle separately. If the job is a step job and if two or more job scripts are specified, the caller needs to replicate a handle or create a new handle for each job script.
If an unknown option and parameter have been detected, or if an incorrect method to specify an option and parameter has been detected, the analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.

B.1.2 pjcmd_submit_parse_pjsub_scriptfile()

```
pjcmd_result_t pjcmd_submit_parse_pjsub_scriptfile(PjcmdHandle_t *handle_p, const char *filename_p,
const char *directive_prefix_p, int32_t *lineno_p, char **detail_pp)
```

This function analyzes the instruction lines in a job script file based on the specification of a pjsub command option while reading the job script file, and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

filename_p

Path to a job script

directive_prefix_p

Character string recognized as an instruction line

lineno_p

The line number where an error was detected is stored in **lineno_p.*

detail_pp

**detail_pp* indicates an option or an option argument where an error was detected. The area indicated by **detail_pp* is a reserved area in a handle. The caller must not directly release it. The area is retained until the handle is released or this function is called again.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.
The line number where an error was detected is stored in *lineno_p*. **detail_pp* indicates the option or the argument where the error was detected.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_INVALID_ARGUMENT

There is an invalid argument (NULL) other than a handle.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected in an instruction line.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option that appears in an instruction line is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option are not specified.

B.1.3 pjcmd_submit_create_pjsub_parser()

```
PjcmdParser_t *pjcmd_submit_create_pjsub_parser(PjcmdHandle_t *handle_p)
```

This function generates a command line parser for job submission operations.
The command line parser retains the same option specification information as the pjsub command option. The parser uses the specifications when uniquely analyzing options with the pjcmd_getopt_long() function. The original option specifications can be customized by changing the information in the command line parser.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

The command line parser is returned. The caller needs to release the returned command line parser by using the pjcmd_submit_destroy_pjsub_parser() function.
If an error occurs, NULL is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.1.4 pjcmd_submit_destroy_pjsub_parser()

```
pjcmd_result_t pjcmd_submit_destroy_pjsub_parser(PjcmdParser_t *parser_p)
```

This function releases a command line parser for job submission operations.

[ARGUMENTS]

parser_p

Pointer to the command line parser to be released

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

parser_p is invalid (NULL).

B.1.5 pjcmd_submit_create_scriptfile_reader()

```
PjcmdScriptfileReader_t *pjcmd_submit_create_scriptfile_reader(const PjcmdHandle_t *handle_p, const
char *filename_p, const char *directive_prefix_p)
```

This function generates data (reader) to read a job script file.

[ARGUMENTS]

handle_p

Pointer to a handle

filename_p

Path of the job script

directive_prefix_p

Character string recognized as an instruction line

[RETURN VALUE]

The reader of the job script file is returned. The caller needs to release the generated reader by using the pjcmd_submit_destroy_scriptfile_reader() function.
If an error occurs, NULL is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_INVALID_ARGUMENT

There is an invalid argument (NULL) other than a handle.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.1.6 pjcmd_submit_destroy_scriptfile_reader()

```
pjcmd_result_t pjcmd_submit_destroy_scriptfile_reader(PjcmdScriptfileReader_t *reader_p)
```

This function releases the reader of a job script file.

[ARGUMENTS]

reader_p

Pointer to the reader of a job script file

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

reader_p is invalid (NULL).

B.1.7 pjcmd_submit_read_scriptfile_directive_line()

```
pjcmd_result_t pjcmd_submit_read_scriptfile_directive_line(PjcmdScriptfileReader_t *reader_p, int
*argc_p, char ***argv_ppp)
```

This function uses the reader of a job script file to read one instruction line in the job script file and returns a command line argument that can be obtained.
This function expands a variable (variable specified in the pjcmd_submit_put_param() function) that is set in the handle corresponding to a reader and analyzes a special character, such as a double quotation mark. If a read line is other than a comment line (a line that does not begin with "#"), the subsequent instruction lines are regarded as comment lines. If the length of a line containing the linefeed character is greater than 4,096 words, or if the number of command line arguments written on an instruction line is more than 64, an error occurs.
This function only reads the instruction lines in a job script file and does not set the contents of the instruction lines in a handle. The caller uniquely analyzes arguments and uses them when values need to be set in a handle.

[ARGUMENTS]

reader_p

Pointer to the reader of a job script file

argc_p

The number of read arguments is stored in **argc_p*.

argv_ppp

The read arguments are stored as an array (**argv_ppp*)[].
The first element in the array *(**argv_ppp*)[0] indicates a character string (example: "#PJM") indicating an instruction line. The contents of an array that is indicated by **argv_ppp* are undetermined after reading the next instruction line.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

reader_p, *argc_p*, or *argv_ppp* is invalid (NULL).

PJCMD_ERROR_TOO_LONG

The length of a read line is over 4,096 words. Alternatively, the number of arguments in a line is more than 64.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_NODATA

An instruction line to be read does not exist. Alternatively, all command lines have been read.

PJCMD_ERROR_INVALID_PARAM

A script file description is invalid.

PJCMD_ERROR_OPEN

Script file reading failed.

PJCMD_ERROR_INTERNAL

Internal error

B.1.8 pjcmd_submit_put_param()

```
pjcmd_result_t pjcmd_submit_put_param(PjcmdHandle_t *handle_p, pjcmd_submit_param_t param, const
void *val_p)
```

This function sets parameters for job submission operations in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be set (See the table below.)

val_p

Pointer to the storage area for a parameter value to be set. For example, if the value type to be set is char * type, the caller must prepare a storage area for the char * type value and specify a pointer (char **) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_SUBMIT_SCRIPTFILE	Path to a job script (one file) If this parameter is not set, the job details are read from the standard input.	char *
PJCMD_SUBMIT_JOBNAME	Job name (equivalent to pjsub -N) If this parameter is not set, a job name is used as a job script name. The job name is "STDIN" when a job script is the standard input.	char *
PJCMD_SUBMIT_COMMENT	Comment character string (equivalent to pjsub --comment)	char *
PJCMD_SUBMIT_JOBMODEL	Job model PJCMD_JOBMODEL_NORMAL: Normal job (Default) PJCMD_JOBMODEL_BULK: Bulk job PJCMD_JOBMODEL_STEP: Step job PJCMD_JOBMODEL_MSWK: Master-worker job	int
PJCMD_SUBMIT_JOBTYPE	Job type PJCMD_JOBTYPE_BATCH: Batch job (Default)	int

param	**val_p*	Type of **val_p*
	PJCMD_JOBTYPE_INTERACTIVE: Interactive job	
PJCMD_SUBMIT_GNAME	Group name when executing a job (equivalent to pjsub --gname) The PJCMD_SUBMIT_GNAME parameter or PJCMD_SUBMIT_GID parameter, whichever is specified last, is valid. If this parameter is not set, the current group name is applied.	char *
PJCMD_SUBMIT_GID	Group ID when executing a job (equivalent to pjsub --gid) The PJCMD_SUBMIT_GNAME parameter or PJCMD_SUBMIT_GID parameter, whichever is specified last, is valid. If this parameter is not set, the current group ID is applied.	gid_t
PJCMD_SUBMIT_MAILOPT	E-mail send time of a report regarding information such as job status and notification contents (equivalent to pjsub -m) b: When starting job execution (beginning) e: When job completed (end) r: When re-executing job (restart) s: Job statistical information is reported (without information for each node) Job statistical information is reported (with information for each node) If this parameter is not set, no report is sent by e-mail.	char *
PJCMD_SUBMIT_MAILLIST	Destination e-mail address to report a job by e-mail (equivalent to pjsub --mail-list) Multiple e-mail addresses can be specified by separating them with a comma (,). If this parameter is not set, an e-mail is sent to the user who submitted the job.	char *
PJCMD_SUBMIT_WAITMODE	Wait mode when submitting a job (equivalent to pjsub -w) - PJCMD_SUBMIT_WAITMODE_WAIT Wait until job submission is completed (equivalent to no specification of the pjsub -w option). (Default) - PJCMD_SUBMIT_WAITMODE_JOBCHK Wait until job acceptance and job check are completed, but do not wait until job submission is completed (equivalent to pjsub -w jobchk). - PJCMD_SUBMIT_WAITMODE_NOWAIT Wait until job acceptance is completed, but do not wait for job check and job submission (equivalent to pjsub -w nowait).	int
PJCMD_SUBMIT_ENV_INHERIT	Specification of whether or not to send all environment variables to compute nodes (equivalent to pjsub -X) 0: Do not transfer (Default) 1: Transfer	int

param	**val_p*	Type of **val_p*
PJCMD_SUBMIT_BULK_STARTNO	Bulk start number of a bulk job (equivalent to pjsub --bulk --sparam *m-n*) This setting is required when submitting a bulk job. The specifiable value ranges from 0 to 999999.	uint32_t
PJCMD_SUBMIT_BULK_ENDNO	Bulk end number of a bulk job (equivalent to (pjsub --bulk --sparam *m-n*) This setting is required when submitting a bulk job. The specifiable value ranges from 0 to 999999.	uint32_t
PJCMD_SUBMIT_STEP_DEPEND	Step job relational expression (equivalent to pjsub --step --sparam sd=*form*) The parameter has the same format as the pjsub command (see pjsub(1).)	char *
PJCMD_SUBMIT_STEP_NO	Step number of a step job (equivalent to pjsub --step --sparam sn=*n*) The specifiable value ranges from 0 to 65534.	uint16_t
PJCMD_SUBMIT_STEP_JOBNAME	Job name of an existing step job (equivalent to pjsub --step --sparam jnam=*name*) The PJCMD_SUBMIT_STEP_JID parameter setting is exclusive. If PJCMD_SUBMIT_STEP_JID is set first, an error occurs.	char *
PJCMD_SUBMIT_STEP_JID	Job ID of an existing step job (equivalent to pjsub --step --sparam jid=*jobid*) The PJCMD_SUBMIT_STEP_JOBNAME parameter setting is exclusive. If PJCMD_SUBMIT_STEP_JOBNAME is set first, an error occurs. The specifiable value ranges from 0 to 2147483647.	uint32_t
PJCMD_SUBMIT_INTERACT_WAITTIME	Time to wait (in seconds) until the resource for an interactive job is allocated (equivalent to pjsub --interact --sparam wait-time=*time*) Values ranging from 0 to 36000 and PJCMD_UNLIMITED can be specified. If this parameter is not set, 0 is applied. If a value in a batch job is set, an error occurs in the pjcmd_submit_execute() function.	uint64_t
PJCMD_SUBMIT_FSNAME	Any character string, such as a file system name (equivalent to pjsub --fs) If a character string is not set and the environment variable PJM_FSNAME is set, its value is used.	char *
PJCMD_SUBMIT_APPNAME	Any character string, such as an application name (equivalent to pjsub --appname) If a character string is not set and the environment variable PJM_APPNAME is set, its value is used.	char *
PJCMD_SUBMIT_ENV	Environment variable that is set when executing a job (equivalent to pjsub -x) Each element must be a character string of "variable name = value" and the last element must be NULL.	char **

param	**val_p*	Type of **val_p*
PJCMD_SUBMIT_VAR	Variable that is used when analyzing the instruction lines in a script file (equivalent to pjsub --vset) Each element must be a character string of "variable name = value" and the last element must be NULL.	char **
PJCMD_SUBMIT_PREFIX	Directive prefix in a job script (equivalent to pjsub -C) It is a character string such as "#PJM" indicating an instruction line. If no character string is set, "#PJM" is applied.	char *
PJCMD_SUBMIT_SCRIPT_DELIMITER	Character that separates multiple job scripts of a step job (equivalent to pjsub --script-delimiter) If this parameter is not set, a comma (,) is used to separate multiple job scripts. This parameter does not affect job submission operations.	char *
PJCMD_SUBMIT_VERBOSE	Equivalent to the specification of the --verbose option in the pjsub command 0: Not specified (Default) 1: Specified This parameter does not affect job submission operations.	int
PJCMD_SUBMIT_Z	Equivalent to the specification of the -z option in the pjsub command A value to be specified is equivalent to the following for the pjsub command: "Jid": Specification of -z jid option Other character string: Specification of only -z option This parameter does not affect job submission operations.	char *
PJCMD_SUBMIT_NET_ROUTE [FX]	Specification of whether or not to continue executing a job when a Tofu interconnect link goes down for an FX server (equivalent to the pjsub --net-route option) - PJCMD_SUBMIT_NET_ROUTE_DYNAMIC Change the Tofu interconnect communication path (equivalent to pjsub --net-route dynamic). Job execution continues. - PJCMD_SUBMIT_NET_ROUTE_STATIC Do not change the Tofu interconnect communication path (equivalent to pjsub --net-route static). The job ends abnormally. If this parameter is not set, the setting is based on the job ACL function settings.	int
PJCMD_SUBMIT_HELP	Equivalent to the specification of the --help option in the pjsub command 0: Not specified (Default) 1: Specified This parameter does not affect job submission operations.	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.1.9 pjcmd_submit_get_param()

```
pjcmd_result_t pjcmd_submit_get_param(const PjcmdHandle_t *handle_p, pjcmd_submit_param_t param,
void *val_p)
```

This function references the parameter that is set in a handle to submit the job.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_submit_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.

- This is not a handle for job submission.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param*.

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

B.1.10 pjcmd_submit_put_job_resource()

```
pjcmd_result_t pjcmd_submit_put_job_resource(PjcmdHandle_t *handle_p, const char *rscname_p, const
void *val_p)
```

This function sets a resource that is allocated to a job.

[ARGUMENTS]

handle_p

Pointer to a handle

rscname_p

Resource name to be set (See the table below.)

val_p

Pointer to the storage area for a resource amount to be set. For example, if resource is "node," the pointer (char *)shape_p that indicates the storage area for the character string "*X*x*Y*x*Z*" indicating a node shape must be prepared, and the pointer (char **)&shape_p to this pointer is specified in *val_p*. If NULL is specified, the parameter value is initialized (not set).

rscname_p	**val_p*	Type of **val_p*
"node"	Number of nodes or node shape (equivalent to pjsub -L node) - One-dimensional shape "*N*[:no-io-exclusive|:io-exclusive][:torus|:mesh|:noncont]" - Two-dimensional shape "*X*x*Y*[:no-io-exclusive|:io-exclusive][:torus|:mesh|:noncont]" - Three-dimensional shape "*X*x*Y*x*Z*[:no-io-exclusive|:io-exclusive][:strict|:strict-io][:torus|:mesh|:noncont]" * The item has the same format as the pjsub command.	char *
"vnode"	Number of virtual nodes (equivalent to pjsub -L vnode) The specifiable value ranges from 1 to 2147483647.	uint64_t
"node-mem"	Upper limit on the memory amount per node when specifying a node (equivalent to pjsub -L node-mem) The specifiable value ranges from 1048576 (1 Mi) to 2251799812636672 (2147483647 Mi) bytes. If the upper limit is not set, PJCMD_UNLIMITED must be specified.	size_t
"elapse"	Elapsed time limit value (equivalent to pjsub -L elapse=*limit*) The specifiable value ranges from 1 to 2147483647 seconds. If the limit value is not specified, PJCMD_UNLIMITED must be specified.	time_t
"adaptive-elapse"	Minimum and maximum values of the elapsed job time (equivalent to pjsub -L elapse=*min*-*max*)	time_t *

rscname_p	**val_p*	Type of **val_p*
	The values are specified as an array, *elapse*[], with two time_t type elements. (Time_t)*elapse*[0]: Minimum value for elapsed time (time_t)*elapse*[1]: Maximum value for elapsed time The specifiable value ranges from 1 to 2147483647 seconds. However, *elapse*[0] must be smaller than *elapse*[1]. If the job executable time is not limited when the elapsed job time exceeds *elapse*[0] (equivalent to pjsub -L elapse=*min*-unlimited), PJCMD_UNLIMITED must be specified in elapse[1]. If PJCMD_UNDEFINED is specified in *elapse*[1] (equivalent to pjsub -L elapse=*min*-), the maximum value that is set by the job ACL function is applied. If PJCMD_UNLIMITED or PJCMD_UNDEFINED is specified in *elapse*[0], an error occurs. Either the resource name "elapse" or "adaptive-elapse", whichever is specified last, is valid.	
"vnode-core"	Number of CPU cores per virtual node when specifying "vnode" (equivalent to pjsub -L vnode-core) The specifiable value ranges from 1 to 2147483647.	unit64_t
"core-mem"	Upper limit on the amount of memory usage per CPU core when specifying "vnode" (equivalent to pjsub -L core-mem) The specifiable value ranges from 1048576 (1 Mi) to 2251799812636672 (2147483647 Mi) bytes. If the upper limit is not set, PJCMD_UNLIMITED must be specified.	size_t
"vnode-mem"	Upper limit on the amount of memory usage per virtual node when specifying "vnode" (equivalent to pjsub -L vnode-mem) The specifiable value ranges from 1048576 (1 Mi) to 2251799812636672 (2147483647 Mi) bytes. If the upper limit is not set, PJCMD_UNLIMITED must be specified.	size_t
"rscunit"	Name of the resource unit to which a job is submitted (equivalent to pjsub -L rscunit, -L ru)	char *
"rscgrp"	Name of the resource group to which a job is submitted (equivalent to pjsub -L rscgrp, -L rg)	char *
"proc-core"	Core file size limit in a process unit (equivalent to pjsub -L proc-core) The specifiable value ranges from 0 to 2147483647 bytes (2 GiB-1). If the size is not limited, PJCMD_UNLIMITED must be specified.	size_t
"proc-cpu"	CPU time limit in a process unit (equivalent to pjsub -L proc-cpu) The specifiable value ranges from 1 to 2147483647. If the upper limit is not set, PJCMD_UNLIMITED must be specified.	uint64_t
"proc-crproc"	Limit on the number of generating processes in a process unit (equivalent to pjsub -L proc-crproc) The specifiable value ranges from 0 to 2147483647. If the number is not limited, PJCMD_UNLIMITED must be specified.	uint64_t
"proc-data"	Limit on the data segment size in a process unit (equivalent to pjsub -L proc-data) The specifiable value ranges from 0 to 2251799812636672 (2147483647 Mi) bytes. If the size is not limited, PJCMD_UNLIMITED must be specified.	size_t
"proc-lockm"	Limit on the lock memory size in a process unit (equivalent to pjsub -L proc-lockm)	size_t

rscname_p	**val_p*	Type of **val_p*
	The specifiable value ranges from 0 to 2251799812636672 (2147483647 Mi) bytes. If the size is not limited, PJCMD_UNLIMITED must be specified.	
"proc-msgq"	Limit on the message queue size in a process unit (equivalent to pjsub -L proc-msgq) The specifiable value ranges from 0 to 2251799812636672 (2147483647 Mi) bytes. If the size is not limited, PJCMD_UNLIMITED must be specified.	size_t
"proc-openfd"	Limit on the number of file descriptors in a process unit (equivalent to pjsub -L proc-openfd) The specifiable value ranges from 0 to 1048576.	uint64_t
"proc-psig"	Limit on the number of pending signals in a process unit (equivalent to pjsub -L proc-psig) The specifiable value ranges from 0 to 2147483647. If the number is not limited, PJCMD_UNLIMITED must be specified.	uint64_t
"proc-filesz"	Limit on the file size in a process unit (equivalent to pjsub -L proc-filesz) The specifiable value ranges from 2048 to 2147483647000000 (2147483647 M) bytes. If the size is not limited, PJCMD_UNLIMITED must be specified.	size_t
"proc-stack"	Limit on the stack size in a process unit (equivalent to pjsub -L proc-stack) The specifiable value ranges from 0 to 2251799812636672 (2147483647 Mi) bytes. If the size is not limited, PJCMD_UNLIMITED must be specified.	size_t
"proc-vmem"	Limit on the virtual memory size in a process unit (equivalent to pjsub -L proc-vmem) The specifiable value ranges from 0 to 2251799812636672 (2147483647 Mi) bytes. If the size is not limited, PJCMD_UNLIMITED must be specified.	size_t
String other than the above	Amount of a custom resource whose name is indicated by the rscname_p argument (equivalent to pjsub -L *CustomResourceName*)	char *

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_INVALID_ARGUMENT

rscname_p is invalid (NULL).

PJCMD_ERROR_INVALID_PARAM

The value of resource amount is invalid.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.1.11 pjcmd_submit_get_job_resource()

```
pjcmd_result_t pjcmd_submit_get_job_resource(const PjcmdHandle_t *handle_p, const char *rscname_p,
void *val_p)
```

This function references the resource amount that is set in a handle and to be allocated to a job.

[ARGUMENTS]

handle_p

Pointer to a handle

rscname_p

Name of the resource whose value is referenced. The resource names that can be specified are the same as those for the pjcmd_submit_put_job_resource() function.

val_p

A value is stored in **val_p* based on the *racname_p* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_INVALID_ARGUMENT

rscname_p or *val_p* is invalid (NULL).

PJCMD_ERROR_NODATA

No resource amount is set for the specified resource.

B.1.12 pjcmd_submit_put_mpi_param()

```
pjcmd_result_t pjcmd_submit_put_mpi_param(PjcmdHandle_t *handle_p, pjcmd_submit_mpi_param_t param,
const void *val_p)
```

This function sets parameters in a handle that are related to MPI job execution.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be set (See the table below.)

val_p

Pointer to the storage area for a parameter value to be set. For example, if the value type to be set is char * type, the caller must prepare a storage area for the char * type value and specify a pointer (char **) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_SUBMIT_MPI_SHAPE	Shape of a process that is generated when starting a program (equivalent to pjsub --mpi shape) - If compute node is FX server - One-dimensional shape: "*X*" - Two-dimensional shape: "*X*x*Y*" - Three-dimensional shape: "*X*x*Y*x*Z*" Only "1" can be specified when the resource name "vnode" is set in a handle. An error occurs in other cases. - If compute node is PRIMERGY server Ignored	char *
PJCMD_SUBMIT_MPI_PROC	Maximum number of processes generated when starting a program (equivalent to pjsub --mpi proc) The specifiable value ranges from 1 to 2147483647. If a value larger than the value that is calculated using the following calculation method is specified, job submission results in an error: number of nodes that is indicated by the shape specified with the resource name "shape" x number of CPU cores in 1 node.	int
PJCMD_SUBMIT_MPI_MAX_PROC_PER_NODE	Maximum number of processes generated in one node by a program (equivalent to pjsub --mpi max-proc-per-node) The specifiable value ranges from 1 to 2147483647. If the specified value is larger than the number of CPU cores in 1 node, or if the value obtained by converting the value specified by PJCMD_SUBMIT_MPI_PROC into the number of processes to be generated in 1 node is larger, job submission results in an error. If this parameter is specified when the resource name "vnode" is set in a handle, job submission results in an error.	int
PJCMD_SUBMIT_MPI_RANK_MAP_BYNODE	Rank allocation rule (equivalent to pjsub --mpi rank-map-bynode) - If compute node is FX server "XY," "YX," "XYZ," "XZY," "YXZ," "YZX," "ZXY," or "ZYX" can be specified in **val_p*. Their meanings are the same as pjsub --mpi rank-map-bynode=*rankmap*. When an empty string ("") is specified in **val_p*, it has the same meaning as pjsub --mpi rank-map-bynode.	char *

param	**val_p*	Type of **val_p*
	- If compute node is PRIMERGY server An empty string must be specified in **val_p*. In this case, it has the same meaning as pjsub --mpi rank-map-bynode.	
PJCMD_SUBMIT_MPI_RANK_MAP_BYCHIP	Rank allocation rule (equivalent to pjsub --mpi rank-map-bychip) - If compute node is FX server "XY," "YX," "XYZ," "XZY," "YXZ," "YZX," "ZXY," or "ZYX" can be specified in **val_p*. Their meanings are the same as pjsub --mpi rank-map-bychip:*rankmap*. When an empty string ("") is specified in **val_p*, it has the same meaning as pjsub --mpi rank-map-bychip. - If compute node is PRIMERGY server An integer number n must be specified in *val_p as a character string. The meaning is the same as pjsub --mpi rank-map-bychip=*n*.	char *
PJCMD_SUBMIT_MPI_RANK_MAP_HOSTFILE	Rank map host file name (equivalent to pjsub --mpi rank-map-hostfile)	char *
PJCMD_SUBMIT_MPI_ASSIGN_ONLINE_NODE	This parameter specifies whether or not to guarantee that a failure node is not included in a node to be assigned (equivalent to pjsub --mpi assign-online-node). 0: Not guaranteed (Default) 1: Guaranteed	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.1.13 pjcmd_submit_get_mpi_param()

```
pjcmd_result_t pjcmd_submit_get_mpi_param(const PjcmdHandle_t *handle_p, pjcmd_submit_mpi_param_t
param, void *val_p)
```

This function references the set parameter values in a handle that are related to MPI job execution.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_submit_put_mpi_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

B.1.14 pjcmd_submit_put_sched_param()

```
pjcmd_result_t pjcmd_submit_put_sched_param(PjcmdHandle_t *handle_p, pjcmd_submit_sched_param_t
param, const void *val_p)
```

This function sets parameters in the *handle_p* handle that are related to scheduling of a job to be submitted.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be set (See the table below.)

val_p

Pointer to the storage area for a parameter value to be set. For example, if the value type to be set is int type, the caller must prepare a storage area for the int type value and specify a pointer (int *) to the area in *val_p*. If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_SUBMIT_SCHED_PRIORITY	Job priority (job priority for the same user, equivalent to pjsub -p) The specified value can be an integer from 0 to 255. If this parameter is not set, the setting is based on the job ACL function settings.	int
PJCMD_SUBMIT_SCHED_AUTORESTART	This parameter specifies whether or not to enable automatic job re-execution. 0: Disable 1: Enable If this parameter is not set, the setting is based on the job operation management settings specified by the administrator (papjm.conf and pmpjm.conf files).	int
PJCMD_SUBMIT_SCHED_STARTDATE	Scheduled time to start executing a job	time_t
PJCMD_SUBMIT_SCHED_VN_POLICY [PG]	Virtual node placement policy - PJCMD_SUBMIT_VN_POLICY_ABSPACK abs-pack (equivalent to pjsub -P vn-policy=abs-pack) - PJCMD_SUBMIT_VN_POLICY_PACK pack (equivalent to pjsub -P vn-policy=pack) - PJCMD_SUBMIT_VN_POLICY_UNPACK unpack (equivalent to pjsub -P vn-policy=unpack) - PJCMD_SUBMIT_VN_POLICY_ABSUNPACK abs-unpack (equivalent to pjsub -P vn-policy=abs-unpack) If this parameter is not set, the setting is based on the job ACL function settings.	int
PJCMD_SUBMIT_SCHED_VN_POLICY_N [PG]	When the virtual node placement policy is unpack or abs-unpack, the number of virtual nodes placed on a physical node is specified. The specifiable value ranges from 1 to 2147483647.	int
PJCMD_SUBMIT_SCHED_EXEC_POLICY [PG]	Execution mode policy - PJCMD_SUBMIT_EXEC_POLICY_SIMPLEX simplex (equivalent to pjsub -P exec-policy=simplex)	int

param	**val_p*	Type of **val_p*
	- PJCMD_SUBMIT_EXEC_POLICY_SHARE share (equivalent to pjsub -P exec-policy=share) If this parameter is not set, the setting is based on the job ACL function settings.	

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.1.15 pjcmd_submit_get_sched_param()

```
pjcmd_result_t pjcmd_submit_get_sched_param( const PjcmdHandle_t *handle_p,
pjcmd_submit_sched_param_t param, void *val_p)
```

This function references the set parameter values in a handle that are related to scheduling.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_submit_put_sched_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

B.1.16 pjcmd_submit_put_fileio_param()

```
pjcmd_result_t pjcmd_submit_put_fileio_param(PjcmdHandle_t *handle_p, pjcmd_submit_fileio_param_t
param, const void *val_p)
```

This function sets parameters that are related to file input/output at job execution.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be set (See the table below.)

val_p

Pointer to the storage area for a parameter value to be set. For example, if the value type to be set is char * type, the caller must prepare a storage area for the char * type value and specify a pointer (char **) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_SUBMIT_FILEIO_OFILE	Output destination path to the standard output for jobs (equivalent to pjsub -o|--out) If this parameter is not set, the file *job-name.job-ID*.out is used as the path.	char *
PJCMD_SUBMIT_FILEIO_EFILE	Path to the standard error output for jobs (equivalent to pjsub -e|--err) If this parameter is not set, the file *job-name.job-ID*.err is used as the path.	char *
PJCMD_SUBMIT_FILEIO_SFILE	Path to output a job statistical information file (equivalent to pjsub --spath) If this parameter is not set, *job-name job-ID*.stats is used as the path.	char *
PJCMD_SUBMIT_FILEIO_SFILE_MODE	Method to output a job statistical information file PJCMD_SUBMIT_SFILE_MODE_DISABLE A job statistical information file is not output. (Default)	int

param	**val_p*	Type of **val_p*
	PJCMD_SUBMIT_SFILE_MODE_JOB Only job information is output to a job statistical information file (equivalent to pjsub -s|--stats). PJCMD_SUBMIT_SFILE_MODE_JOB_AND_NODE Both job information and node information are output to a job statistical information file (equivalent to pjsub -S|--STATS).	
PJCMD_SUBMIT_FILEIO_MERGE	This parameter specifies whether or not the standard error output for jobs is the same file as the standard output. 0: Do not output to same file (Default) 1: Output to same file	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.1.17 pjcmd_submit_get_fileio_param()

```
pjcmd_result_t pjcmd_submit_get_fileio_param(const PjcmdHandle_t *handle_p,
pjcmd_submit_fileio_param_t param, void *val_p)
```

This function references the set parameters in a handle that are related to file input/output at job execution.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_submit_put_fileio_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

B.1.18 pjcmd_submit_put_llio_param()

```
pjcmd_result_t pjcmd_submit_put_llio_param(PjcmdHandle_t *handle_p, pjcmd_submit_llio_param_t param,
const void *val_p)
```

This function sets parameters in a handle that are related to usage of the layered storage system at job execution.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be set (See the table below.)

val_p

Pointer to the storage area for a parameter value to be set. For example, if the value type to be set is size_t type, the caller must prepare a storage area for the size_t type value and specify a pointer (size_t *) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_SUBMIT_LLIO_SHAREDTMP_SIZE	Size of a shared temporary area on first-layer storage The specifiable value ranges from 0 to 2251799812636672 (2147483647 Mi) bytes. The value must be specified in units of 1048576 (1 Mi) bytes.	size_t
PJCMD_SUBMIT_LLIO_LOCALTMP_SIZE	Size of a node temporary area on first-layer storage	size_t

param	**val_p*	Type of **val_p*
	The size of a shared temporary area that can be used by one job is a value that is calculated by multiplying the specified value by the number of compute nodes allocated to the job. The specifiable value ranges from 0 to 2251799812636672 (2147483647 Mi) bytes. The value must be specified in units of 1048576 (1 Mi) bytes.	
PJCMD_SUBMIT_LLIO_AUTO_READAHEAD	Whether or not to automatically read ahead an area in the layered storage system after a job has tried to read the area multiple times consecutively 0: Do not read in advance 1: Read in advance	int
PJCMD_SUBMIT_LLIO_ASYNC_CLOSE	Whether or not to asynchronously close files on the layered storage system 0: Synchronous close 1: Asynchronous close	int
PJCMD_SUBMIT_LLIO_CN_CACHED_WRITE_SIZE	Threshold value indicating whether or not to cache when writing to first-layer storage If the size of data to be written is smaller than the specified value when writing the data to first-layer storage, the data is not immediately written to the storage but instead temporarily stored in a cache on a compute node. The specifiable value ranges from 0 to 2251799812636672 (2147483647 Mi) bytes. The value must be specified in units of 4194304 (4 Mi) bytes.	size_t
PJCMD_SUBMIT_LLIO_CN_CACHE_SIZE	Size of a compute node cache The specifiable value ranges from 4194304 (4 Mi) to 2251799812636672 (2147483647 Mi) bytes. If the value is not larger than the value calculated by multiplying the stripe size (the PJCMD_SUBMIT_LLIO_STRIPE_SIZE parameter value) by 10, an error occurs.	size_t
PJCMD_SUBMIT_LLIO_CN_READ_CACHE	Whether or not to cache data on a compute node when a job reads a file from the layered storage system 0: Do not cache 1: Cache	int
PJCMD_SUBMIT_LLIO_SIO_READ_CACHE	Whether or not to cache data read from second-layer storage on first-layer storage when reading data 0: Do not cache 1: Cache	int
PJCMD_SUBMIT_LLIO_STRIPE_COUNT	Number of stripes per file when files are distributed on first-layer storage	int

param	**val_p*	Type of **val_p*
	The specifiable value ranges from 1 to 2147483647.	
PJCMD_SUBMIT_LLIO_STRIPE_SIZE	Stripe size for distributing files on first-layer storage The specifiable value ranges from 65536 (64 Ki) to 4294901760 (4194240 Ki) bytes. The value must be specified in units of 65536 (64 Ki) bytes.	size_t
PJCMD_SUBMIT_LLIO_UNCOMPLETED_FILEINFO_PATH	Path to output an unwritten file list This is the path to output information on files that could not finish writing from a cache to second-layer storage when completing a job.	char *
PJCMD_SUBMIT_LLIO_PERF	Whether or not to output LLIO performance information 0: Do not output 1: Output	int
PJCMD_SUBMIT_LLIO_PERF_PATH	A path is specified when the output destination of LLIO performance information needs to be changed.	char *

For any of the above parameters that are not set, the setting is based on the job ACL function settings.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.1.19 pjcmd_submit_get_llio_param()

```
pjcmd_result_t pjcmd_submit_get_llio_param(const PjcmdHandle_t *handle_p, pjcmd_submit_llio_param_t
param, void *val_p)
```

This function references the set parameters in a handle that are related to the use of the layered storage system at job execution.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_submit_put_llio_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

B.1.20 pjcmd_submit_create_scriptfile_from_stdin()

```
char *pjcmd_submit_create_scriptfile_from_stdin(const char *basedir_p, const char *filename_p)
```

This function creates standard input contents as a job script. If a job script is not specified in a command line argument, job details are obtained from the standard input and used to specify a job script with the pjcmd_submit_put_param() function.

[ARGUMENTS]

basedir_p

Path to a directory to create a job script.
Access privilege for the user who calls the function is required. If NULL is specified, the current directory that is used to call the function is used.

filename_p

File name for the job script that is created in the *basedir_p* directory. If NULL is specified, a file name that does not duplicate that of an existing file is automatically determined.

[RETURN VALUE]

Path name of a job script where standard input contents are stored. The caller needs to release the area. If the function fails, NULL is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_OPEN

A job script could not be created (no privilege, existing file with the same name, or inappropriate path name), or input is interrupted.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

[Note]

- The user is responsible for deleting the created job script file after submitting a job.
- This function does not return until the standard input is closed.

B.1.21 pjcmd_submit_create_scriptfile_by_args()

```
char *pjcmd_submit_create_scriptfile_by_args(const char *basedir_p, const char *filename_p, int argc,
const char *argv_p[])
```

This function creates a job script whose contents are command line arguments.

[ARGUMENTS]

basedir_p

Directory for creating a job script.
Access privilege for the user who calls the function is required. If NULL is specified, the current directory that is used to call the function is used.

filename_p

File name for the job script that is created in the *basedir_p* directory. If NULL is specified, a file name that does not duplicate that of an existing file is automatically determined.

argc

Number of command line arguments

argv_p[]

Array of command line arguments

[RETURN VALUE]

Path to a job script where the contents of a command line argument are stored. If the function fails, NULL is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

argc is 0 or *argv_p* is NULL.

PJCMD_ERROR_OPEN

A job script could not be created (no privilege, existing file with the same name, or inappropriate path name).

PJCMD_ERROR_NOMEM

Memory acquisition failed.

[Note]

The user is responsible for deleting the created job script file.

B.1.22 pjcmd_submit_set_callback()

```
pjcmd_result_t pjcmd_submit_set_callback(
    PjcmdHandle_t *handle_p,
    void (*job_accept_callback_func_p)(const PjcmdSubjobid_t *),
    int (*start_wait_callback_func_p)(const PjcmdSubjobid_t *),
    void (*job_start_callback_func_p)(const PjcmdSubjobid_t *),
    void (*job_end_callback_func_p)(const PjcmdSubjobid_t *))
```

This function registers a callback function that is called at a specific time according to the progress of interactive job processing. This function is used when a user needs to call their own process at a specific time. For example, this function can be used to output a message that indicates the progress of interactive job processing.
If a NULL pointer is specified as a callback function, it is regarded that the callback function is not set.

[ARGUMENTS]

handle_p

Pointer to a handle

job_accept_callback_func_p

Pointer to the function that is called when an interactive job is accepted

start_wait_callback_func_p

Pointer to the function that is called when an interactive job is waiting to be executed. When this function is registered, it is called every three minutes until job execution begins.

job_start_callback_func_p

Pointer to the function that is called when interactive job execution begins

job_end_callback_func_p

Pointer to the function that is called when an interactive job is completed.

The job ID of the interactive job executed when the function is called is passed to the callback function as a sub job ID structure.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

B.1.23 pjcmd_submit_execute()

```
PjcmdResp_t *pjcmd_submit_execute(const PjcmdHandle_t *handle_p)
```

This function requests the job operation management function to submit a job based on a handle. This function can be called from the login node and compute cluster management node.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about a job submission request.
The caller must release the obtained response information by using the pjcmd_destroy_resp() function. If a job submission request fails, NULL is returned, and the cause is set in pjcmd_errcode.
The response information indicates whether the request succeeded or failed. Whether or not job submission has been accepted by the

job operation management function needs to be checked with a result code in the response information by using the pjcmd_get_jobresult_info() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node.
The function can be called from the login node and compute cluster management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_BUSY

An operation cannot be requested because another operation request function is being processed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

PJCMD_ERROR_NOENT

A file or directory that is set in the handle does not exist.

PJCMD_ERROR_ACCESS

There is no privilege to access a file that is set in the handle.

PJCMD_ERROR_OPEN

A file that is set in the handle failed to open.

PJCMD_ERROR_INTERNAL

Internal error

B.1.24 pjcmd_submit_executev()

```
PjcmdResp_t *pjcmd_submit_executev(const PjcmdHandle_t **handle_pp, int n)
```

This function is the same as the pjcmd_submit_execute() function except for specifying a handle as an array. However, if two or more handles are specified, a handle that is related to submission of the sub jobs of all step jobs must be specified. This function can be called from the login node and compute cluster management node.

[ARGUMENTS]

handle_pp

Array of pointers to a handle

n

Number of handles

[RETURN VALUE]

Response information about a job submission request.
The caller must release the obtained response information by using the pjcmd_destroy_resp() function. If a job submission request fails, NULL is returned, and the cause is set in pjcmd_errcode.
The response information indicates whether the request succeeded or failed. Whether or not job submission has been accepted by the job operation management function needs to be checked with a result code in the response information by using the pjcmd_get_jobresult_info() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job submission.
- If multiple handles are specified, the handles for jobs other than step jobs are included.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node.
The function can be called from the login node and compute cluster management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_BUSY

An operation cannot be requested because another operation request function is being processed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

PJCMD_ERROR_NOENT

A file or directory that is set in the handle does not exist.

PJCMD_ERROR_ACCESS

There is no privilege to access a file that is set in the handle.

PJCMD_ERROR_OPEN

A file that is set in the handle failed to open.

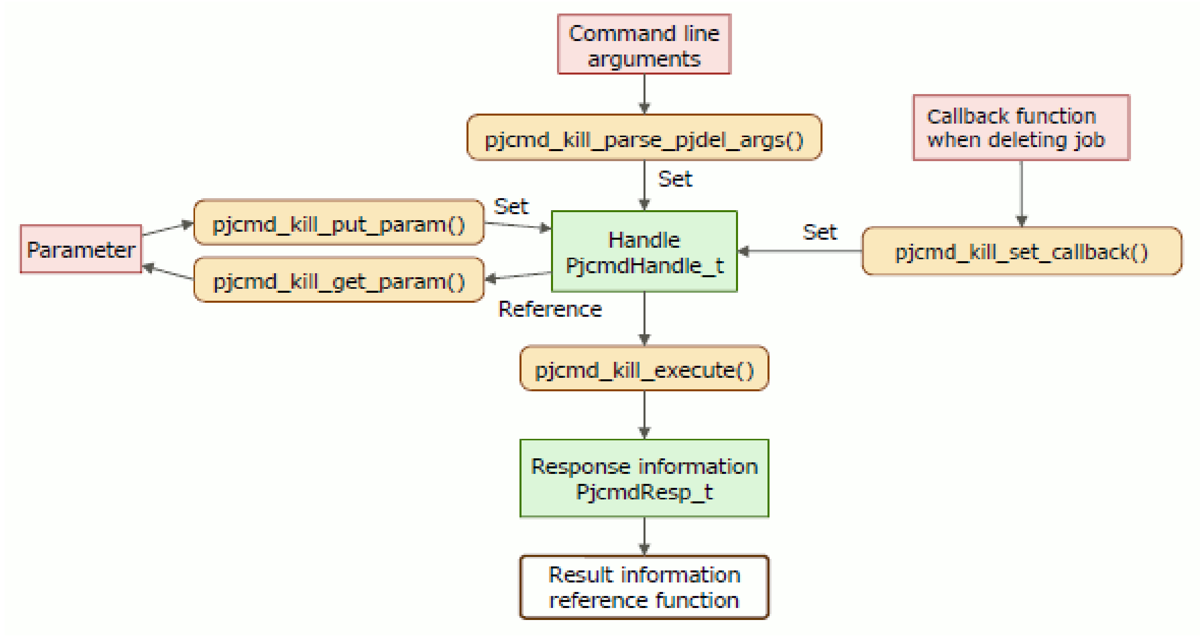
PJCMD_ERROR_INTERNAL

Internal error

B.2 Job Deletion

This section describes the functions for deleting (canceling) jobs.

Figure B.3 Requesting Job Deletion


Command line arguments
pjcmd_kill_parse_pjdel_args()
Set
Callback function when deleting job
Set
Parameter
pjcmd_kill_put_param()
Handle PjcmdHandle_t
Set
pjcmd_kill_set_callback()
pjcmd_kill_get_param()
Reference
pjcmd_kill_execute()
Response information PjcmdResp_t
Result information reference function

B.2.1 pjcmd_kill_parse_pjdel_args()

```
pjcmd_result_t pjcmd_kill_parse_pjdel_args(PjcmdHandle_t *handle_p, int argc, char **argv_pp)
```

This function analyzes command line arguments based on the specification of a pjdel command option and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job deletion.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option is not specified.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

Calling this function moves arguments other than options to the end of the *argv_pp*[] array.
When the operation is successfully completed, the pjcmd_optind variable indicates a job ID (the first argument other than options). The caller needs to set the job ID in a handle.
If an unrecognizable option is detected, analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.

B.2.2 pjcmd_kill_put_param()

```
pjcmd_result_t pjcmd_kill_put_param(PjcmdHandle_t *handle_p, pjcmd_kill_param_t param, const void
*val_p)
```

This function sets the parameters in a handle that are related to job deletion.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be set (See the table below.)

val_p

Pointer to the storage area for a parameter value to be set. For example, if the value type to be set is int type, the caller must prepare a storage area for the int type value and specify a pointer (int *) to the area in *val_p*. If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_KILL_ENFORCE	This parameter specifies whether or not to interrupt execution of a prologue or epilogue script if one is being executed (equivalent to pjdel --enforce). 0: Do not interrupt (Default) 1: Forcibly interrupt. Job will be deleted	int
PJCMD_KILL_REASON	Message that is output to job statistical information as a deletion reason (equivalent to pjdel --reason). This character string must be within 64 bytes including the NULL character at the end. The available characters are single-byte alphanumeric characters and symbols that can be displayed.	char *
PJCMD_KILL_NO_STATS	If the job being deleted is in the QUEUED state, suppress the output of the job statistical information file (.stats file) for that job (equivalent to pjdel --no-stats). 0: Do not suppress (Default) 1: Suppress	int

param	**val_p*	Type of **val_p*
PJCMD_KILL_NO_HISTORY	If the job being deleted is in the QUEUED state, suppress the output of that job to the job history information that is output by the -H option of the pjstat command (equivalent to pjdel --no-history). 0: Do not suppress (Default) 1: Suppress	int
PJCMD_KILL_LLIO_FLUSH	This parameter specifies whether or not to wait until flushing of unwritten files completes within a time range that does not exceed the limit value of the job execution time before deleting the job (equivalent to pjdel --llio-flush). 0: Do not wait until flushing completes (Default) 1: Wait until flushing completes	int
PJCMD_KILL_HELP	Equivalent to the --help option in the pjdel command 0: Not specified (Default) 1: Specified This parameter does not affect job deletion operations.	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job deletion.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.2.3 pjcmd_kill_get_param()

```
pjcmd_result_t pjcmd_kill_get_param(const PjcmdHandle_t *handle_p, pjcmd_kill_param_t param, void
*val_p)
```

This function references the parameters that are set in a handle for job deletion.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_kill_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job deletion.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

B.2.4 pjcmd_kill_execute()

```
PjcmdResp_t *pjcmd_kill_execute(const PjcmdHandle_t *handle_p)
```

This function requests the job operation management function to delete a job based on a handle. This function can be called from the login node and compute cluster management node.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about job deletion.
The caller must release the obtained response information by using the pjcmd_destroy_resp() function. If a job deletion request fails, NULL is returned, and the cause is set in pjcmd_errcode.
The response information indicates whether or not the request was successful. Whether or not job deletion has been accepted needs to be checked with a result code in the response information by using the pjcmd_get_jobresult_info() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job deletion.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node.
The function can be called from the login node and compute cluster management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_BUSY

An operation cannot be requested because another operation request function is being processed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

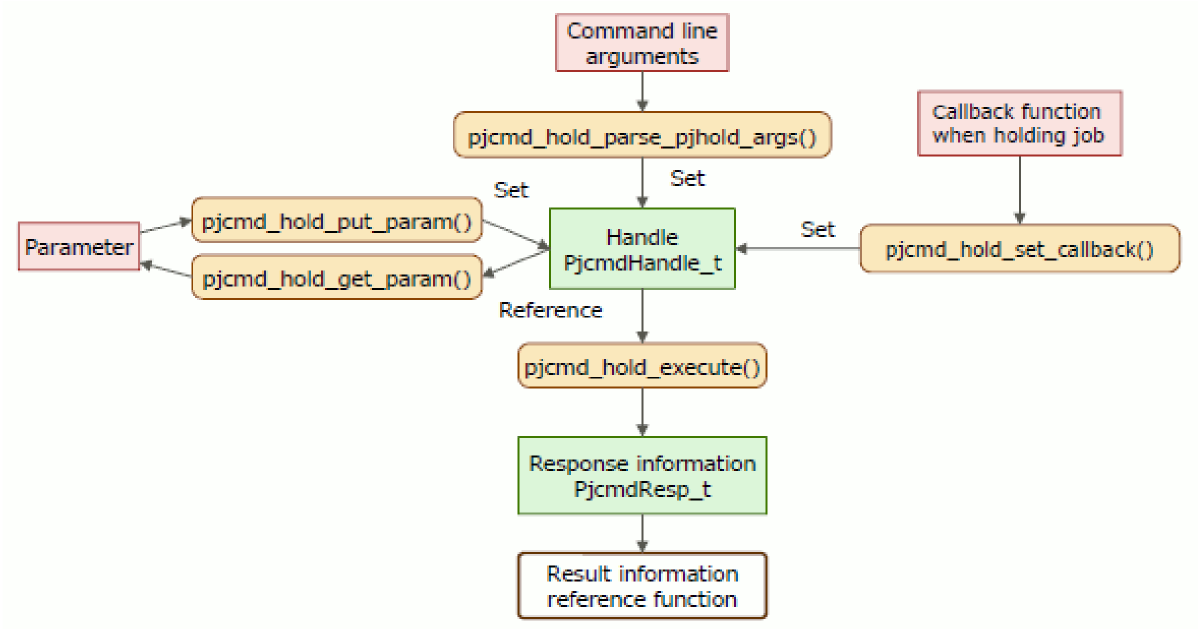
PJCMD_ERROR_INTERNAL

Internal error

B.3 Job Hold

This section describes the functions for holding jobs.

Figure B.4 Requesting Job Hold


Command line arguments
pjcmd_hold_parse_pjhold_args()
Callback function when holding job
Set
Set
Set
Parameter
pjcmd_hold_put_param()
pjcmd_hold_get_param()
Handle PjcmdHandle_t
pjcmd_hold_set_callback()
Reference
pjcmd_hold_execute()
Response information PjcmdResp_t
Result information reference function

B.3.1 pjcmd_hold_parse_pjhold_args()

```
pjcmd_result_t pjcmd_hold_parse_pjhold_args(PjcmdHandle_t *handle_p, int argc, char **argv_pp)
```

This function analyzes command line arguments based on the specification of a pjhold command option and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job hold.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option is not specified.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

Calling this function moves arguments other than options to the end of the *argv_pp*[] array.
When the operation is successfully completed, the pjcmd_optind variable indicates a job ID (the first argument other than options). The caller needs to set the job ID in a handle.
If an unrecognizable option is detected, analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.

B.3.2 pjcmd_hold_put_param()

```
pjcmd_result_t pjcmd_hold_put_param(PjcmdHandle_t *handle_p, pjcmd_hold_param_t param, const void
*val_p)
```

This function sets parameters in a handle that are related to job hold.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be set (See the table below.)

val_p

Pointer to the storage area for a parameter value to be set. For example, if the value type to be set is int type, the caller must prepare a storage area for the int type value and specify a pointer (int *) to the area in *val_p*. If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_HOLD_ENFORCE	This parameter specifies whether or not to interrupt the execution of a prologue or epilogue script if one is being executed. 0: Do not interrupt (Default) 1: Forcibly interrupt. Job will be held	int
PJCMD_HOLD_REASON	Message that is output to job statistical information as a reason to hold the job (equivalent to pjhold --reason) This character string must be within 64 bytes including the NULL character at the end. The available characters are single-byte alphanumeric characters and symbols that can be displayed.	char *
PJCMD_HOLD_LLIO_FLUSH	This parameter specifies whether or not to wait until flushing of unwritten files completes within a time range that does not exceed the limit value of the job execution time before holding the job (equivalent to pjhold --llio-flush). 0: Do not wait until flushing completes (Default) 1: Wait until flushing completes	int
PJCMD_HOLD_HELP	Equivalent to the --help option in the pjhold command 0: Not specified (Default) 1: Specified This parameter does not affect job hold operations.	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job hold.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param*.

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.3.3 pjcmd_hold_get_param()

```
pjcmd_result_t pjcmd_hold_get_param(const PjcmdHandle_t *handle_p, pjcmd_hold_param_t param, void
*val_p)
```

This function references the parameters that are set in a handle for job hold.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_hold_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job hold.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

B.3.4 pjcmd_hold_set_callback()

```
pjcmd_result_t pjcmd_hold_set_callback(
    PjcmdHandle_t *handle_p,
    void (*hold_wait_callback_func_p)(void),
    void (*hold_accept_callback_func_p)(void))
```

This function registers a callback function that is called at a specific time according to the progress of request to job hold. This function is used when a user needs to call their own process at a specific time. For example, this function can be used to output a message indicating that a job is waiting for a hold request to be accepted.
If a NULL pointer is specified as a callback function, it is regarded that the callback function is not set.

[ARGUMENTS]

handle_p

Pointer to a handle

hold_wait_callback_func_p

Pointer to the function that is called when the request to hold a job is pending acceptance. When this function is registered, it is called every three minutes until the request has been accepted.

hold_accept_callback_func_p

Poniter to the function that is called when the job hold request has been accepted.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job hold.

B.3.5 pjcmd_hold_execute()

```
PjcmdResp_t *pjcmd_hold_execute(const PjcmdHandle_t *handle_p)
```

This function requests the job operation management function to hold a job based on a handle. This function can be called from the login node and compute cluster management node.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about a job hold.
The caller must release the obtained response information by using the pjcmd_destroy_resp() function. If a job hold request fails, NULL is returned, and the cause is set in pjcmd_errcode.
The response information indicates whether or not the request was successful. Whether or not the job hold request has been accepted needs to be checked with a result code in the response information by using the pjcmd_get_jobresult_info() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job hold.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node.
The function can be called from the login node and compute cluster management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_BUSY

An operation cannot be requested because another operation request function is being processed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

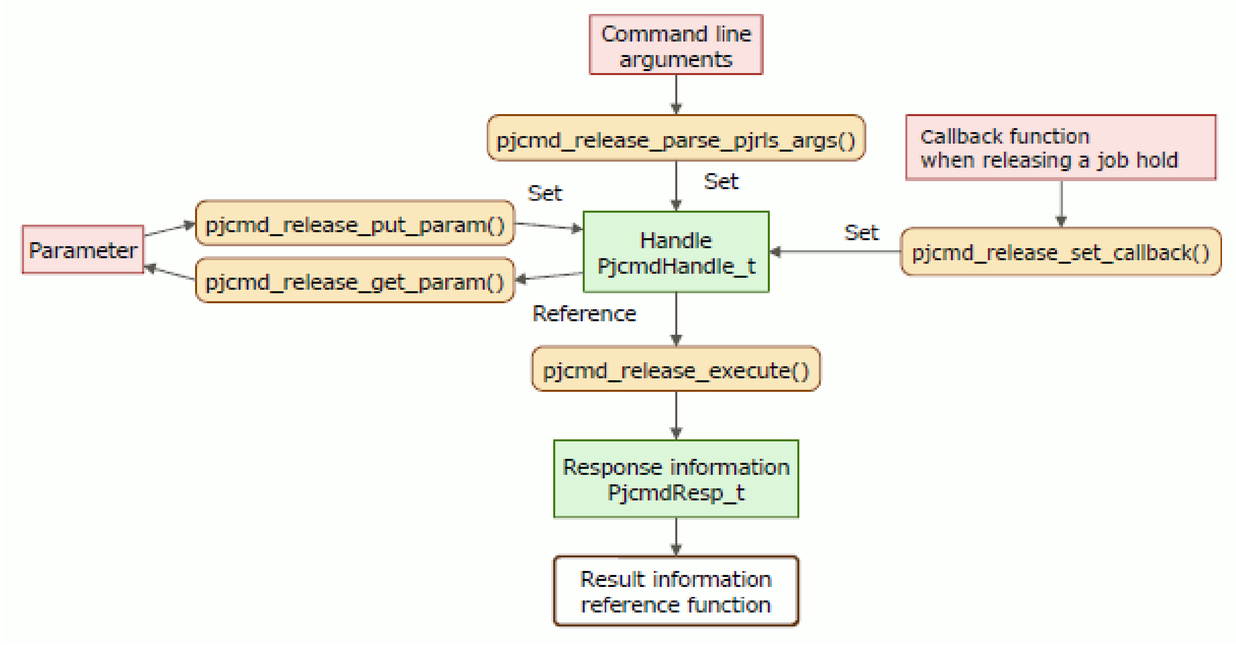
PJCMD_ERROR_INTERNAL

Internal error

B.4 Release of a Job Hold

This section describes the functions for releasing a job hold.

Figure B.5 Requesting the Release of a Job Hold


Command line arguments
pjcmd_release_parse_pjrls_args()
Callback function when releasing a job hold
Set
Set
Set
Parameter
pjcmd_release_put_param()
pjcmd_release_get_param()
Handle PjcmdHandle_t
pjcmd_release_set_callback()
Reference
pjcmd_release_execute()
Response information PjcmdResp_t
Result information reference function

B.4.1 pjcmd_release_parse_pjrls_args()

```
pjcmd_result_t pjcmd_release_parse_pjrls_args(PjcmdHandle_t *handle_p, int argc, char **argv_pp)
```

This function analyzes command line arguments based on the specification of a pjrls command option and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for releasing a job hold.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

Calling this function moves arguments other than options to the end of the *argv_pp*[] array.
When the operation is successfully completed, the pjcmd_optind variable indicates a job ID (the first argument other than options). The caller needs to set the job ID in a handle.
If an unrecognizable option is detected, analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.

B.4.2 pjcmd_release_put_param()

```
pjcmd_result_t pjcmd_release_put_param(PjcmdHandle_t *handle_p, pjcmd_release_param_t param, const
void *val_p)
```

This function sets parameters in a handle that are related to releasing job hold.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be set (See the table below.)

val_p

Pointer to the storage area for a parameter value to be set. For example, if the value type to be set is int type, the caller must prepare a storage area for the int type value and specify a pointer (int *) to the area in *val_p*. If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_RELEASE_HELP	Equivalent to the --help option in the pjrls command 0: Not specified (Default) 1: Specified This parameter does not affect the operation of releasing a job hold.	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for releasing a job hold.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param*.

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.4.3 pjcmd_release_get_param()

```
pjcmd_result_t pjcmd_release_get_param(const PjcmdHandle_t *handle_p, pjcmd_release_param_t param,
void *val_p)
```

This function references the parameters that are set in a handle for releasing job hold.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_release_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for releasing a job hold.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

B.4.4 pjcmd_release_set_callback()

```
pjcmd_result_t pjcmd_release_set_callback(
    PjcmdHandle_t *handle_p,
    void (*release_wait_callback_func_p)(void),
    void (*release_accept_callback_func_p)(void))
```

This function registers a callback function that is called at a specific time according to the progress of request to release the job hold. This function is used when a user needs to call their own process at a specific time. For example, this function can be used to output a message indicating that a job is waiting for a job release request to be accepted.
If a NULL pointer is specified as a callback function, it is regarded that the callback function is not set.

[ARGUMENTS]

handle_p

Pointer to a handle

release_wait_callback_func_p

Pointer to the function that is called when the request to release hold job is pending acceptance. When this function is registered, it is called every three minutes until the request has been accepted.

release_accept_callback_func_p

Poniter to the function that is called when the job release request has been accepted.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for releasing a job hold.

B.4.5 pjcmd_release_execute()

```
PjcmdResp_t *pjcmd_release_execute(const PjcmdHandle_t *handle_p)
```

This function requests the job operation management function to release a job hold based on a handle. This function can be called from the login node and compute cluster management node.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about releasing a job hold.
The caller must release the obtained response information by using the pjcmd_destroy_resp() function. If a request to release a job hold fails, NULL is returned, and the cause is set in pjcmd_errcode.
The response information indicates whether or not the request was successful. Whether or not the request to release a job hold has been accepted needs to be checked with a result code in the response information by using the pjcmd_get_jobresult_info() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for releasing a job hold.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node.
The function can be called from the login node and compute cluster management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_BUSY

An operation cannot be requested because another operation request function is being processed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

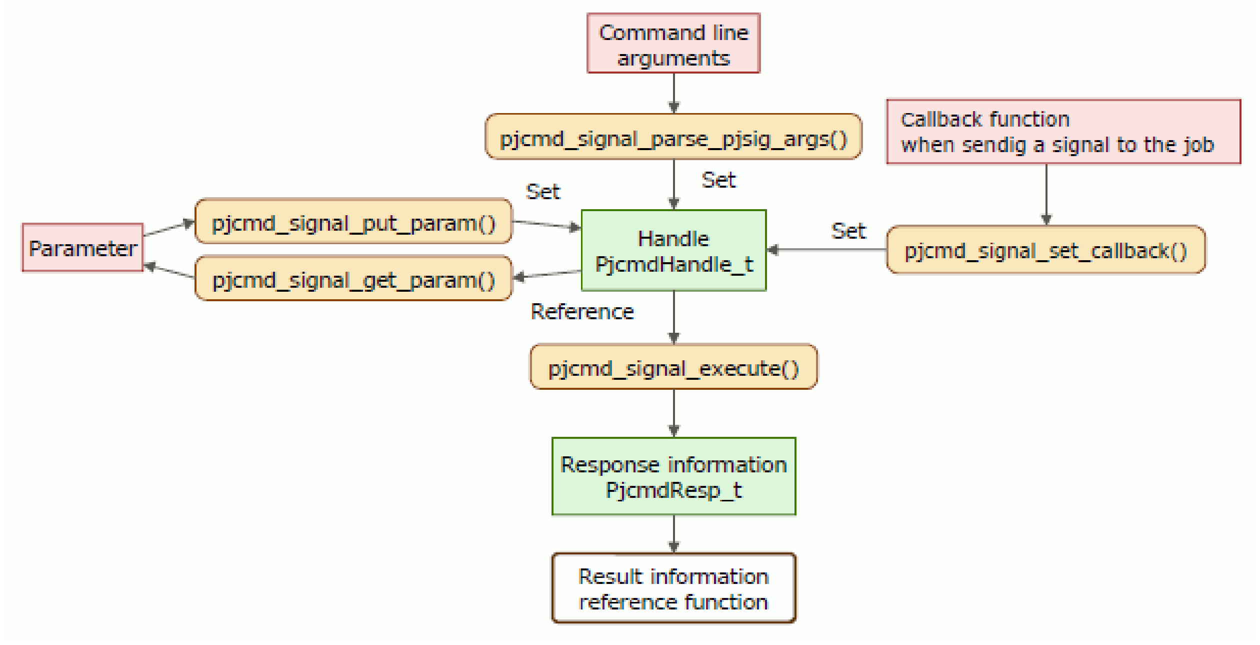
PJCMD_ERROR_INTERNAL

Internal error

B.5 Signal Sending to Jobs

This section describes the functions for sending signals to jobs.

Figure B.6 Requesting the Sending of Signals to a Job


Command line arguments
pjcmd_signal_parse_pjsig_args()
Callback function when sendig a signal to the job
Set
Set
Set
Parameter
pjcmd_signal_put_param()
pjcmd_signal_get_param()
Handle PjcmdHandle_t
pjcmd_signal_set_callback()
Reference
pjcmd_signal_execute()
Response information PjcmdResp_t
Result information reference function

B.5.1 pjcmd_signal_parse_pjsig_args()

```
pjcmd_result_t pjcmd_signal_parse_pjsig_args(PjcmdHandle_t *handle_p, int argc, char **argv_pp)
```

This function analyzes command line arguments based on the specification of a pjsig command option and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for sending a signal.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option is not specified.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

Calling this function moves arguments other than options to the end of the *argv_pp*[] array.
When the operation is successfully completed, the pjcmd_optind variable indicates a job ID (the first argument other than options). The caller needs to set the job ID in a handle.
If an unrecognizable option is detected, analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.

B.5.2 pjcmd_signal_put_param()

```
pjcmd_result_t pjcmd_signal_put_param(PjcmdHandle_t *handle_p, pjcmd_signal_param_t param, const
void *val_p)
```

This function sets parameters in a handle that are related to sending signals to a job.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be set (See the table below.)

val_p

Pointer to the storage area for a parameter value to be set. For example, if the value type to be set is int type, the caller must prepare a storage area for the int type value and specify a pointer (int *) to the area in *val_p*. If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_SIGNAL_SIGNUM	Number of signals to be sent The specified value can be an integer from 1 to 64.	int
PJCMD_SIGNAL_SIGNAME	Name for the signals to be sent The signal name (*) must be recognizable by the destination compute node and within 15 characters (excluding the NULL character at the end). (*) Specifically, the signal name is a name such as "SIGHUP" or "SIGKILL" indicated in the header file signal.h and the man page signal(7).	char *
PJCMD_SIGNAL_HELP	Equivalent to the --help option in the pjsig command 0: Not specified (Default) 1: Specified	int

param	**val_p*	Type of **val_p*
	This parameter does not affect signal sending operations.	

Either the PJCMD_SIGNAL_SIGNUM or PJCMD_SIGNAL_SIGNAME parameter, whichever is set last, is valid. If neither parameter is set in a handle, an error occurs when the pjcmd_signal_execute() function is called.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for sending a signal.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

[Note]

A signal number must be used based on the specifications of the OS on the destination node (compute node executing a job).

B.5.3 pjcmd_signal_get_param()

```
pjcmd_result_t pjcmd_signal_get_param(const PjcmdHandle_t *handle_p, pjcmd_signal_param_t param,
void *val_p)
```

This function references the parameter values that are set in a handle for sending signals.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_signal_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for sending a signal.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

B.5.4 pjcmd_signal_set_callback()

```
pjcmd_result_t pjcmd_signal_set_callback(
    PjcmdHandle_t *handle_p,
    void (*signal_wait_callback_func_p)(void),
    void (*signal_accept_callback_func_p)(void))
```

This function registers a callback function that is called at a specific time according to the progress of sending signal to the job. This function is used when a user needs to call their own process at a specific time. For example, this function can be used to output a message indicating that a request to send a signal to a job is pending acceptance.
If a NULL pointer is specified as a callback function, it is regarded that the callback function is not set.

[ARGUMENTS]

handle_p

Pointer to a handle

signal_wait_callback_func_p

Pointer to the function that is called when the request to send a signal to the job is pending acceptance. When this function is registered, it is called every three minutes until the request has been accepted.

signal_accept_callback_func_p

Poniter to the function that is called when the request to send a signal to the job has been accepted.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for sending a signal.

B.5.5 pjcmd_signal_execute()

```
PjcmdResp_t *pjcmd_signal_execute(const PjcmdHandle_t *handle_p)
```

This function requests the job operation management function to send signals to a job based on a handle. This function can be called from the login node and compute cluster management node.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about signal transmission.
The caller must release the obtained response information by using the pjcmd_destroy_resp() function. If a request to send signals to a job fails, NULL is returned, and the cause is set in pjcmd_errcode.
The response information indicates whether the request succeeded or failed. Whether or not the request to send signals to the job has been accepted by the job operation management function needs to be checked with a result code in the response information by using the pjcmd_get_result() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for sending a signal.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node.
The function can be called from the login node and compute cluster management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_BUSY

An operation cannot be requested because another operation request function is being processed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

PJCMD_ERROR_INTERNAL

Internal error

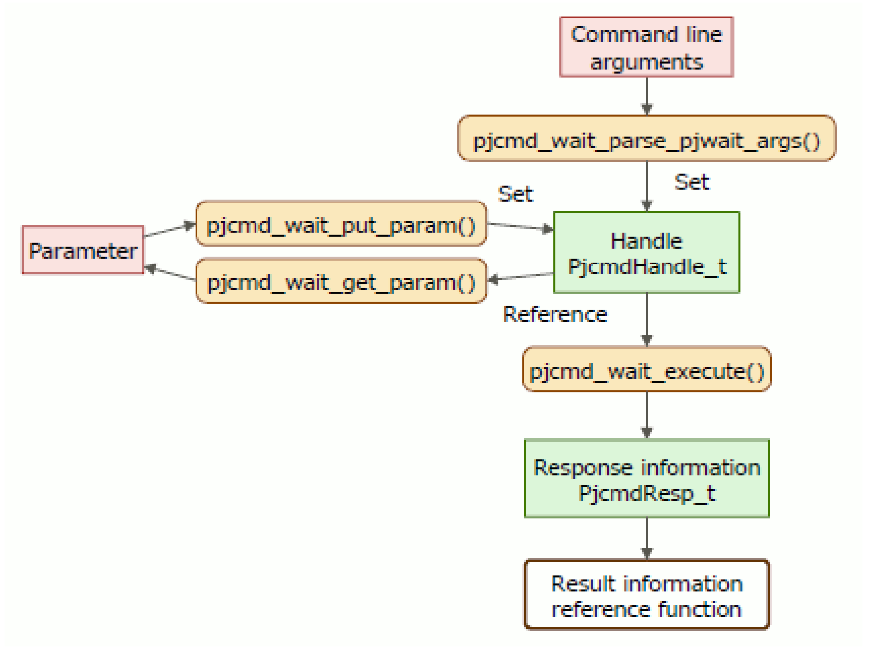
[Note]

Since the signal transmission request function can only make requests to send signals, an error does not occur even if the specified signal number cannot be recognized by the destination node. The user needs to check whether or not the request has been sent successfully.

B.6 Waiting for Job Completion

This section describes the functions for waiting for a job to complete.

Figure B.7 Request to Wait for a Job to Complete



B.6.1 pjcmd_wait_parse_pjwait_args()

```
pjcmd_result_t pjcmd_wait_parse_pjwait_args(PjcmdHandle_t *handle_p, int argc, char **argv_pp)
```

This function analyzes command line arguments based on the specification of a pjwait command option and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for waiting for job completion.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option is not specified.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

Calling this function moves arguments other than options to the end of the *argv_pp*[] array.
When the operation is successfully completed, the pjcmd_optind variable indicates a job ID (the first argument other than options). The caller needs to set the job ID in a handle.
If an unrecognizable option is detected, analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.

B.6.2 pjcmd_wait_put_param()

```
pjcmd_result_t pjcmd_wait_put_param(PjcmdHandle_t *handle_p, pjcmd_wait_param_t param, const void
*val_p)
```

This function sets parameters in a handle that are related to waiting for job completion.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be set (See the table below.)

val_p

Pointer to the storage area for a parameter value to be set. For example, if the value type to be set is int type, the caller must prepare a storage area for the int type value and specify a pointer (int *) to the area in *val_p*. If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_WAIT_MODE	This parameter specifies a mode for waiting for job completion, namely the return condition of the pjcmd_wait_execute() function (equivalent to pjwait -w). - PJCMD_WAIT_ALL The function returns when all jobs are completed. (Default) - PJCMD_WAIT_ANY The function returns when at least one job is completed. - PJCMD_WAIT_NONE The function returns immediately without waiting for a job to complete.	int
PJCMD_WAIT_Z	Equivalent to the -z option in the pjwait command 0: Not specified (Default) 1: Specified	int

param	**val_p*	Type of **val_p*
	This parameter does not affect the operation of waiting for job completion.	
PJCMD_WAIT_HELP	Equivalent to the --help option in the pjwait command 0: Not specified (Default) 1: Specified This parameter does not affect the operation of waiting for job completion.	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for waiting for job completion.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.6.3 pjcmd_wait_get_param()

```
pjcmd_result_t pjcmd_wait_get_param(const PjcmdHandle_t *handle_p, pjcmd_wait_param_t param, void
*val_p)
```

This function references the parameters that are set in a handle for waiting for job completion.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_wait_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for waiting for job completion.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

B.6.4 pjcmd_wait_execute()

```
PjcmdResp_t *pjcmd_wait_execute(const PjcmdHandle_t *handle_p)
```

This function requests the job operation management function to wait for job completion based on a handle. This function can be called from the login node and compute cluster management node.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about waiting for job completion.
The caller must release the obtained response information by using the pjcmd_destroy_resp() function. If a request to wait for job completion fails, NULL is returned, and the cause is set in pjcmd_errcode.
The response information indicates whether the request succeeded or failed. The results of waiting for job completion need to be checked with a result code in the response information by using the pjcmd_get_jobresult_info() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for waiting for job completion.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node.
The function can be called from the login node and compute cluster management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_BUSY

An operation cannot be requested because another operation request function is being processed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

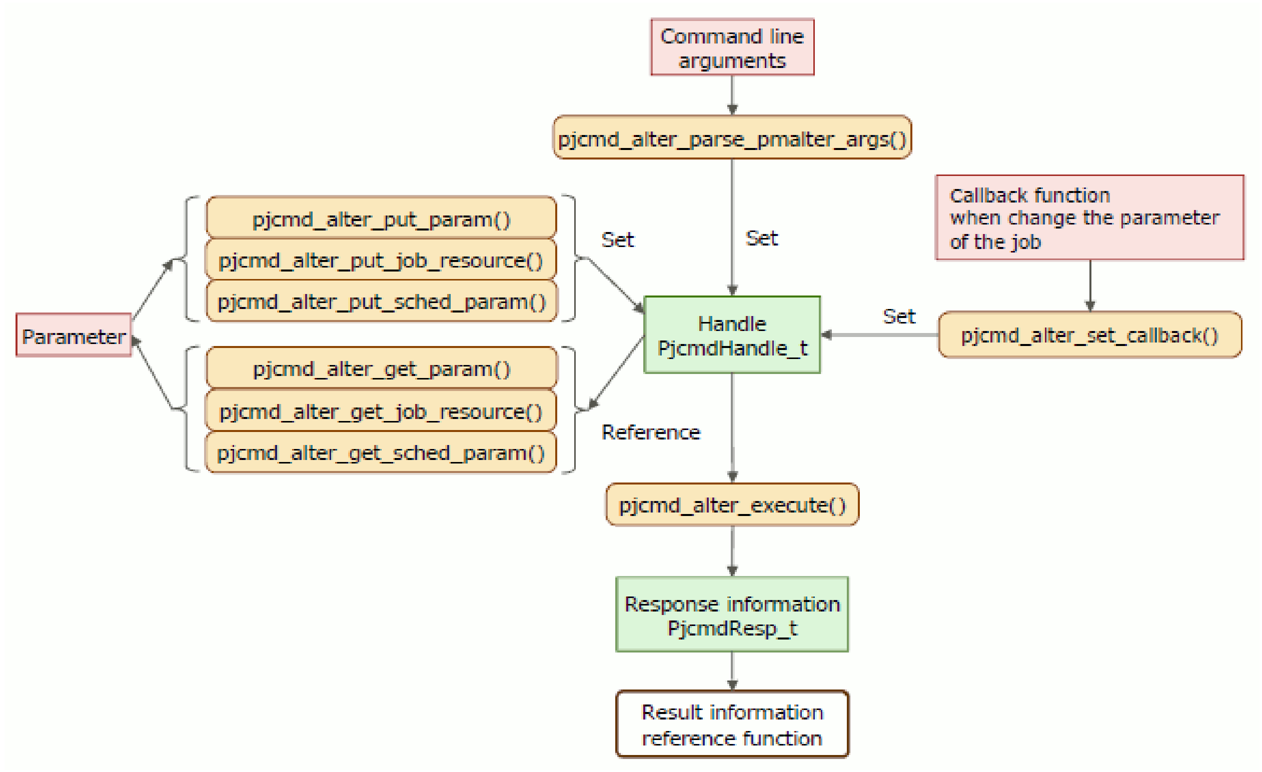
PJCMD_ERROR_INTERNAL

Internal error

B.7 Job Parameter Change

This section describes the functions for changing job parameters.

Figure B.8 Request to Change a Job Parameter


Command line arguments
pjcmd_alter_parse_pmalter_args()
pjcmd_alter_put_param()
pjcmd_alter_put_job_resource()
pjcmd_alter_put_sched_param()
Set
Set
Callback function when change the parameter of the job
Parameter
Handle PjcmdHandle_t
Set
pjcmd_alter_set_callback()
pjcmd_alter_get_param()
pjcmd_alter_get_job_resource()
pjcmd_alter_get_sched_param()
Reference
pjcmd_alter_execute()
Response information PjcmdResp_t
Result information reference function

B.7.1 pjcmd_alter_parse_pmalter_args()

```
pjcmd_result_t pjcmd_alter_parse_pmalter_args(PjcmdHandle_t *handle_p, int argc, char **argv_pp)
```

This function analyzes command line arguments based on the specifications of pjalter and pmalter command options and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for changing a job parameter.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option is not specified.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

Calling this function moves arguments other than options to the end of the *argv_pp*[] array.
When the operation is successfully completed, the pjcmd_optind variable indicates a job ID (the first argument other than options). The caller needs to set the job ID in a handle.
If an unrecognizable option is detected, analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.

[Note]

This function can set a handle for both the pjalter and pmalter command options. However, if the -P option, which can be specified only in the pmalter command for the administrators, is set in a handle, administrator privileges are required for requesting the job operation management function to change a job parameter (the pmcmd_alter_execute() function).

B.7.2 pjcmd_alter_put_param()

```
pjcmd_result_t pjcmd_alter_put_param(PjcmdHandle_t *handle_p, pjcmd_alter_param_t param, const void
*val_p)
```

This function sets the job parameter to be changed in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be set (See the table below.)

val_p

Pointer to the storage area for a parameter value to be set. For example, if the value type to be set is char * type, the caller must prepare a storage area for the char * type value and specify a pointer (char **) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_ALTER_CLUSTER	Cluster name (only one name, equivalent to pmalter -c) If this parameter is not set, the value of the environment variable PXMYCLST is applied. If the variable also does not have a set value, an error occurs in the pjcmd_alter_execute() function. This parameter is valid only when called from the system management node. If this parameter is called from a node other than the system management node, the setting is ignored and the name of the cluster to which the node belongs is applied.	char *
PJCMD_ALTER_HELP	Equivalent to the specifications of the --help option in the pjalter or pmalter command 0: Not specified (Default) 1: Specified This parameter does not affect job parameter changes.	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for changing a job parameter.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.7.3 pjcmd_alter_get_param()

```
pjcmd_result_t pjcmd_alter_get_param(const PjcmdHandle_t *handle_p, pjcmd_alter_param_t param, void
*val_p)
```

This function references the set details in a handle for changing job parameters.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_alter_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for changing a job parameter.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

B.7.4 pjcmd_alter_put_job_resource()

```
pjcmd_result_t pjcmd_alter_put_job_resource(PjcmdHandle_t *handle_p, char *rscname_p, const void
*val_p)
```

This function sets the job resource name to be changed and the value after the change in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

rscname_p

Name of the resource whose value needs to be changed (See the table below.)

val_p

Pointer to the storage area of the resource value to be changed. For example, if the resource to be changed is "elapse," the caller needs to prepare a variable where the value after the change (time_t type) is stored, and specify a pointer to the variable (time_t *) in *val_p*. If NULL is specified, the parameter value is initialized (not set).

rscname_p	**val_p*	Type of **val_p*
"elapse"	Limit value of job executable time The specifiable value ranges from 1 to 2147483647 seconds. If the time is not limited, PJCMD_UNLIMITED is specified.	time_t
"rscunit"	Name of the resource unit to which a job is submitted	char *
"rscgrp"	Name of the resource group to which a job is submitted	char *

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for changing a job parameter.

PJCMD_ERROR_INVALID_ARGUMENT

rscname_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

**val_p* is invalid.

- A specification method is incorrect.
- The value (character string) that is specified as the "rscunit" or "rscgrp" parameter is NULL.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.7.5 pjcmd_alter_get_job_resource()

```
pjcmd_result_t pjcmd_alter_get_job_resource(const PjcmdHandle_t *handle_p, char *rscname_p, void
*val_p)
```

This function references the job resource amount that is set to be changed in a handle for changing job parameters.

[ARGUMENTS]

handle_p

Pointer to a handle

rscname_p

The name of resource to be referenced. The value that can be specified are the same as those for the pjcmd_alter_put_job_resource) function.

val_p

A value is stored in **val_p* based on the *racuname_p* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for changing a job parameter.

PJCMD_ERROR_INVALID_ARGUMENT

rscname_p or *val_p* is invalid (NULL).

PJCMD_ERROR_NODATA

A specified resource is not set in a handle.

B.7.6 pjcmd_alter_put_sched_param()

```
pjcmd_result_t pjcmd_alter_put_sched_param(PjcmdHandle_t *handle_p, pjcmd_alter_sched_param_t param,
const void *val_p)
```

This function sets parameters in a handle that are used to change job scheduling.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be changed (See the table below.)

val_p

Pointer to the storage area for a parameter value to be set. For example, if the value type to be set is int type, the caller must prepare a storage area for the int type value and specify a pointer (int *) to the area in *val_p*. If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_ALTER_SCHED_PRIORITY	Priority after changing jobs for the same users The specified value can be an integer from 0 to 255.	int
PJCMD_ALTER_SCHED_APRIORITY	Priority after changing jobs in a resource unit The specified value can be an integer from 0 to 255.	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for changing a job parameter.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

B.7.7 pjcmd_alter_get_sched_param()

```
pjcmd_result_t pjcmd_alter_get_sched_param(const PjcmdHandle_t *handle_p, pjcmd_alter_sched_param_t
param, void *val_p)
```

This function references scheduling-related parameter values that are set in a handle for changing job parameters.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_alter_put_sched_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for changing a job parameter.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

B.7.8 pjcmd_alter_set_callback()

```
pjcmd_result_t pjcmd_alter_set_callback(
    PjcmdHandle_t *handle_p,
    void (*alter_wait_callback_func_p)(void),
    void (*alter_accept_callback_func_p)(void))
```

This function registers a callback function that is called at a specific time according to the progress of request to change the job paramter. This function is used when a user needs to call their own process at a specific time. For example, this function can be used to output a message indicating that a request to change the parameter of the parameter is pending acceptance.
If a NULL pointer is specified as a callback function, it is regarded that the callback function is not set.

[ARGUMENTS]

handle_p

Pointer to a handle

alter_wait_callback_func_p

Pointer to the function that is called when the request to change the job parameter is pending acceptance. When this function is registered, it is called every three minutes until the request has been accepted.

alter_accept_callback_func_p

Poniter to the function that is called when the request to change the job parameter has been accepted.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for changing a job parameter.

B.7.9 pjcmd_alter_execute()

```
PjcmdResp_t *pjcmd_alter_execute(const PjcmdHandle_t *handle_p)
```

This function requests the job operation management to change a job parameter based on a handle. This function can be called from the login node, system management node, and computer cluster management node.
If the priority of jobs in the resource unit is set in the handle as the parameter to be changed, administrator (root) privileges are required.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about changing a job parameter.
The caller must release the obtained response information by using the pjcmd_destroy_resp() function. If a request to change a job parameter fails, NULL is returned, and the cause is set in pjcmd_errcode.
The response information indicates whether the request succeeded or failed. Whether or not the request to change a job parameter has

been accepted by the job operation management function needs to be checked with a result code in the response information by using the pjcmd_get_jobresult_info() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for changing a job parameter.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node.
The function can be called from the login node, system management node and compute cluster management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_BUSY

An operation cannot be requested because another operation request function is being processed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted. Alternatively, an attempt to change the job priority in a resource unit was made with privileges other than administrator privileges.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

PJCMD_ERROR_INTERNAL

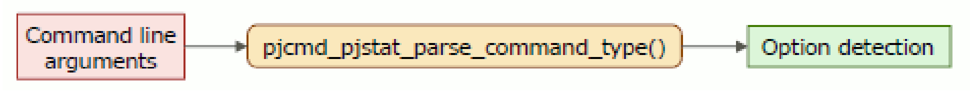
Internal error

Appendix C Information Acquisition API Reference

C.1 Common Information of the Information Acquisition API

This section describes utility functions for getting job information and resource information.

Figure C.1 Detecting pjstat Command Arguments



C.1.1 pjcmd_pjstat_parse_command_type()

```
pjcmd_pjstat_command_type_t pjcmd_pjstat_parse_command_type(int argc, const char * const *argv_pp)
```

This function analyzes command line arguments as pjstat command arguments and determines whether or not the --rsc option, --limit option, and --help option exist.
The function is used to individually call a job information acquisition API, resource information acquisition API, resource status acquisition API or display the method of use, based on the pjstat command option specified in a command line argument.

[ARGUMENTS]

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_PJSTAT_SHOW_JOB

The command argument corresponds to displaying job information (pjstat).

PJCMD_PJSTAT_SHOW_RESOURCE

The command argument corresponds to displaying job resource information (pjstat --rsc).

PJCMD_PJSTAT_SHOW_LIMIT

The command argument corresponds to displaying a limit value when submitting a job (pjstat --limit).

PJCMD_PJSTAT_SHOW_HELP

The command argument corresponds to displaying the method of use (pjstat --help).

PJCMD_PJSTAT_UNKNOWN_COMMAND_TYPE

Nothing could be determined from the command argument.

[pjcmd_errcode]

PJCMD_SUCCESS

Success. This code is set when the return value is PJCMD_PJSTAT_SHOW_JOB, PJCMD_PJSTAT_SHOW_RESOURCE, PJCMD_PJSTAT_SHOW_LIMIT, or PJCMD_PJSTAT_SHOW_HELP.

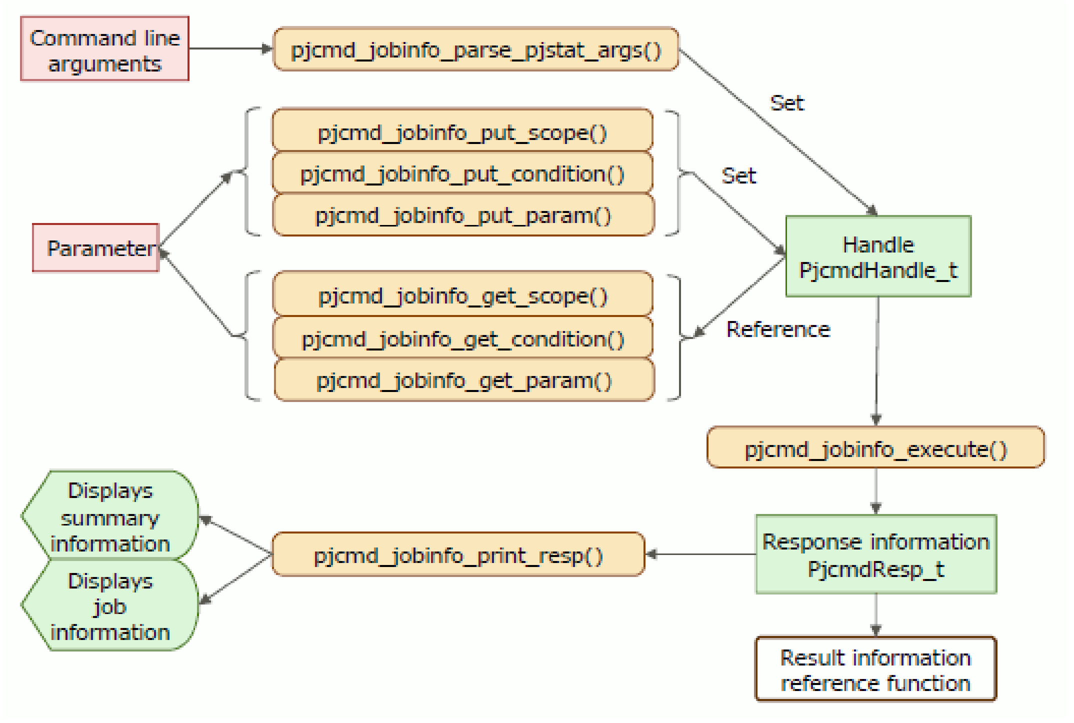
PJCMD_ERROR_UNKNOWN_OPTION

Nothing could be determined from the command argument. This code is set when the return value is PJCMD_PJSTAT_UNKNOWN_COMMAND_TYPE.

C.2 Getting Job Information

This section describes the functions for getting job information (job statistical information).

Figure C.2 Requesting Job Information Acquisition



Command line arguments
pjcmd_jobinfo_parse_pjstat_args()
Set
pjcmd_jobinfo_put_scope()
pjcmd_jobinfo_put_condition()
pjcmd_jobinfo_put_param()
Set
Parameter
Handle PjcmdHandle_t
pjcmd_jobinfo_get_scope()
pjcmd_jobinfo_get_condition()
pjcmd_jobinfo_get_param()
Reference
pjcmd_jobinfo_execute()
Displays summary information
Displays job information
pjcmd_jobinfo_print_resp()
Response information PjcmdResp_t
Result information reference function

Figure C.3 Referencing an Information Group and Getting Job Information

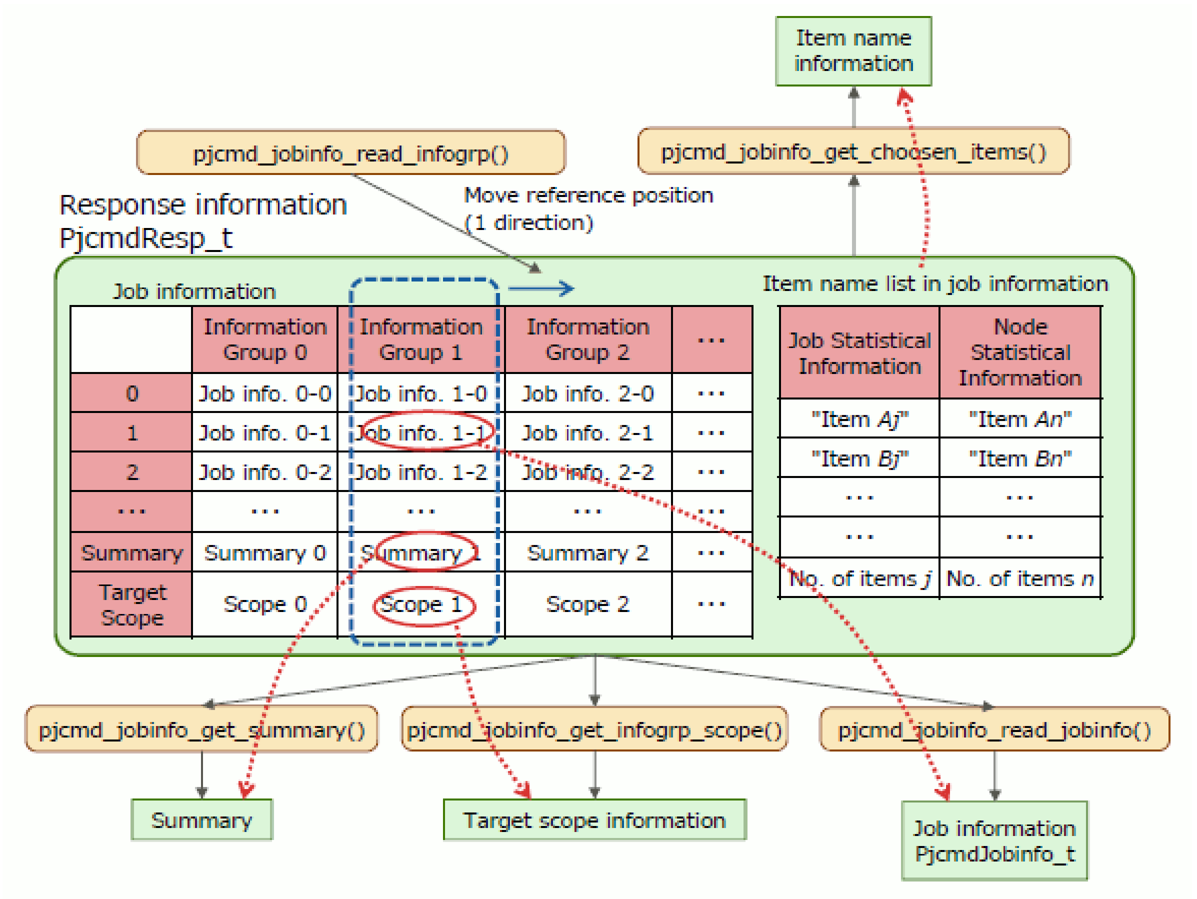
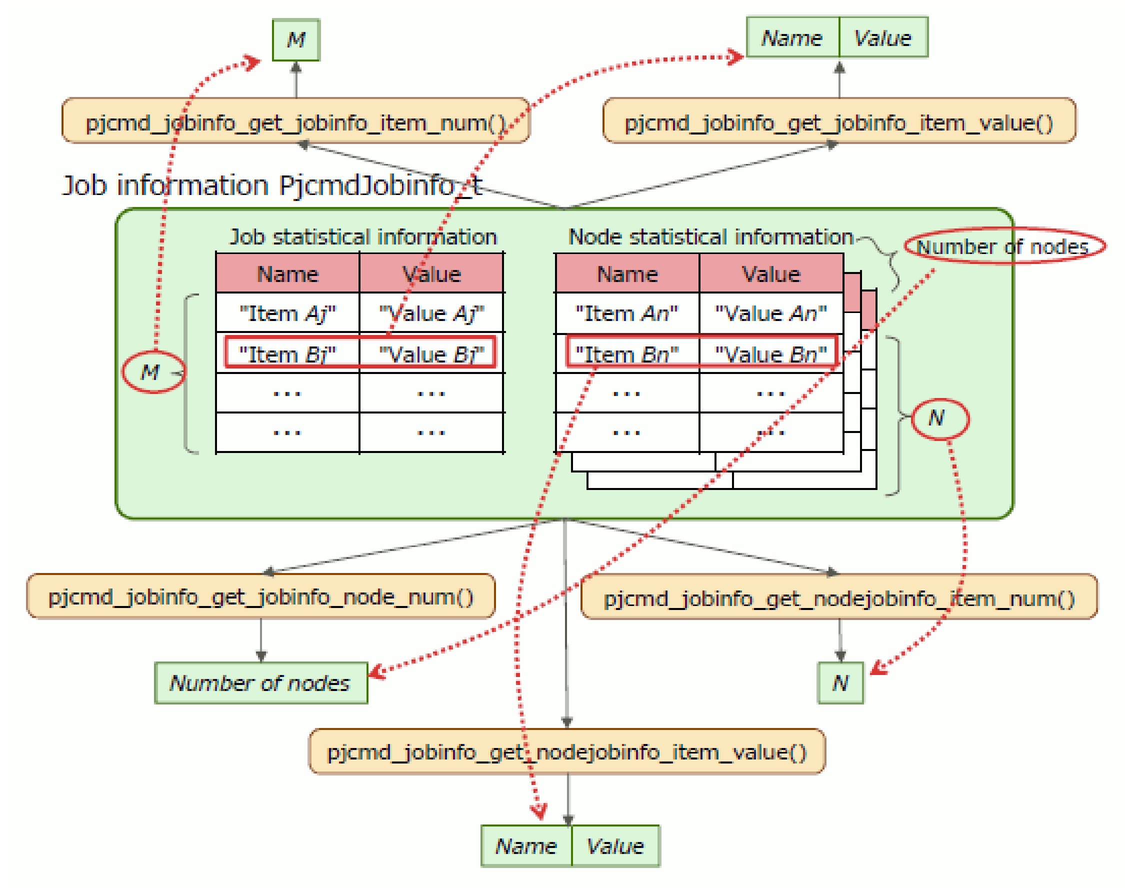


Figure C.4 Referencing Job Information


M
Name
Value
pjcmd_jobinfo_get_jobinfo_item_num()
pjcmd_jobinfo_get_jobinfo_item_value()
Job information PjcmdJobinfo_t
Job statistical information
Node statistical information
Number of nodes
Name
Value
"Item Aj"
"Value Aj"
"Item Bj"
"Value Bj"
"Item An"
"Value An"
"Item Bn"
"Value Bn"
M
N
pjcmd_jobinfo_get_jobinfo_node_num()
pjcmd_jobinfo_get_nodejobinfo_item_num()
Number of nodes
N
pjcmd_jobinfo_get_nodejobinfo_item_value()
Name
Value

C.2.1 pjcmd_jobinfo_parse_pjstat_args()

```
pjcmd_result_t pjcmd_jobinfo_parse_pjstat_args(PjcmdHandle_t *handle_p, int argc, char **argv_pp)
```

This function analyzes command line arguments based on the specification of a pjstat command option and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job information acquisition.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option is not specified.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

Calling this function moves arguments other than options to the end of the *argv_pp*[] array.
If an unrecognizable option is detected, analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.
This function treats the --rsc option and --limit option in the pjstat command as well as other options that can be specified only with these options as invalid options.

C.2.2 pjcmd_jobinfo_put_scope()

```
pjcmd_result_t pjcmd_jobinfo_put_scope(PjcmdHandle_t *handle_p, pjcmd_scope_t scope, const void
*val_p, uint32_t n)
```

This function sets the target range for getting job information in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier indicating the type of target range for getting information (See the following table.)

val_p

Pointer to the storage area for the value indicating the target to get job information. For example, if the value type to be set is char * type, the caller must prepare a storage area for the char * type value and specify a pointer (char **) to the area in *val_p*. If NULL is specified, the parameter value is initialized (not set).

n

Number of elements of *val_p*

scope	**val_p*	Type of **val_p*
PJCMD_SCOPE_CLUSTER	Cluster name (only one name, equivalent to pjstat -c) If this parameter is not set, the PXMYCLST environment variable value is applied. If the parameter and the environment variable are not set, an error occurs in the pjcmd_jobinfo_execute() function. This parameter is valid only when called from the system management node. When the parameter is called from a node other than the system management node, the cluster to which the	char *

scope	**val_p*	Type of **val_p*
	calling node belongs is applied. 1 must be specified in *n* for the parameter.	
PJCMD_SCOPE_RSCUNIT	This parameter gets information on each resource unit and specifies resource unit names as an array (number of elements is *n*, equivalent to pjstat --rscunit). If "*" is specified as a resource unit name, all resource units are targeted.	char **
PJCMD_SCOPE_RSCGRP	This parameter gets information on each resource group and specifies resource group names as an array (number of elements is *n*, equivalent to pjstat --rscgrp). If "*" is set as a resource group name, all resource groups are targeted.	char **

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job information acquisition.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown or unspecifiable value is specified in *scope.*

PJCMD_ERROR_INVALID_PARAM

val_p or *n* is invalid.

C.2.3 pjcmd_jobinfo_get_scope()

```
pjcmd_result_t pjcmd_jobinfo_get_scope(const PjcmdHandle_t *handle_p, pjcmd_scope_t scope, void
*val_p, uint32_t *n_p)
```

This function references the target range for getting the information that is set in a handle for job information acquisition.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier of the target range to be referenced. The identifiers that can be specified are the same as those for the pjcmd_jobinfo_put_scope() function.

val_p

A value is stored in **val_p* based on the *scope* type. The caller needs to prepare an area of a sufficient size according to the value type.

n_p

The number of elements of *val_p* is stored in **n_p*. The caller needs to prepare the area.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job information acquisition.

PJCMD_ERROR_INVALID_ARGUMENT

val_p or *n_p* is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *scope.*

PJCMD_ERROR_NODATA

A specified *scope* is not set in a handle.

C.2.4 pjcmd_jobinfo_put_condition()

```
pjcmd_result_t pjcmd_jobinfo_put_condition(PjcmdHandle_t *handle_p, pjcmd_jobinfo_condition_type_t
type, const char **item_pp, int n)
```

This function sets conditions for the job information to be obtained in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

type

Type of condition to get job information (See the following table.)

item_pp

Array of an acquisition condition item (character string). The value to be specified varies depending on the *type* argument. (See the following table.)
If NULL is specified, a parameter value is initialized (not set).

n

Number of elements in the *item_pp*[] array

type	*item_pp*[]
PJCMD_JOBINFO_GROUPING	This parameter specifies a unit to group information (unit to obtain). "usr": Information is grouped by user. "grp": Information is grouped by group. "rscu": Information is grouped by resource unit. "rscg": Information is grouped by resource group. Multiple units can be specified as the *item_pp*[] array. Example: Specifying "usr" and "rscg" indicates that information is grouped by the same user and resource group.

type	*item_pp*[]
	If this parameter is not specified, information is not grouped by a specific unit. If a resource unit or resource group is specified in the pjcmd_jobinfo_put_scope() function, it is regarded that "rscu" or "rscg" is specified.
PJCMD_JOBINFO_CHOOSE PJCMD_JOBINFO_NOT_CHOOSE	The information to be obtained is selected out of job information (*). (*) Information that can be obtained by the command API is the same as the information that can be output by the pjstat command (see pjstatsinfo(7).) The PJCMD_JOBINFO_CHOOSE parameter specifies the information to be obtained (equivalent to pjstat --choose). The PJCMD_JOBINFO_NOT_CHOOSE parameter specifies information that is not obtained. Either parameter can be specified. If neither parameter is specified, all information is obtained. The item name of the information to be obtained is specified in the *item_pp*[] array. Only the job information items that are listed in the Table C.1 Items That Can be Specified as Conditions to Get Job Information can be specified. Multiple item names can be specified.
PJCMD_JOBINFO_FILTER	Only the job information for the specified items with specific values is obtained (equivalent to pjstat --filter). Multiple conditions can be specified using the *item_pp*[] array. The condition is specified in one array element in the "*item* = *value*" format. An item name listed in the Table C.1 Items That Can be Specified as Conditions to Get Job Information can be specified in *item*. A wild card and an expression to specify a range can be used for *value* (based on the --filter option format of the pjstat command). If a resource unit and resource group are specified in the pjcmd_jobinfo_put_scope() function, it is regarded that a corresponding item (rscu or rscg) is specified.
PJCMD_JOBINFO_SORT	This parameter specifies conditions to sort the information to be obtained (equivalent to pjstat --sort). Multiple conditions can be specified using the *item_pp*[] array. The conditions are specified in one array element in the "*item*:*order*" format. An item name listed in the Table C.1 Items That Can be Specified as Conditions to Get Job Information can be specified in *item*. "A" (ascending order) or "B" (descending order) can be specified in *order*. If specification of the job information order is omitted, it is sorted in job ID order.

The following table lists the item names that can be specified as conditions to get job information (PJCMD_JOBINFO_CHOOSE, PJCMD_JOBINFO_NOT_CHOOSE, PJCMD_JOBINFO_FILTER, and PJCMD_JOBINFO_SORT).

Table C.1 Items That Can be Specified as Conditions to Get Job Information

Item	Description
jid	Job ID or sub job ID
snum	Number of sub jobs of a bulk job or step job
jnam	Job name
jtyp	Job type
jmdl	Job model
usr	Name of user executing job
grp	Name of group executing job
rscu	Resource unit
rscg	Resource group

Item	Description
pri	Job priority
sh	Shell path name
mail	E-mail send flag
adr	E-mail send destination address
sde	Step job dependency relational expression
mask	umask value of user submitting job
std	Standard output file name
stde	Standard error output file path
infop	Statistical information file path
pcl	CPU usage time limit by process
pcfl	Core file limit by process
pcpl	Max user process count limit by process
pdl	Data segment limit by process
prml	Lock memory size limit by process
pmql	POSIX message queue size limit by process
pofl	File descriptor limit by process
ppsl	Signal count limit by process
ppl	File size limit by process
psl	Stack segment limit by process
pvml	Virtual memory size limit by process
cmt	Comment of job
rnum	Retry count of job
lst	Previous state of job
st	Current state of job
adt	Job submission time
qdt	Last queuing time
sdt	Job execution start time
edt	Job execution end time
exc	EXIT/CANCEL state transition time
thldtm	Accumulated hold time of job
lhusr	Last hold user name
holnm	Hold count of job
prmdt	Time when the job resource manager function collects data from each node
ec	End code of shell script
sn	Signal number
pc	PJM code
ermsg	Error message (REASON)
elpl	Elapsed time limit of job (Executable time)
elp	Execution elapsed time

Item	Description
uctmut	Total user CPU time of job
sctmut	Total system CPU time of job
usctmut	Total user CPU time and total of system CPU time
mszl	Physical memory amount limit by node
msza	Physical memory amount allocated to a node
mmszu	Total max physical memory usage of nodes
cnumr	Requested number of CPUs
cnumat	Total number of CPUs allocated to job
cnumut	Total number of CPUs used in job
nnumr	Number of nodes and its shape requested by job
nnuma	Number of nodes and its shape allocated to job
nnumu	Number of nodes used by job
nnumv	Number of unavailable nodes within the allocated nodes
nidlu	Node ID list of the nodes used by job
tofulu	Tofu coordinate list of the nodes used by job

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job information acquisition.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *type.*

PJCMD_ERROR_NOMEM

Memory acquisition failed.

C.2.5 pjcmd_jobinfo_get_condition()

```
pjcmd_result_t pjcmd_jobinfo_get_condition(const PjcmdHandle_t *handle_p,
pjcmd_jobinfo_condition_type_t type, char ***item_ppp, int *n_p)
```

This function references the contents of information acquisition conditions that are set in a handle for job information acquisition.

[ARGUMENTS]

handle_p

Pointer to a handle

type

Identifier indicating the conditions to get job information to be referenced. The identifiers that can be specified are the same as those of the pjcmd_jobinfo_put_condition() function.

item_ppp

The item names specified in *type* for information acquisition conditions are stored in the character string array (**item_ppp*)[]. (**item_ppp*)[] indicates an area that is set in a handle.

n_p

The number of elements in the character string array (**item_ppp*)[] is stored in **n_p.*

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job information acquisition.

PJCMD_ERROR_INVALID_ARGUMENT

item_ppp or *n_p* is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *type.*

PJCMD_ERROR_NODATA

The acquisition conditions specified in *type* are not set.

C.2.6 pjcmd_jobinfo_put_param()

```
pjcmd_result_t pjcmd_jobinfo_put_param(PjcmdHandle_t *handle_p, pjcmd_jobinfo_param_t param, void
*val_p)
```

This function sets parameters related to job information acquisition.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifiers of a parameter to be set that is related to job information acquisition (See the following table.)

val_p

Pointer to the storage area for the parameter value to be set. For example, if the value type to be set is int type, the caller must prepare a storage area for the int type value and specify a pointer (int *) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_JOBINFO_USER_ID	This parameter specifies the array of the target user ID whose information is obtained. The last element in the array must be -1.	uid_t *

param	**val_p*	Type of **val_p*
PJCMD_JOBINFO_USER_NAME	This parameter specifies the target user name whose information is obtained. The last element in the array must be NULL.	char **
PJCMD_JOBINFO_GROUP_ID	This parameter specifies the target group ID whose information is obtained. The last element in the array must be -1.	gid_t *
PJCMD_JOBINFO_GROUP_NAME	This parameter specifies the target group name whose information is obtained. The last element in the array must be NULL.	char **
PJCMD_JOBINFO_HISTORY_DAY	This parameter specifies information on a job that has completed within the specified number of days (equivalent to pjstat -H and -H day=*n*). An integer value from 1 to 365 can be specified. If -1 is specified, this parameter gets information on jobs that were completed within the past 3 days (equivalent to a case where the pjstat command does not have the -H option argument).	int
PJCMD_JOBINFO_HISTORY_START	This parameter specifies the start date of a period to get information on completed jobs (equivalent to pjstat -H start=*date*).	time
PJCMD_JOBINFO_HISTORY_END	This parameter specifies the end date of a period to get information on completed jobs (equivalent to pjstat -H end=*date*).	time
PJCMD_JOBINFO_HISTORY_PERIOD	This parameter specifies jobs that were completed within the specified number of days after the first day specified in the PJCMD_JOBINFO_HISTORY_START parameter (equivalent to pjstat -H period=*days*). The value range that can be specified is from 1 to 2147483647 days.	int
PJCMD_JOBINFO_VERBOSITY	This parameter specifies the granularity of the information to be obtained. - PJCMD_JOBINFO_VERBOSITY_JOB This gets only job information. (Default) - PJCMD_JOBINFO_VERBOSITY_SUBJOB This gets job information and sub job information (equivalent to pjstat -E).	int
PJCMD_JOBINFO_SUMMARY	This parameter specifies whether or not to get summary information when getting job information. - PJCMD_JOBINFO_NO_SUMMARY Summary information is not obtained. (Default) - PJCMD_JOBINFO_WITH_SUMMARY Both summary information and job information are obtained (equivalent to pjstat --with-summary). - PJCMD_JOBINFO_ONLY_SUMMARY Only summary information is obtained (equivalent to pjstat --summary).	int
PJCMD_JOBINFO_LEVEL	Level of information to be obtained	int

param	**val_p*	Type of **val_p*
	- PJCMD_JOBINFO_LEVEL_0 Only basic information is obtained (equivalent to pjstat, default). - PJCMD_JOBINFO_LEVEL_1 Basic information and additional information are obtained (equivalent to pjstat -v). - PJCMD_JOBINFO_LEVEL_2 Job statistical information is obtained (equivalent to pjstat -s). - PJCMD_JOBINFO_LEVEL_3 Job statistical information and node statistical information are obtained (equivalent to pjstat -S).	
PJCMD_JOBINFO_LEVEL_PATTERN	If the level of information to be obtained is PJCMD_JOBINFO_LEVEL_1, this parameter specifies additional information to be obtained (equivalent to pjstat -v pattern=*n*). Currently, only 1 can be specified.	uint32_t
PJCMD_JOBINFO_OTHERSJOB	This parameter specifies whether or not to get job information submitted by other users (users other than the user who called this function) (equivalent to pjstat -A). - PJCMD_JOBINFO_OTHERSJOB_NONE Job information submitted by other users is not obtained. (Default) - PJCMD_JOBINFO_OTHERSJOB_ALL Job information submitted by other users is also obtained. However, without privileges, some information cannot be referenced.	int
PJCMD_JOBINFO_DATA	Equivalent to the specification of the --data option in the pjstat command 0: Not specified (Default) 1: Specified This parameter does not affect the job information to be obtained.	int
PJCMD_JOBINFO_DELIMITER	Equivalent to the specification of the --delimiter option in the pjstat command If the PJCMD_JOBINFO_DATA parameter is specified, this parameter specifies a character for separating job information displayed by the pjcmd_jobinfo_print_resp() function. If this parameter is not set, a comma (,) is used as the character separating job information. This parameter does not affect the job information to be obtained.	char *
PJCMD_JOBINFO_HELP	Equivalent to the specification of the --help option in the pjstat command 0: Not specified (Default) 1: Specified	int

param	**val_p*	Type of **val_p*
	This parameter does not affect the job information to be obtained.	

As for the PJCMD_JOBINFO_USER_ID or PJCMD_JOBINFO_USER_NAME parameter and the PJCMD_JOBINFO_GROUP_ID or PJCMD_JOBINFO_GROUP_NAME parameter, the parameter specified last in the respective pairs is valid. If it is not specified, the user or group that calls the function is applied.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job information acquisition.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown or unspecifiable value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

C.2.7 pjcmd_jobinfo_get_param()

```
pjcmd_result_t pjcmd_jobinfo_get_param(const PjcmdHandle_t *handle_p, pjcmd_jobinfo_param_t param,
void *val_p)
```

This function references information acquisition-related parameters that are set in a handle for job information acquisition.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_jobinfo_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job information acquisition.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

C.2.8 pjcmd_jobinfo_execute()

```
PjcmdResp_t *pjcmd_jobinfo_execute(PjcmdHandle_t *handle_p)
```

This function requests the job operation management function to get job information based on a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about acquiring job information.
The caller must release the obtained response information by using pjcmd_destroy_resp(). If a request to get job information has failed, NULL is returned and the cause is set in pjcmd_errcode.
The response information indicates whether a request has succeeded or failed. Whether or not job information has been obtained successfully needs to be checked with a result code in the response information by using the pjcmd_get_result() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for job information acquisition.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node.
The function can be called from the login node, compute cluster management node and system management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_BUSY

An operation cannot be requested because another operation request function is being processed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

PJCMD_ERROR_INTERNAL

Internal error

C.2.9 pjcmd_jobinfo_get_choosen_item()

```
pjcmd_result_t pjcmd_jobinfo_get_choosen_item(const PjcmdResp_t *resp_p, char ***jobinfo_item_ppp,
int *jobinfo_item_n_p, char ***nodeinfo_item_ppp, int *nodeinfo_item_n_p)
```

This function gets the information item name list included in job information acquisition results.

[ARGUMENTS]

resp_p

Pointer to a response information

jobinfo_item_ppp

The list of job information item names is stored in the character string array (**jobinfo_item_ppp*)[]. For details on the item names to be stored, see "Table C.2 Item Names, Names, and Values of Job Information (1)."

jobinfo_item_n_p

The number of elements in the (**jobinfo_item_ppp*)[] array, in which the list of job information item names is stored, is stored in **jobinfo_item_n_p*.

nodeinfo_item_ppp

The list of job information item names for each node is stored in the character string array (**nodeinfo_item_ppp*)[]. For details on the item names to be stored, see "Table C.3 Item Names, Names, and Values of Job Information (2)."

nodeinfo_item_n_p

The number of elements in the array (**nodeinfo_item_ppp*)[], in which the list of job information item names for each node is stored, is stored in *node*info_item_n_p*.

The areas that are specified by *jobinfo_item_ppp* and *nodeinfo_item_ppp* are reserved areas in response information. The caller must not directly release them. These areas are reserved until response information is released.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring job information.

PJCMD_ERROR_INVALID_ARGUMENT

jobinfo_item_ppp, *nodeinfo_item_ppp*, or *nodeinfo_item_n_p* is invalid (NULL).

C.2.10 pjcmd_jobinfo_read_infogrp()

```
pjcmd_result_t pjcmd_jobinfo_read_infogrp(PjcmdResp_t *resp_p)
```

This function moves a reference position from the information group that is an acquisition unit of information stored in response information to the next information group.

[ARGUMENTS]

resp_p

Pointer to a response information

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring job information.

PJCMD_ERROR_NODATA

There is no next information group (all information was referenced).

PJCMD_ERROR_INVALID_PARAM

A parameter in response information is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

C.2.11 pjcmd_jobinfo_print_resp()

```
pjcmd_result_t pjcmd_jobinfo_print_resp(PjcmdResp_t *resp_p, pjcmd_jobinfo_print_type_t type)
```

This function outputs job information included in the reference target information group to the standard output based on the specification in the pjstat command.

If the following operation is performed for the target information group, an error occurs.

- This function is called twice or more.
- The information group is referenced with the pjcmd_jobinfo_read_jobinfo() function before calling this function.

The job information in the target information group is referenced at the end after calling this function. Therefore, if the pjcmd_jobinfo_read_jobinfo() function is called for the same information group after calling this function, job information cannot be obtained.

[ARGUMENTS]

resp_p

Pointer to a response information

type

Output detail type

PJCMD_JOBINFO_PRINT_SUMMARY

Job summary information is displayed.

PJCMD_JOBINFO_PRINT_JOBINFO

Detailed job information is displayed.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring job information.
- This is not response information that was successfully obtained.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown or unspecifiable value is specified in *type.*

PJCMD_ERROR_INVALID_PARAM

Information could not be obtained because the information acquisition range and the conditions specified in the pjcmd_jobinfo_put_scope() function and pjcmd_jobinfo_put_param() function are invalid.
Alternatively the job information in an information group is referenced by the pjcmd_jobinfo_read_jobinfo() function before calling this function.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

C.2.12 pjcmd_jobinfo_get_summary()

```
pjcmd_result_t pjcmd_jobinfo_get_summary(const PjcmdResp_t *resp_p, pjcmd_job_status_t status,
int64_t *num_p)
```

This function references the number of jobs and the number of sub jobs that are in a specific state out of the jobs included in an information group.
If PJCMD_JOBINFO_VERBOSITY_SUBJOB is not specified as the granularity of information (PJCMD_JOBINFO_VERBOSITY) in the pjcmd_jobinfo_put_param() function, the number of sub jobs is 0. Alternatively, if PJCMD_JOBINFO_HISTORY is not specified in the pjcmd_jobinfo_put_param() function, the number of completed jobs cannot be obtained. In this case, an error occurs.

[ARGUMENTS]

resp_p

Pointer to a response information

status

Identifier indicating the status of jobs to be referenced (See the following table.)

status	Description
PJCMD_JOB_STATUS_ACCEPT	Job in the submission accept state (ACCEPT state)
PJCMD_JOB_STATUS_QUEUED	Job waiting to be executed (QUEUED state)
PJCMD_JOB_STATUS_RUNNING_A	Job whose execution is being requested (RUNNING-A state)
PJCMD_JOB_STATUS_RUNNING_P	Job in the prologue process (RUNNING-P state)
PJCMD_JOB_STATUS_RUNNING	Job that is being executed (RUNNING state)
PJCMD_JOB_STATUS_RUNNING_E	Job in the epilogue process (RUNNING-E state)
PJCMD_JOB_STATUS_RUNOUT	Job waiting for the completion process to complete (RUNOUT state)
PJCMD_JOB_STATUS_HOLD	Job that is held by a user (HOLD state)
PJCMD_JOB_STATUS_ERROR	Job that is held due to an error (ERROR state)
PJCMD_JOB_STATUS_SUSPEND	Job that is being suspended (SUSPEND state)
PJCMD_JOB_STATUS_SUSPENDED	Suspended job (SUSPENDED state)
PJCMD_JOB_STATUS_RESUME	Job that is being resumed (RESUME state)
PJCMD_JOB_STATUS_REJECT	Job whose submission was not accepted (REJECT state)
PJCMD_JOB_STATUS_EXIT	Completed job (EXIT state)
PJCMD_JOB_STATUS_CANCEL	Job whose execution was canceled (CANCEL state)
PJCMD_JOB_STATUS_ALL	All jobs that have not completed (ACCEPT, QUEUED, RUNNING_A, RUNNING_P, RUNNING, RUNNING_E, RUNOUT, HOLD, ERROR, SUSPEND, SUSPENDED, and RESUME states)
PJCMD_JOB_STATUS_END	Completed jobs (REJECT, EXIT, and CANCEL states)
PJCMD_JOB_STATUS_TOTAL	All jobs (ACCEPT, QUEUED, RUNNING_A, RUNNING_P, RUNNING, RUNNING_E, RUNOUT, HOLD, ERROR, SUSPEND, SUSPENDED, RESUME, REJECT, EXIT, and CANCEL states)

num_p

The number of jobs and number of sub jobs are stored in the *num_p*[] array. The caller needs to prepare an array area where two int64_t type values can be stored.

[RETURN VALUE]

PJCMD_OK

Success. The number of jobs is stored in the *num_p*[0] argument and the number of sub jobs are stored in the *num_p*[1] argument.

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring job information.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_ARGUMENT

num_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown or unspecifiable value is specified in *status.*

PJCMD_ERROR_NODATA

There was an attempt to obtain the number of completed jobs when the completed job information was not included in response information.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

PJCMD_ERROR_INVALID_PARAM

Information could not be obtained because the information acquisition range and the conditions specified in the pjcmd_jobinfo_put_scope() function and pjcmd_jobinfo_put_param() function are invalid.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

C.2.13 pjcmd_jobinfo_get_infogrp_scope()

```
pjcmd_result_t pjcmd_jobinfo_get_infogrp_scope(const PjcmdResp_t *resp_p, char **rscunit_pp, char
**rscgrp_pp, char **uname_pp, char **gname_pp)
```

This function gets the resource unit name, resource group name, user name, and group name of the information group that is being referenced now.

[ARGUMENTS]

resp_p

Pointer to a response information

rscunit_pp

The pointer to a resource unit name is stored in **rscunit_pp.*

rscgrp_pp

The pointer to a resource group name is stored in **rscgrp_pp.*

uname_pp

The pointer to a user name is stored in **uname_pp*.

gname_pp

The pointer to a group name is stored in **gname_pp*.

The areas specified by *rscunit_pp, rscgrp_pp, uname_pp, and gname_pp* are reserved areas in response information. The caller must not directly release them. These areas are reserved until the response information is released or the next information group is read.

[RETURN VALUE]

PJCMD_OK

Success. A value that can be obtained is information in the acquisition target unit (resource unit, resource group, user, or group) specified in the pjcmd_jobinfo_put_scope() function and the pjcmd_jobinfo_put_condition() function. Therefore, the names of unspecified targets cannot be obtained. In this case, (char *) NULL is stored.

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring job information.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_ARGUMENT

rscunit_pp, *rscgrp_pp*, *uname_pp*, or *gname_pp* is invalid (NULL).

C.2.14 pjcmd_jobinfo_read_jobinfo()

```
PjcmdJobinfo_t *pjcmd_jobinfo_read_jobinfo(PjcmdResp_t *resp_p)
```

This function gets the information on the next job that is included in an information group that is being referenced.
However, if the granularity of information (PJCMD_JOBINFO_VERBOSITY) that is specified in the pjcmd_jobinfo_put_param() function is limited to summary information (PJCMD_JOBINFO_VERBOSITY_SUMMARY), the information on the next job cannot be obtained.

[ARGUMENTS]

resp_p

Pointer to a response information

[RETURN VALUE]

Job information.
If the function fails, NULL is returned and the cause is set in pjcmd_errcode. The contents of the job information to be returned are undetermined after the next job information is read.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring job information.
- This is not response information that was successfully obtained.

PJCMD_ERROR_NODATA

The next job information is not included in the current information group. Alternatively, only summary information is obtained. Therefore, there is no job information.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

PJCMD_ERROR_INVALID_PARAM

Information could not be obtained because the information acquisition range and the conditions specified in the pjcmd_jobinfo_put_scope() function and pjcmd_jobinfo_put_param() function are invalid.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

C.2.15 pjcmd_jobinfo_get_jobinfo_item_num()

```
int pjcmd_jobinfo_get_jobinfo_item_num(const PjcmdJobinfo_t *jobinfo_p)
```

This function returns the number of information items included in job information.

[ARGUMENTS]

jobinfo_p

Pointer to a job information

[RETURN VALUE]

Number of information items included in job information.
This value indicates the total number of items that can be obtained as job information. Items that are not obtained are also included.
If the function fails, -1 is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

jobinfo_p is invalid (NULL).

C.2.16 pjcmd_jobinfo_get_jobinfo_item_value()

```
pjcmd_result_t pjcmd_jobinfo_get_jobinfo_item_value(PjcmdJobinfo_t *jobinfo_p, int indx, char
**name_pp, char **val_pp)
```

This function gets the name and value of each information item contained in job information.

[ARGUMENTS]

jobinfo_p

Pointer to a job information

indx

Index of information that needs to be obtained. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the pjcmd_jobinfo_get_jobinfo_item_num() function.

name_pp

The pointer to the name of information specified in *Indx* is stored in **name_pp*. For details on the names to be stored, see "Table C.2 Item Names, Names, and Values of Job Information (1)."

val_pp

The pointer to an information value (character string) specified by *Indx* is stored in **val_pp*. For details on the values to be stored, see "Table C.2 Item Names, Names, and Values of Job Information (1)."

The areas specified by *name_pp* and *val_pp* are reserved areas in response information. The caller must not directly release them. The area specified by *name_pp* is retained until response information is released. The area specified by *val_pp* is retained until response information is released or until the next time this function is called.

[RETURN VALUE]

PJCMD_OK

Success. If information about specified items is not obtained, NULL is stored in (char *)**val_pp*.
The PJCMD_JOBINFO_OTHERSJOB parameter (job information for other users is obtained) is set in PJCMD_JOBINFO_OTHERSJOB_ALL (all job information is obtained) when job information is obtained, and the job information for all users can be obtained. However, the character string that is specified by (char *) **val_pp* becomes "?" for information without the reference privilege.

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

jobinfo_p, *name_pp*, or *val_pp* is invalid (NULL).

PJCMD_ERROR_NODATA

The value of *indx* is out of range.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

C.2.17 pjcmd_jobinfo_get_jobinfo_node_num()

```
pjcmd_result_t pjcmd_jobinfo_get_jobinfo_node_num(const PjcmdJobinfo_t *jobinfo_p, uint32_t *num_p)
```

This function obtains the amount of information in a node unit (number of nodes) that is included in job information.

[ARGUMENTS]

jobinfo_p

Pointer to a job information

num_p

The amount of information (number of nodes) in a node unit is stored in **num_p.*
If the target jobs are step jobs, summary jobs of bulk jobs, or jobs before execution, there is no node information. In this case, 0 is stored in **num_p.*

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

Job information is invalid. *Jobinfo_p* is NULL, or the level of job information acquisition is not PJCMD_JOBINFO_LEVEL_3.

C.2.18 pjcmd_jobinfo_get_nodejobinfo_item_num()

```
int pjcmd_jobinfo_get_nodejobinfo_item_num(const PjcmdJobinfo_t *jobinfo_p)
```

This function returns the number of information items included in job information in a node unit.

[ARGUMENTS]

jobinfo_p

Pointer to a job information

[RETURN VALUE]

Number of information items included in job information in a node unit.
This value indicates the total number of items that can be obtained as job information. Items that are not obtained are also included.
However, if job information in a node unit is not obtained, an error occurs.
If the function fails, -1 is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

jobinfo_p is invalid (NULL).

PJCMD_ERROR_NODATA

There is no job information in a node unit in job information.

C.2.19 pjcmd_jobinfo_get_nodejobinfo_item_value()

```
pjcmd_result_t pjcmd_jobinfo_get_nodejobinfo_item_value(PjcmdJobinfo_t *jobinfo_p, uint32_t
node_indx, int item_indx, char **name_pp, char **val_pp)
```

This function obtains the name and value of each information item contained in job information in a node unit.

[ARGUMENTS]

jobinfo_p

Pointer to a job information

node_indx

Node index in job information. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that can be obtained by the pjcmd_jobinfo_get_jobinfo_node_num() function.

item_indx

Index of information that needs to be obtained. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the pjcmd_jobinfo_get_nodejobinfo_item_num() function.

name_pp

The pointer to the name of information specified by *item_indx* is stored in **name_pp*. For details on the names to be stored, see "Table C.3 Item Names, Names, and Values of Job Information (2)."

val_pp

The pointer to an information value (character string) specified by *item_indx* is stored in **val_pp*. For details on the values to be stored, see "Table C.3 Item Names, Names, and Values of Job Information (2)."

The areas specified by *name_pp* and *val_pp* are reserved areas in response information. The caller must not directly release them. The area specified by *name_pp* is retained until response information is released. The area specified by *val_pp* is retained until response information is released or until the next time this function is called.

[RETURN VALUE]

PJCMD_OK

Success. If information is not obtained, NULL is stored in ***val_pp*.

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

nodejobinfo_p, *name_pp*, or *val_pp* is invalid (NULL).

PJCMD_ERROR_NODATA

The *node_indx* or *item_indx* value is out of the specified range.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

C.2.20 Item Names, Names, and Values of Job Information

"Table C.2 Item Names, Names, and Values of Job Information (1)" lists information obtained by the following functions.

- Item names of information that are stored in the *jobinfo_item_ppp* argument of the pjcmd_jobinfo_get_choosen_item() function
- Information names that are stored in the *name_pp* argument of the pjcmd_jobinfo_get_jobinfo_item_value() function
- Values of information that are stored in the *val_pp* argument of the pjcmd_jobinfo_get_jobinfo_item_value() function

Table C.2 Item Names, Names, and Values of Job Information (1)

Item (*jobinfo_item_ppp*)	Name (*name_pp*)	Value (*val_pp*)
jid	JOB ID	Job ID or sub job ID The item has the same format as the job statistical information JOB ID displayed by the pjstat command (see pjstatsinfo(7).)
snum	SUB JOB NUM	Number of sub jobs of a bulk job or step job The format is the same format as the job statistical information SUB JOB NUM displayed by the pjstat command (see pjstatsinfo(7).)
nnumr	NODE NUM (REQUIRE)	Number of nodes and shape that are requested by a job The format is the same format as the job statistical information NODE NUM (REQUIRE) displayed by the pjstat command (see pjstatsinfo(7).)
nnuma	NODE NUM (ALLOC)	Number of nodes and shape that are assigned to a job The format is the same format as the job statistical information NODE NUM (ALLOC) displayed by the pjstat command (see pjstatsinfo(7).)
Other than above	Same as the items that are output by the pjstat command, among the job statistical information provided in apjstatsinfo(7) Example: If an item name is "jnam", the name is "JOB NAME" and the value is the job name.	

"Table C.3 Item Names, Names, and Values of Job Information (2)" lists information obtained by the following functions.

- Item names of information that are stored in the *nodeinfo_item_ppp* argument of the pjcmd_jobinfo_get_choosen_item() function
- Information names that are stored in the *name_pp* argument of the pjcmd_jobinfo_get_nodejobinfo_item_value() function

- Values of information that are stored in the *val_pp* argument of the pjcmd_jobinfo_get_nodejobinfo_item_value() function

Table C.3 Item Names, Names, and Values of Job Information (2)

Item (*nodeinfo_item_ppp*)	Name (*name_pp*)	Value (*val_pp*)
ntofu	TOFU COORDINATE	Tofu coordinates of a node The item has the same format as the node/virtual node statistical information TOFU COORDINATE displayed by the pjstat command (see pjstatsinfo(7).)
node	NODE COORDINATE	Coordinates of a node to be used The item has the same format as the node/virtual node statistical information NODE COORDINATE displayed by the pjstat command (see pjstatsinfo(7).)
Other than above	Same as the items that are output by the pjstat command, among the node/virtual node statistical information provided in pjstatsinfo(7) Example: If an item name is "vnid", the name is "VNODE ID" and the value is the virtual node ID.	

C.3 Getting Resource Information for Jobs

This section describes the functions for getting information on the system resources used to execute jobs.

Figure C.5 Requesting Resource Information Acquisition

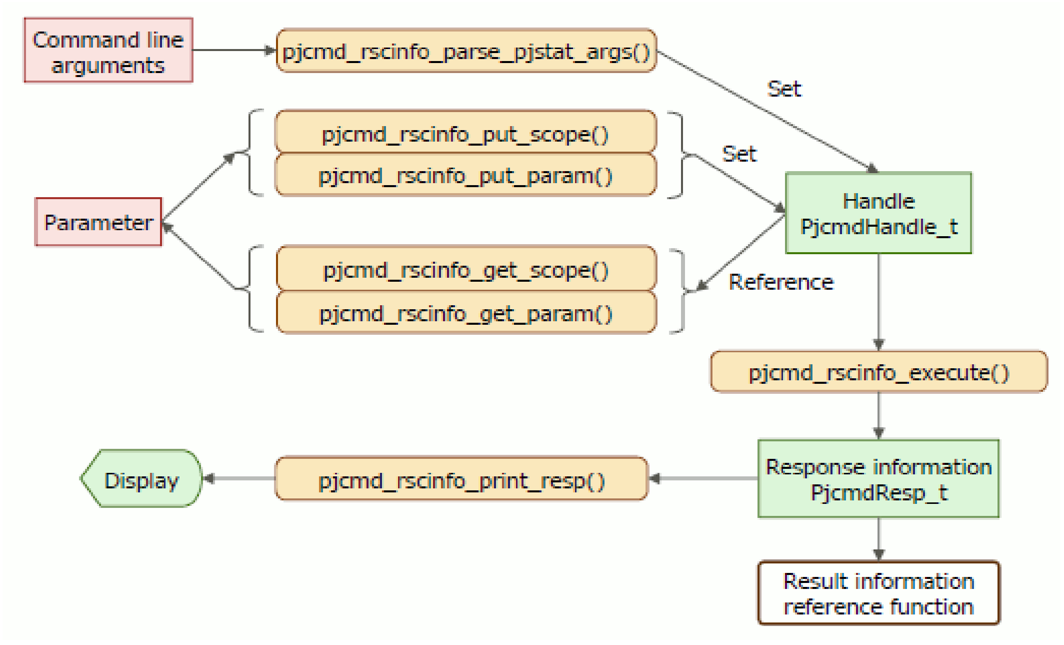
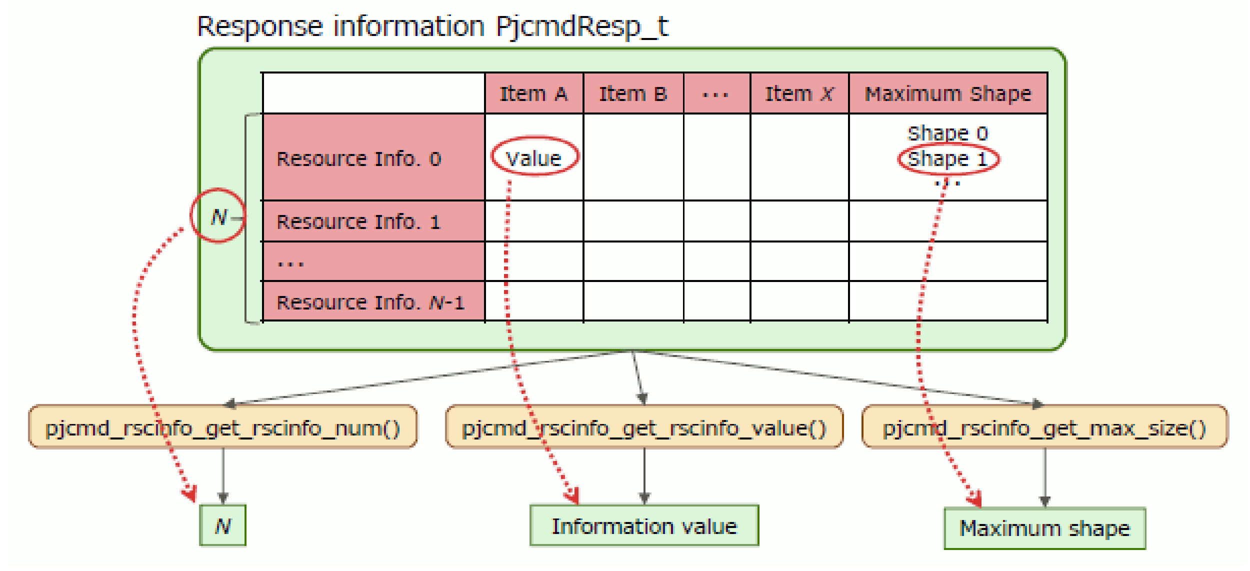

Command line arguments
pjcmd_rscinfo_parse_pjstat_args()
Set
pjcmd_rscinfo_put_scope()
pjcmd_rscinfo_put_param()
Set
Parameter
Handle PjcmdHandle_t
pjcmd_rscinfo_get_scope()
pjcmd_rscinfo_get_param()
Reference
pjcmd_rscinfo_execute()
Display
pjcmd_rscinfo_print_resp()
Response information PjcmdResp_t
Result information reference function

Figure C.6 Referencing Resource Information



C.3.1 pjcmd_rscinfo_parse_pjstat_args()

```
pjcmd_result_t pjcmd_rscinfo_parse_pjstat_args(PjcmdHandle_t *handle_p, int argc, char **argv_pp)
```

This function analyzes command line arguments based on the specification used when specifying the --rsc option in the pjstat command and sets them in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for resource information acquisition.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option is not specified.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

Calling this function moves arguments other than options to the end of the *argv_pp*[] array.
If an unrecognizable option is detected, analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.
This function recognizes only the --rsc option in the pjstat command and other options that can be specified with the --rsc option.

C.3.2 pjcmd_rscinfo_put_scope()

```
pjcmd_result_t pjcmd_rscinfo_put_scope(PjcmdHandle_t *handle_p,pjcmd_scope_t scope, void *val_p,
uint32_t n)
```

This function sets the target range for getting resource information.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier indicating the type of target range for getting information (See the following table.)

val_p

Pointer to the storage area for the value indicating the target to get information. For example, if the value type to be set is char * type, the caller must prepare a storage area for the char * type value and specify a pointer (char **) to the area in *val_p*. If NULL is specified, the parameter value is initialized (not set).

n

Number of elements of *val_p*

scope	**val_p*	Type of **val_p*
PJCMD_SCOPE_CLUSTER	Cluster name (only one name, equivalent to pjstat -c) If this parameter is not set, the PXMYCLST environment variable value is applied. If the parameter and the environment variable are not set, an error occurs in the pjcmd_rscinfo_execute() function. This parameter is valid only when called from the system management node. When the parameter is called from a node other than the system management node, the cluster to which the calling node belongs is applied. 1 must be specified in *n* for the parameter.	char *
PJCMD_SCOPE_RSCUNIT	Array of a resource unit name (number of elements is *n*, equivalent to --rscunit of the pjstat command) If this parameter is not set, all the resource units in a cluster are the target.	char **
PJCMD_SCOPE_RSCGRP	Array of a resource group name (number of elements is *n*, equivalent to the --rscgrp of the pjstat command).	char **

scope	**val_p*	Type of **val_p*
	If this parameter is not set, all the resource units in a cluster are the target.	

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for resource information acquisition.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown or unspecifiable value is specified in *scope.*

PJCMD_ERROR_INVALID_PARAM

val_p or *n* is invalid.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

C.3.3 pjcmd_rscinfo_get_scope()

```
pjcmd_result_t pjcmd_rscinfo_get_scope(const PjcmdHandle_t *handle_p, pjcmd_scope_t scope, void
*val_p, uint32_t *n_p)
```

This function references the resource information acquisition range that is set in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier of the target range to be referenced. The identifiers that can be specified are the same as those for the pjcmd_rscinfo_put_scope() function.

val_p

A value is stored in **val_p* based on the *scope* type. The caller needs to prepare an area of a sufficient size according to the value type.

n_p

The number of elements of *val_p* is stored in **n_p*. The caller needs to prepare the area.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for resource information acquisition.

PJCMD_ERROR_INVALID_ARGUMENT

val_p or *n* is invalid.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *scope.*

PJCMD_ERROR_NODATA

A specified *scope* is not set in a handle.

C.3.4 pjcmd_rscinfo_put_param()

```
pjcmd_result_t pjcmd_rscinfo_put_param(PjcmdHandle_t *handle_p, pjcmd_rscinfo_param_t param, const
void *val_p)
```

This function sets parameters in a handle that are related to resource information acquisition.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifiers of the parameters to be set that are related to resource information acquisition (See the following table.)

val_p

Pointer to the storage area for the value of parameter to be set. For example, if the value type to be set is int type, the caller must prepare a storage area for the int type value and specify a pointer (int *) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_RSCINFO_SHAPE	Equivalent to the specification of the --shape option in the pjstat command 0: Do not display maximum shape information for resource group (Default) 1: Display maximum shape information for resource group This parameter does not affect the contents of the information to be obtained but does affect the contents displayed by the pjcmd_rscinfo_print_resp() function.	int
PJCMD_RSCINFO_HELP	Equivalent to the specification of the --help option in the pjstat command 0: Not specified (Default) 1: Specified This parameter does not affect the job information to be obtained.	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for resource information acquisition.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

C.3.5 pjcmd_rscinfo_get_param()

```
pjcmd_result_t pjcmd_rscinfo_get_param(const PjcmdHandle_t *handle_p, pjcmd_rscinfo_param_t param,
void *val_p)
```

This function references information acquisition-related parameters that are set in a handle for resource information acquisition.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of the parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_rscinfo_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for resource information acquisition.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param*.

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

C.3.6 pjcmd_rscinfo_execute()

```
PjcmdResp_t* pjcmd_rscinfo_execute(const PjcmdHandle_t *handle_p)
```

This function requests the job operation management function to get resource information based on a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about getting resource information for a resource unit/resource group.
The caller must release the obtained response information by using pjcmd_destroy_resp(). If a request to get job information has failed, NULL is returned and the cause is set in pjcmd_errcode.
The response information indicates whether a request has succeeded or failed. Whether or not job information has been obtained successfully needs to be checked with a result code in the response information by using the pjcmd_get_result() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for resource information acquisition.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node.
The function can be called from the login node, compute cluster management node and system management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_BUSY

An operation cannot be requested because another operation request function is being processed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

PJCMD_ERROR_INTERNAL

Internal error

C.3.7 pjcmd_rscinfo_print_resp()

```
pjcmd_result_t pjcmd_rscinfo_print_resp(const PjcmdResp_t *resp_p)
```

This function outputs resource information acquisition results to the standard output based on the specification used when specifying the --rsc option in the pjstat command.

[ARGUMENTS]

resp_p

Pointer to a response information

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring resource information.
- This is not response information that was successfully obtained.

C.3.8 pjcmd_rscinfo_get_rscinfo_num()

```
int pjcmd_rscinfo_get_rscinfo_num(const PjcmdResp_t *resp_p)
```

This function gets the amount of resource information contained in response information about acquiring resource information.

[ARGUMENTS]

resp_p

Pointer to a response information

[RETURN VALUE]

Amount of resource information.
If the function fails, -1 is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring resource information.
- This is not response information that was successfully obtained.

C.3.9 pjcmd_rscinfo_get_rscinfo_value()

```
pjcmd_result_t pjcmd_rscinfo_get_rscinfo_value(const PjcmdResp_t *resp_p, int indx,
pjcmd_rscinfo_item_t item, void *val_p)
```

This function gets values related to specific resource information from multiple pieces of resource information included in response information about acquiring resource information.

[ARGUMENTS]

resp_p

Pointer to a response information

indx

Index of resource information that needs to be referenced in response information. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is returned by the pjcmd_rscinfo_get_rscinfo_num() function.

item

Identifier of an item that needs to be referenced in resource information (See the following table.)

val_p

A value is stored in **val_p* based on the *item* type. The caller needs to prepare an area of a sufficient size according to the value type.

item	**val_p*	Type of **val_p*
PJCMD_RSCINFO_ITEM_RU_NAME	Resource unit name The area specified by **val_p* is undetermined after the release of response information.	char *
PJCMD_RSCINFO_ITEM_RU_JOB_SUBMIT	Whether or not jobs can be submitted to a resource unit 0: New jobs cannot be submitted. 1: New jobs can be submitted.	int
PJCMD_RSCINFO_ITEM_RU_JOB_EXECUTE	Whether or not jobs can be executed in a resource unit 0: New jobs cannot be executed. 1: New jobs can be executed.	int
PJCMD_RSCINFO_ITEM_RU_SIZE	Resource unit size - If compute node is FX server: Tofu shape (unsigned int)*val_p*[0]: Size in X direction (unsigned int)*val_p*[1]: Size in Y direction (unsigned int)*val_p*[2]: Size in Z direction - If compute node is PRIMERGY server: Number of nodes (unsigned int)*val_p*[0]: Number of nodes (unsigned int)*val_p*[1]: Not used (unsigned int)*val_p*[2]: Not used	Array of unsigned int (Three elements) (*)
PJCMD_RSCINFO_ITEM_RG_NAME	Resource group name The area specified by **val_p* is undetermined after the release of response information.	char *
PJCMD_RSCINFO_ITEM_RG_JOB_SUBMIT	Whether or not jobs can be submitted to a resource group 0: New jobs cannot be submitted. 1: New jobs can be submitted.	int
PJCMD_RSCINFO_ITEM_RG_JOB_EXECUTE	Whether or not jobs can be executed in a resource group 0: New jobs cannot be executed. 1: New jobs can be executed.	int

item	**val_p*	Type of **val_p*
PJCMD_RSCINFO_ITEM_RG_SIZE	Resource group size - If compute node is FX server: Node shape (unsigned int)*val_p*[0]: Size in X direction (unsigned int)*val_p*[1]: Size in Y direction (unsigned int)*val_p*[2]: Size in Z direction - If compute node is PRIMERGY server: Number of nodes (unsigned int)*val_p*[0]: Number of nodes (unsigned int)*val_p*[1]: Not used (unsigned int)*val_p*[2]: Not used	Array of unsigned int (Three elements) (*)
PJCMD_RSCINFO_ITEM_MAX_SIZE_NUM	Number of variations of the maximum node shape that can be assigned to a job This is used to get the maximum node shape by using the pjcmd_rscinfo_get_max_size() function.	unsigned int

(*) If the compute node is the PRIMERGY server, only the first element in an array is used. However, the areas must be prepared for three elements.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring resource information.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *item.*

PJCMD_ERROR_NODATA

The value of *indx* is out of range.

C.3.10 pjcmd_rscinfo_get_max_size()

```
pjcmd_result_t pjcmd_rscinfo_get_max_size(PjcmdResp_t *resp_p, int indx1, int indx2, unsigned int
*val_p)
```

This function gets the maximum node shape that can be assigned to a job for one piece of resource information contained in response information.

[ARGUMENTS]

resp_p

Pointer to a response information

indx1

Index of resource information, which needs to be referenced, in response information. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the pjcmd_rscinfo_get_rscinfo_num() function.

indx2

Index that indicates one of the variations of the maximum node shape in resource information. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the parameter PJCMD_RSCINFO_ITEM_MAX_SIZE_NUM of the pjcmd_rscinfo_get_info() function.

val_p

The maximum node shape is stored in the *val_p*[] array. The caller needs to reserve the storage area for an unsigned int-type array (number of elements is 3).

- If the compute node is the FX server: Node shape
  val_p[0]: Size in X direction
  val_p[1]: Size in Y direction
  val_p[2]: Size in Z direction

- If the compute node is the PRIMERGY server: Number of nodes
  val_p[0]: Number of nodes
  val_p[1]: Not used
  Val_p[2]: Not used

If the compute node is the PRIMERGY server, only the first element in an array is used. However, the areas must be prepared for three elements.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring resource information.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *item.*

PJCMD_ERROR_NODATA

The value of *indx1* or *indx2* is out of range.

C.4 Getting Limit Value Information When Submitting a Job

This section describes the functions for getting information on limit values when submitting a job.

Figure C.7 Getting Limit Value Information

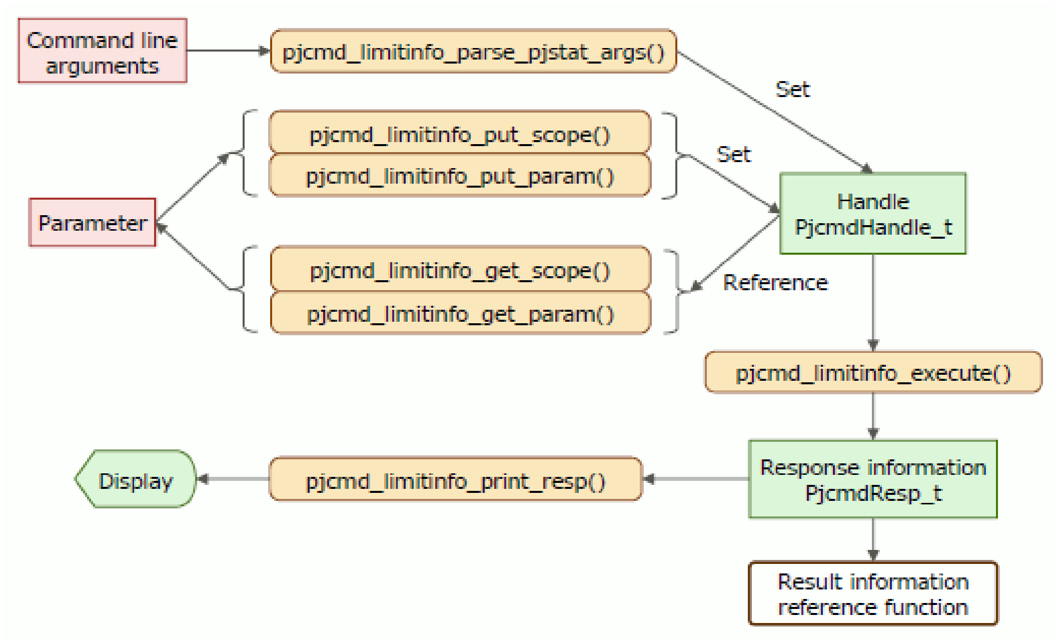
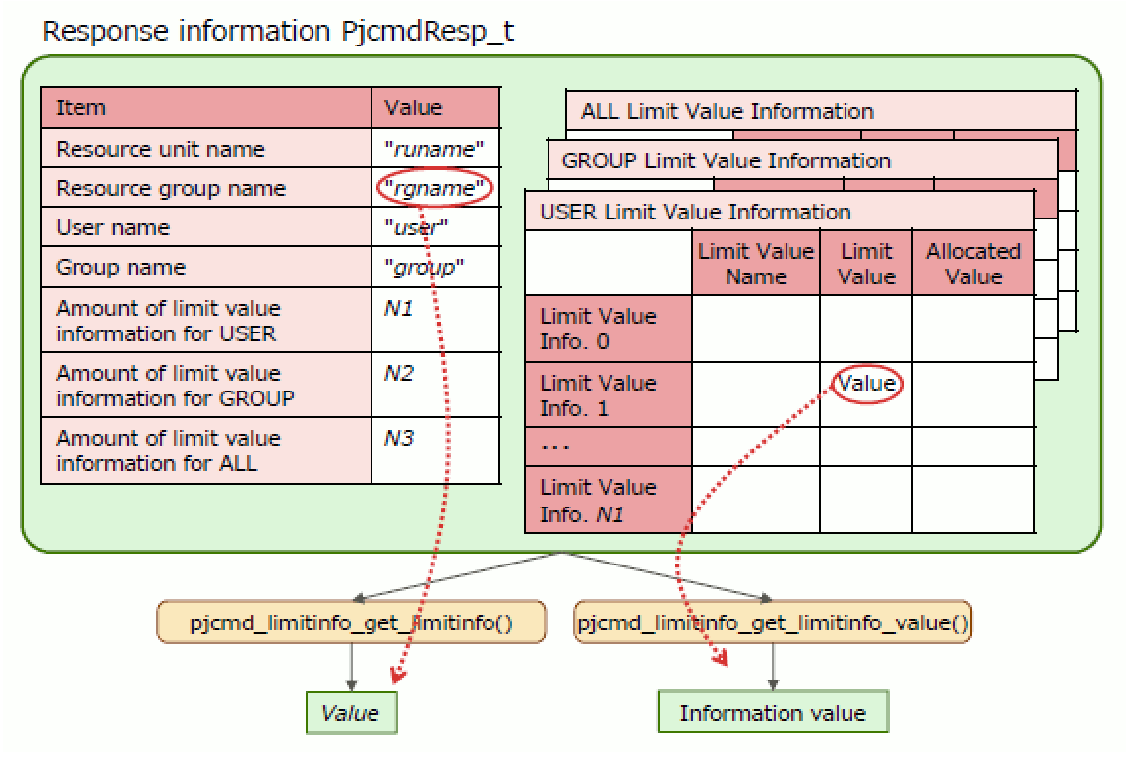

Command line arguments
pjcmd_limitinfo_parse_pjstat_args()
Set
pjcmd_limitinfo_put_scope()
pjcmd_limitinfo_put_param()
Set
Parameter
Handle PjcmdHandle_t
pjcmd_limitinfo_get_scope()
pjcmd_limitinfo_get_param()
Reference
pjcmd_limitinfo_execute()
Display
pjcmd_limitinfo_print_resp()
Response information PjcmdResp_t
Result information reference function

Figure C.8 Referencing Limit Value Information


Response information PjcmdResp_t
ALL Limit Value Information
GROUP Limit Value Information
USER Limit Value Information
Limit Value Name
Limit Value
Allocated Value
Limit Value Info. 0
Limit Value Info. 1
...
Limit Value Info. N1
Value
pjcmd_limitinfo_get_limitinfo()
pjcmd_limitinfo_get_limitinfo_value()
Value
Information value

Item	Value
Resource unit name	"*runame*"
Resource group name	"*rgname*"
User name	"*user*"
Group name	"*group*"
Amount of limit value information for USER	*N1*
Amount of limit value information for GROUP	*N2*
Amount of limit value information for ALL	*N3*

C.4.1 pjcmd_limitinfo_parse_pjstat_args()

```
pjcmd_result_t pjcmd_limitinfo_parse_pjstat_args(PjcmdHandle_t *handle_p, int argc, char **argv_pp)
```

This function analyzes command line arguments based on the specification used when specifying the --limit option in the pjstat command and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for acquiring limit value information.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option is not specified.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

Calling this function moves arguments other than options to the end of the *argv_pp*[] array.
If an unrecognizable option is detected, analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.
This function recognizes only the --limit option in the pjstat command and other options that can be specified with the --limit option at the same time.

C.4.2 pjcmd_limitinfo_put_scope()

```
pjcmd_result_t pjcmd_limitinfo_put_scope(PjcmdHandle_t *handle_p, pjcmd_scope_t scope, void *val_p)
```

This function sets the target range for getting the limit value information in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier indicating the type of target range for getting information (See the following table.)

val_p

Pointer to the storage area for the value indicating the target to get information. For example, if the value type to be set is char * type, the caller must prepare a storage area for the char * type value and specify a pointer (char **) to the area in *val_p*. If NULL is specified, the parameter value is initialized (not set).

scope	**val_p*	Type of **val_p*
PJCMD_SCOPE_CLUSTER	Cluster name (only one name, equivalent to pjstat -c) If this parameter is not set, the PXMYCLST environment variable value is applied. If the parameter and the environment variable are not set, an error occurs in the pjcmd_limitinfo_execute() function. This parameter is valid only when called from the system management node. When the parameter is called from a node other than the system management node, the cluster to which the calling node belongs is applied.	char *
PJCMD_SCOPE_RSCUNIT	Resource unit name (only one name, equivalent to pjstat --rscunit) If this parameter is not set, the default resource unit of the execution user is applied.	char *
PJCMD_SCOPE_RSCGRP	Resource group name (only one name, equivalent to pjstat --rscgrp) If this parameter is not set, resource unit information is obtained.	char *

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for acquiring limit value information.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown or unspecifiable value is specified in *scope*.

PJCMD_ERROR_INVALID_PARAM

val_p is invalid.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

C.4.3 pjcmd_limitinfo_get_scope()

```
pjcmd_result_t pjcmd_limitinfo_get_scope(const PjcmdHandle_t *handle_p, pjcmd_scope_t scope, void
*val_p)
```

This function references a range for getting the limit value information that is set in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier of the target range to get information to be referenced. The identifiers that can be specified are the same as those for the pjcmd_limitinfo_put_scope() function.

val_p

A value is stored in **val_p* based on the *scope* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for acquiring limit value information.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *scope.*

PJCMD_ERROR_NODATA

A specified *scope* is not set in a handle.

C.4.4 pjcmd_limitinfo_put_param()

```
pjcmd_result_t pjcmd_limitinfo_put_param(PjcmdHandle_t *handle_p, pjcmd_limitinfo_param_t param,
const void *val_p)
```

This function sets parameters in a handle that are related to limit value information acquisition.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of the parameters related to the limit value information to be obtained (See the following table.)

val_p

Pointer to the storage area for the value of parameter to be set. For example, if the value type to be set is uid_t type, the caller must prepare a storage area for the uid_t type value and specify a pointer (uid_t *) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_LIMITINFO_USER_ID	Target user ID for which limit value information is obtained	uid_t

param	**val_p*	Type of **val_p*
PJCMD_LIMITINFO_USER_NAME	Target user name for which limit value information is obtained	char *
PJCMD_LIMITINFO_GROUP_ID	Target group ID for which limit value information is obtained	gid_t
PJCMD_LIMITINFO_GROUP_NAME	Target group name for which limit value information is obtained	char *
PJCMD_LIMITINFO_HELP	Equivalent to the specification of the --help option in the pjstat command 0: Not specified (Default) 1: Specified This parameter does not affect the information to be obtained.	int

As for the PJCMD_LIMITINFO_USER_ID or PJCMD_LIMITINFO_USER_NAME parameter, and the PJCMD_LIMITINFO_GROUP_ID or PJCMD_LIMITINFO_GROUP_NAME parameter, the one specified last is valid, respectively. If it is not specified, the user or group that calls the function is applied.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for acquiring limit value information.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

C.4.5 pjcmd_limitinfo_get_param()

```
pjcmd_result_t pjcmd_limitinfo_get_param(PjcmdHandle_t *handle_p, pjcmd_limitinfo_param_t param,
void *val_p)
```

This function references information acquisition-related parameters that are set in a handle for acquisition of limit value information.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of the parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_limitinfo_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for acquiring limit value information.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

C.4.6 pjcmd_limitinfo_execute()

```
PjcmdResp_t *pjcmd_limitinfo_execute(const PjcmdHandle_t *handle_p)
```

This function requests the job operation management function to get limit value information based on a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about acquiring limit value information.
The caller must release the obtained response information by using pjcmd_destroy_resp(). If a request to get job information has failed, NULL is returned and the cause is set in pjcmd_errcode.
The response information indicates whether a request has succeeded or failed. Whether or not job information has been obtained successfully needs to be checked with a result code in the response information by using the pjcmd_get_result() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for acquiring limit value information.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node.
The function can be called from the login node, compute cluster management node and system management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_BUSY

An operation cannot be requested because another operation request function is being processed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

PJCMD_ERROR_INTERNAL

Internal error

C.4.7 pjcmd_limitinfo_print_resp()

```
pjcmd_result_t pjcmd_limitinfo_print_resp(const PjcmdResp_t *resp_p)
```

This function outputs restriction information acquisition results to the standard output based on the specification used when specifying the --limit option in the pjstat command.

[ARGUMENTS]

resp_p

Pointer to a response information

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring restriction information.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring restriction information.

- This is not response information that was successfully obtained.

C.4.8 pjcmd_limitinfo_get_limitinfo()

```
pjcmd_result_t pjcmd_limitinfo_get_limitinfo(const PjcmdResp_t *resp_p, pjcmd_limitinfo_item_t item,
void *val_p)
```

This function gets specific item information from the results of limit value information acquisition.

[ARGUMENTS]

resp_p

Pointer to a response information

item

Identifier indicating an information item to be obtained (See the following table.)

val_p

A value is stored in **val_p* based on the *item* type. The caller needs to prepare an area of a sufficient size according to the value type.

item	**val_p*	Type of **val_p*
PJCMD_LIMITINFO_ITEM_RSCUNIT	Resource unit name	char *
PJCMD_LIMITINFO_ITEM_RSCGRP	Resource group name	char *
PJCMD_LIMITINFO_ITEM_USER	Target user name	char *
PJCMD_LIMITINFO_ITEM_GROUP	Target group name	char *
PJCMD_LIMITINFO_ITEM_USER_INFO_NUM	Amount of limit value information for user	uint32_t
PJCMD_LIMITINFO_ITEM_GROUP_INFO_NUM	Amount of limit value information for group	uint32_t
PJCMD_LIMITINFO_ITEM_ALL_INFO_NUM	Amount of limit value information for total value of all users	uint32_t

The resource unit name, resource group name, target user name, and target group name are pointers to character strings that are stored in response information. Therefore, operation is undetermined when these character strings are referenced after the release of response information.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring restriction information.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *item.*

C.4.9 pjcmd_limitinfo_get_limitinfo_value()

```
pjcmd_result_t pjcmd_limitinfo_get_limitinfo_value(const PjcmdResp_t *resp_p,
pjcmd_limitinfo_class_t class, int indx, pjcmd_limitinfo_value_t item, void *val_p)
```

This function gets user restriction information, group restriction information, or specific information values in all restriction information from response information about acquiring limit value information.

[ARGUMENTS]

resp_p

Pointer to a response information

class

Identifier indicating the layer of limit value information to be referenced (user restriction information, group restriction information, or all restriction information)

class	Description
PJCMD_LIMITINFO_CLASS_USER	Limit value information for user
PJCMD_LIMITINFO_CLASS_GROUP	Limit value information for group
PJCMD_LIMITINFO_CLASS_ALL	Limit value information for total value of all users

indx

Index of reference information in the limit value information layer indicated by *class.* The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from the amount of limit value information in each layer that is obtained by the pjcmd_limitinfo_get_limitinfo() function.

item

Identifier indicating a limit value information item to be referenced (See the following table.)

val_p

A value is stored in **val_p* based on the *item* type. The caller needs to prepare an area of a sufficient size according to the value type.

item	**val_p*	Type of **val_p*
PJCMD_LIMITINFO_VALUE_NAME	Name of limit value	char *
PJCMD_LIMITINFO_VALUE_UPPER	Upper limit value	uint64_t
PJCMD_LIMITINFO_VALUE_ALLOC	Amount to be allocated	uint64_t

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring restriction information.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *item.*

PJCMD_ERROR_NODATA

The value of *indx* is out of range.

C.5 Getting Information on the Job ACL Function Settings

This section describes the functions for referencing information on the job ACL function settings.

Figure C.9 Requesting to Get Information on the Job ACL Function Settings

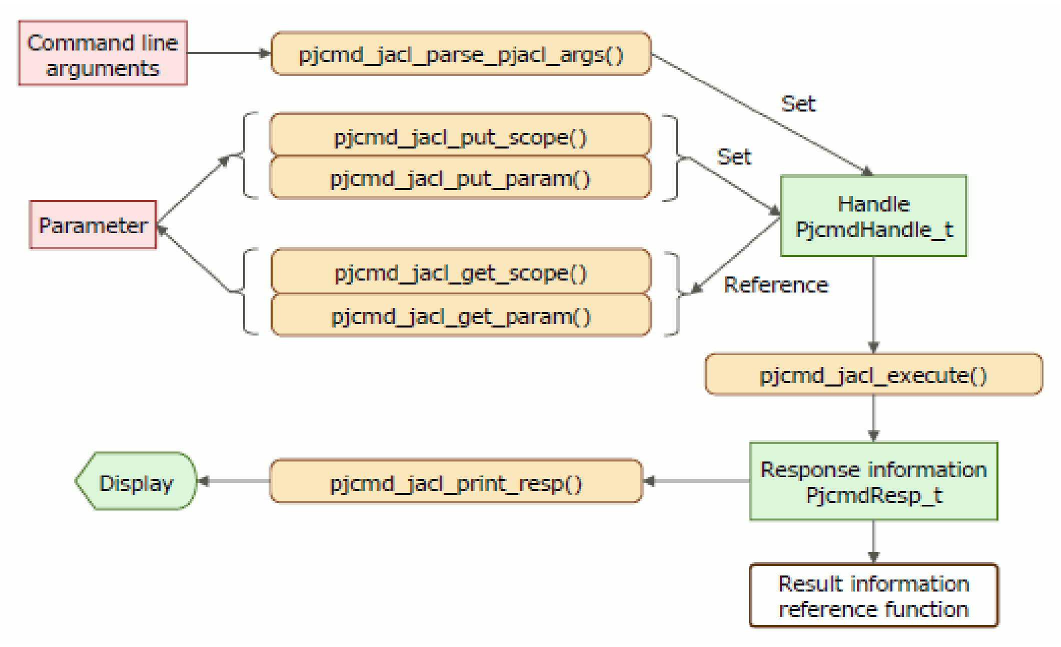
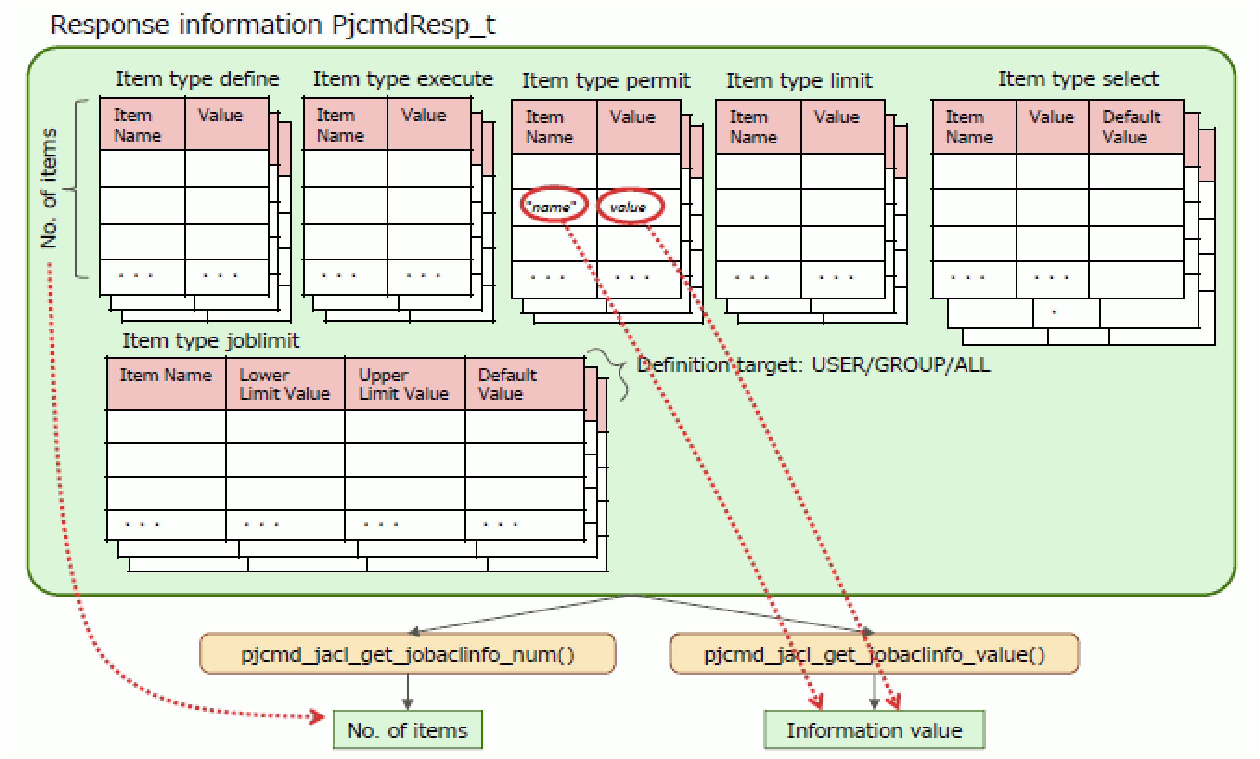


Figure C.10 Referencing Information on the Job ACL Function Settings



C.5.1 pjcmd_jacl_parse_pjacl_args()

```
pjcmd_result_t pjcmd_jacl_parse_pjacl_args(PjcmdHandle_t *handle_p, int argc, char **argv_pp)
```

This function analyzes command line arguments based on the specification of a pjacl command option and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting job ACL information.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option is not specified.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

Calling this function moves arguments other than options to the end of the *argv_pp*[] array.
If an unrecognizable option is detected, analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.

C.5.2 pjcmd_jacl_put_scope()

```
pjcmd_result_t pjcmd_jacl_put_scope(PjcmdHandle_t *handle_p, pjcmd_scope_t scope, void *val_p)
```

This function sets a range to get the job ACL information in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier indicating the type of target range for getting information (See the following table.)

val_p

Pointer to the storage area for the value indicating the target to get information. For example, if the value type to be set is char * type, the caller must prepare a storage area for the char * type value and specify a pointer (char **) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

scope	**val_p*	Type of **val_p*
PJCMD_SCOPE_RSCUNIT	Resource unit name (only one. equivalent to pjacl --rscunit) If this parameter is not set, the default resource unit of the execution user is applied.	char *
PJCMD_SCOPE_RSCGRP	Resource group name (only one. equivalent to pjacl --rscgrp) If this parameter is not set, resource unit information is obtained.	char *

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting job ACL information.

PJCMD_ERROR_INVALID_PARAM

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown or unspecifiable value is specified in *scope.*

PJCMD_ERROR_NOMEM

Memory acquisition failed.

C.5.3 pjcmd_jacl_get_scope()

```
pjcmd_result_t pjcmd_jacl_get_scope(const PjcmdHandle_t *handle_p, pjcmd_scope_t scope, void *val_p)
```

This function references the name of the target range (resource unit or resource group) for getting job ACL information that is set in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier of the target range to get information to be referenced. The identifiers that can be specified are the same as those for the pjcmd_jacl_put_scope() function.

val_p

A value is stored in **val_p* based on the *scope* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting job ACL information.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *scope.*

PJCMD_ERROR_NODATA

A specified *scope* is not set in a handle.

C.5.4 pjcmd_jacl_put_param()

```
pjcmd_result_t pjcmd_jacl_put_param(PjcmdHandle_t *handle_p, pjcmd_jacl_param_t param, void *val_p)
```

This function sets parameters in a handle that are related to job ACL information acquisition.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter related to the job ACL information to be obtained (See the following table.)

val_p

Pointer to the storage area for the value of parameter to be set. For example, if the value type to be set is uid_t type, the caller must prepare a storage area for the uid_t type value and specify a pointer (uid_t *) to the area in *val_p*. If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_JACL_USER_ID	Target user ID for which limit value information is obtained	uid_t
PJCMD_JACL_USER_NAME	Target user name for which limit value information is obtained	char *
PJCMD_JACL_GROUP_ID	Target group ID for which limit value information is obtained	gid_t
PJCMD_JACL_GROUP_NAME	Target group name for which limit value information is obtained	char *
PJCMD_JACL_DATA	Equivalent to the --data option in the pjacl command 0: Not specified (Default) 1: Specified This parameter does not affect the information to be obtained but does affect the results output by the pjcmd_jacl_print_resp() function.	int
PJCMD_JACL_DELIMITER	Equivalent to the specification of a character that is used to separate the information displayed using the --delimiter option in the pjacl command. This parameter does not affect the information to be obtained but does affect the results output by the pjcmd_jacl_print_resp() function. If this parameter is not specified, a comma (,) is used.	char *
PJCMD_JACL_HELP	Equivalent to the specification of the --help option in the pjacl command 0: Not specified (Default) 1: Specified This parameter does not affect the job information to be obtained.	int

As for the PJCMD_JACL_USER_ID or PJCMD_JACL_USER_NAME parameter, and the PJCMD_JACL_GROUP_ID or PJCMD_JACL_GROUP_NAME parameter, the one specified last is valid, respectively. If it is not specified, the user or group that calls the function is applied.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting job ACL information.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

val_p is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

C.5.5 pjcmd_jacl_get_param()

```
pjcmd_result_t pjcmd_jacl_get_param(const PjcmdHandle_t *handle_p, pjcmd_jacl_param_t param, void
*val_p)
```

This function references the set parameters in a handle that are related to job ACL information acquisition.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of the parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_jacl_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting job ACL information.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

C.5.6 pjcmd_jacl_execute()

```
PjcmdResp_t *pjcmd_jacl_execute(const PjcmdHandle_t *handle_p)
```

This function requests the job operation management function to get job ACL information based on a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about acquiring job ACL information.
The caller must release the obtained response information by using pjcmd_destroy_resp(). If a request to get job information has failed, NULL is returned and the cause is set in pjcmd_errcode.
The response information indicates whether a request has succeeded or failed. Whether or not job information has been obtained successfully needs to be checked with a result code in the response information by using the pjcmd_get_result() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting job ACL information.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node.
The function can be called from the login node and compute cluster management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_BUSY

Information acquisition cannot be requested because another operation request function is being processed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

PJCMD_ERROR_INTERNAL

Internal error

C.5.7 pjcmd_jacl_print_resp()

```
pjcmd_result_t pjcmd_jacl_print_resp(const PjcmdResp_t *resp_p)
```

This function outputs the results of job ACL information acquisition to the standard output based on the specification in the pjacl command.

[ARGUMENTS]

resp_p

Pointer to a response information

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring job ACL information.
- This is not response information that was successfully obtained.

C.5.8 pjcmd_jacl_get_jaclinfo_num()

```
int pjcmd_jacl_get_jaclinfo_num(const PjcmdResp_t *resp_p, pjcmd_jacl_class_t class,
pjcmd_jacl_type_t type)
```

This function gets the total number of job ACL definition items for specific definition targets and specific types of job ACL definitions from response information about acquiring job ACL information.

[ARGUMENTS]

resp_p

Pointer to a response information

class

Identifier indicating the definition targets (USER, GROUP, or ALL definitions) of the job ACL definition items to be obtained

class	Description
PJCMD_JACL_CLASS_USER	USER definitions
PJCMD_JACL_CLASS_GROUP	GROUP definitions
PJCMD_JACL_CLASS_ALL	ALL definitions

type

Identifier indicating the definition types (define, joblimit, execute, permit, limit, or select) of the job ACL definition items to be obtained

type	Description
PJCMD_JACL_TYPE_DEFINE	Definition type define
PJCMD_JACL_TYPE_JOBLIMIT	Definition type joblimit
PJCMD_JACL_TYPE_EXECUTE	Definition type execute
PJCMD_JACL_TYPE_PERMIT	Definition type permit
PJCMD_JACL_TYPE_LIMIT	Definition type limit
PJCMD_JACL_TYPE_SELECT	Definition type select

[RETURN VALUE]

Total number of job ACL definition items.
If the function fails, -1 is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring job ACL information.
- This is not response information that was successfully obtained.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *class* or *type.*

PJCMD_ERROR_INVALID_PARAM

There is no definition item indicated by the combination of *class* and *type*.

C.5.9 pjcmd_jacl_get_jaclinfo_value()

```
pjcmd_result_t pjcmd_jacl_get_jaclinfo_value(const PjcmdResp_t *resp_p, pjcmd_jacl_class_t class,
pjcmd_jacl_type_t type, int indx, pjcmd_jacl_info_t info, void *val_p)
```

This function references values of job ACL definition information.

[ARGUMENTS]

resp_p

Pointer to a response information

class

Identifier indicating the definition target (USER, GROUP, or ALL definitions) of a job ACL definition item. The values that can be specified are the same as those of the pjcmd_jacl_get_jaclinfo_num() function.

type

Identifier indicating the definition type (define, joblimit, execute, permit, limit, or select) of a job ACL definition item. The values that can be specified are the same as those of the pjcmd_jacl_get_jaclinfo_num() function.

indx

Index of definition items that needs to be referenced among the job ACL definition items of the definition target *class* and the definition type *type*. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the pjcmd_jacl_get_jaclinfo_num() function.

info

Identifier of a value to be referenced (See the following table.)

val_p

A value is stored in **val_p* based on the *info* type. The caller needs to prepare an area of a sufficient size according to the value type.

info	**val_p*	Type of **val_p*
PJCMD_JACL_INFO_NAME	Name of a job ACL definition item The name is the same as the definition item name that is described when setting the job ACL definition.	char *
PJCMD_JACL_INFO_DEFINE_VALUE	Value of a job ACL definition item whose definition type is define	char *

info	**val_p*	Type of **val_p*
PJCMD_JACL_INFO_JOBLIMIT_LOWER	Lower limit value of a job ACL definition item whose definition type is joblimit	uint64_t
PJCMD_JACL_INFO_JOBLIMIT_UPPER	Upper limit value of a job ACL definition item whose definition type is joblimit	uint64_t
PJCMD_JACL_INFO_JOBLIMIT_DEFAULT	Initial value of a job ACL definition item whose definition type is joblimit	uint64_t
PJCMD_JACL_INFO_EXECUTE_VALUE	Value of a job ACL definition item whose definition type is execute	char *
PJCMD_JACL_INFO_PERMIT_VALUE	Value of a job ACL definition item whose definition type is permit	char *
PJCMD_JACL_INFO_LIMIT_VALUE	Value of a job ACL definition item whose definition type is limit	uint64_t
PJCMD_JACL_INFO_SELECT_VALUE	Value of a job ACL definition item whose definition type is select	char *
PJCMD_JACL_INFO_SELECT_DEFAULT	Initial value of a job ACL definition item whose definition type is select	char *

If the value type is char * type (a pointer to a character string), the character string is in an area in response information. Therefore, operation is undetermined when the character string is referenced after the release of response information.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring job ACL information.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *class*, *type*, or *info.*

PJCMD_ERROR_INVALID_PARAM

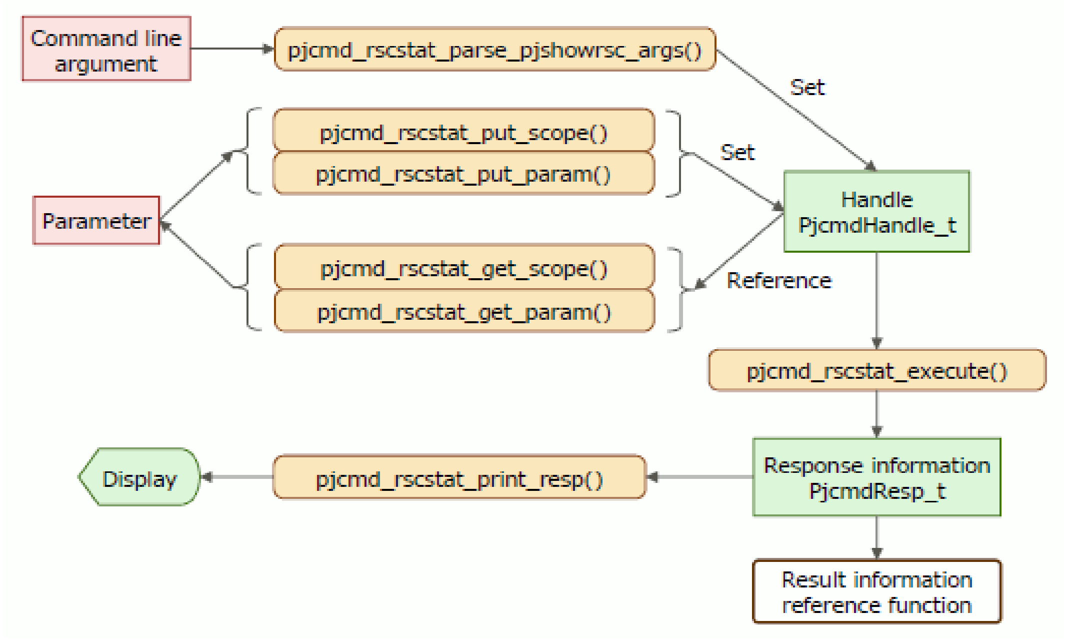
The specified parameter is invalid.

- The information *info* cannot be specified in the job ACL definition item *type.*
- There is no definition item indicated by the combination of *class* and *type*.
- The *indx* value is out of range.

C.6 Getting the Status of Job Resource Usage

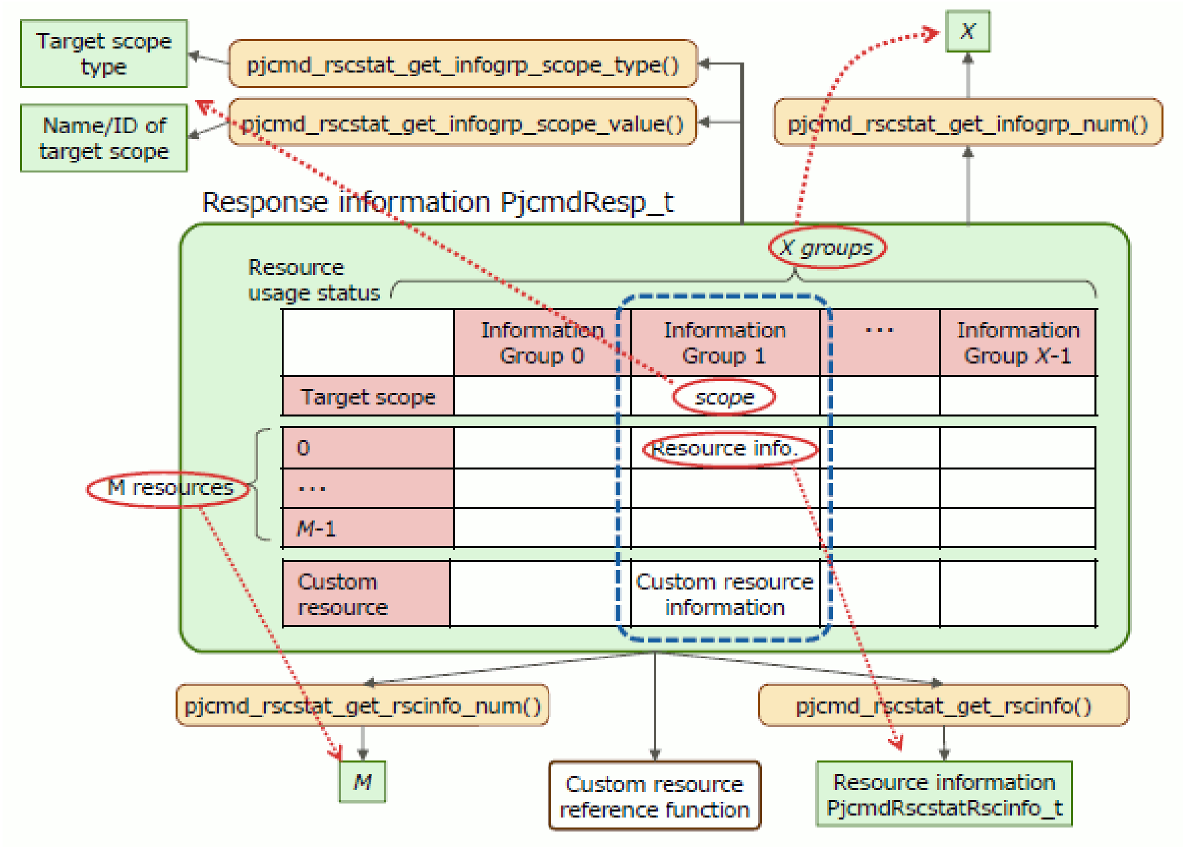
This section describes the functions for getting the status of job resource usage.

Figure C.11 Requesting to Get the Status of Job Resource Usage



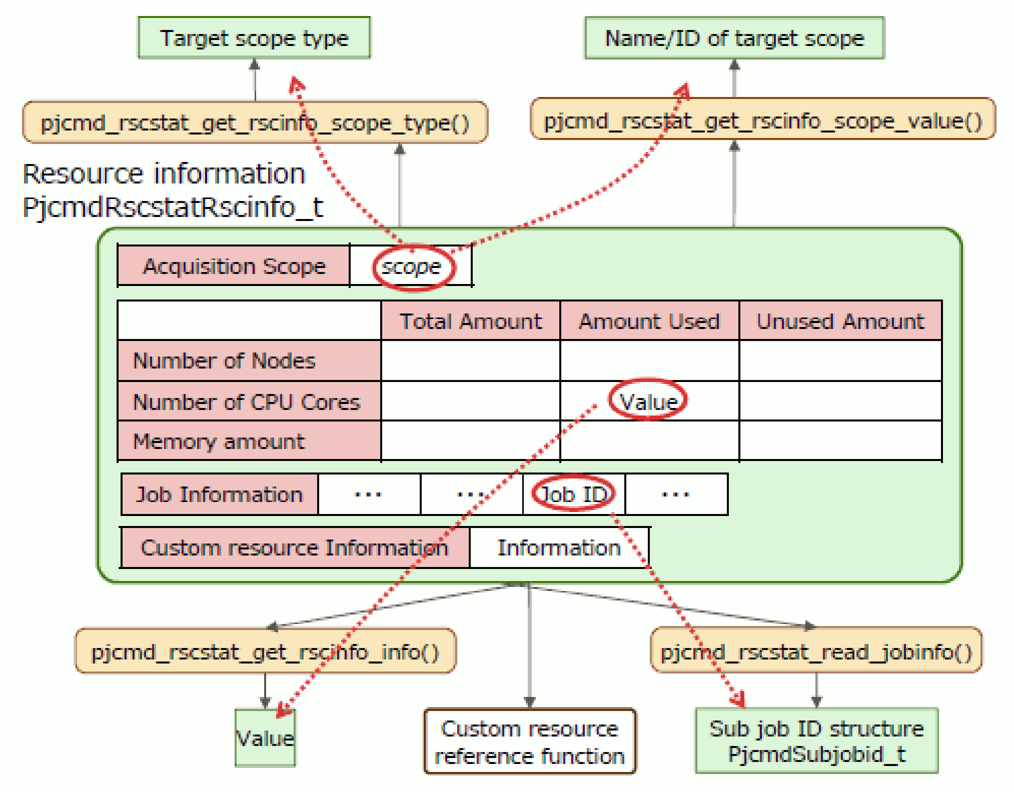
Command line argument
pjcmd_rscstat_parse_pjshowrsc_args()
Set
pjcmd_rscstat_put_scope()
pjcmd_rscstat_put_param()
Set
Handle PjcmdHandle_t
Parameter
pjcmd_rscstat_get_scope()
pjcmd_rscstat_get_param()
Reference
pjcmd_rscstat_execute()
Display
pjcmd_rscstat_print_resp()
Response information PjcmdResp_t
Result information reference function

Figure C.12 Referencing Information Group (Information Acquisition Unit)



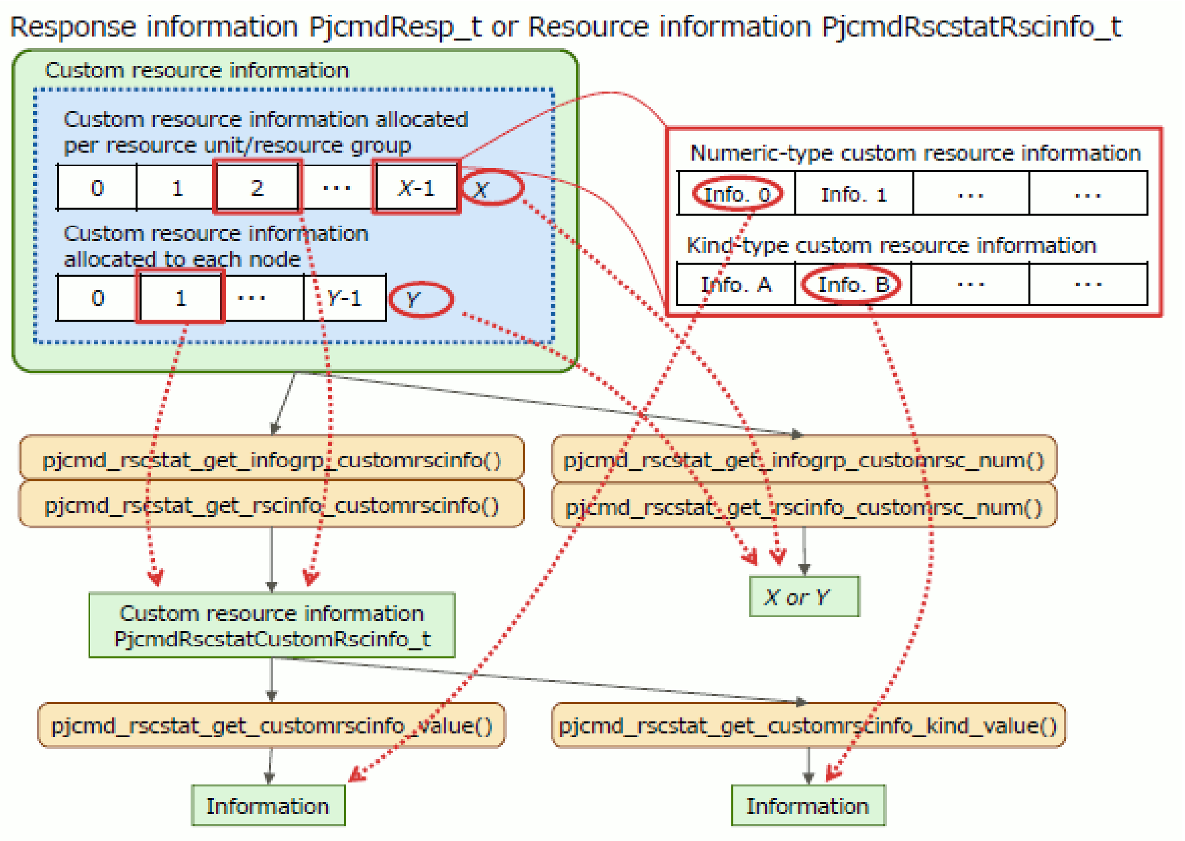
Target scope type
pjcmd_rscstat_get_infogrp_scope_type()
X
Name/ID of target scope
pjcmd_rscstat_get_infogrp_scope_value()
pjcmd_rscstat_get_infogrp_num()
Response information PjcmdResp_t
X groups
Resource usage status
Information Group 0
Information Group 1
・・・
Information Group X-1
Target scope
scope
0
Resource info.
M resources
・・・
M-1
Custom resource
Custom resource information
pjcmd_rscstat_get_rscinfo_num()
pjcmd_rscstat_get_rscinfo()
M
Custom resource reference function
Resource information PjcmdRscstatRscinfo_t

Figure C.13 Referencing Resource Information



Target scope type
Name/ID of target scope
pjcmd_rscstat_get_rscinfo_scope_type()
pjcmd_rscstat_get_rscinfo_scope_value()
Resource information
PjcmdRscstatRscinfo_t
Acquisition Scope
scope
Total Amount
Amount Used
Unused Amount
Number of Nodes
Number of CPU Cores
Memory amount
Value
Job Information
Job ID
Custom resource Information
Information
pjcmd_rscstat_get_rscinfo_info()
pjcmd_rscstat_read_jobinfo()
Value
Custom resource reference function
Sub job ID structure PjcmdSubjobid_t

Figure C.14 Referencing Custom Resource Information



Response information PjcmdResp_t or Resource information PjcmdRscstatRscinfo_t
Custom resource information
Custom resource information allocated per resource unit/resource group
0
1
2
X-1
X
Custom resource information allocated to each node
0
1
Y-1
Y
Numeric-type custom resource information
Info. 0
Info. 1
Kind-type custom resource information
Info. A
Info. B
pjcmd_rscstat_get_infogrp_customrscinfo()
pjcmd_rscstat_get_rscinfo_customrscinfo()
pjcmd_rscstat_get_infogrp_customrsc_num()
pjcmd_rscstat_get_rscinfo_customrsc_num()
X or Y
Custom resource information PjcmdRscstatCustomRscinfo_t
pjcmd_rscstat_get_customrscinfo_value()
pjcmd_rscstat_get_customrscinfo_kind_value()
Information
Information

C.6.1 pjcmd_rscstat_parse_pjshowrsc_args()

```
pjcmd_result_t pjcmd_rscstat_parse_pjshowrsc_args(PjcmdHandle_t *handle_p, int argc, char **argv_pp)
```

This function analyzes command line arguments based on the specification of a pjshowrsc command option and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting the resource usage status.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option is not specified.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_INTERNAL

Internal error

Calling this function moves arguments other than options to the end of the *argv_pp*[] array.
If an unrecognizable option is detected, analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.

C.6.2 pjcmd_rscstat_put_scope()

```
pjcmd_result_t pjcmd_rscstat_put_scope(PjcmdHandle_t *handle_p, pjcmd_scope_t scope, const void
*val_p, uint32_t n)
```

This function sets a range to get the resource usage status in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier indicating the type of target range for getting information (See the following table.)

val_p

Pointer to the storage area for the value indicating the target to get information. For example, if the value type to be set is char * type, the caller must prepare a storage area for the char * type value and specify a pointer (char **) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

n

Number of elements of *val_p*

scope	**val_p*	Type of **val_p*
PJCMD_SCOPE_CLUSTER	Cluster name (only one name, equivalent to pjstat -c) If this parameter is not set, the PXMYCLST environment variable value is applied. If the parameter and the environment variable are not set, all authorized clusters are applied. This parameter is valid only when called from the system management node. When the parameter is called from a node other than the system management node, the cluster to which the calling node belongs is applied. 1 must be specified in *n* for the parameter.	char *
PJCMD_SCOPE_NODEGRP	Array of a node group ID (number of elements is *n*, equivalent to pjshowrsc --nodegrp) If this parameter is set in a handle, administrator privileges are required for calling the pjcmd_rscstat_execute() function.	uint32_t *
PJCMD_SCOPE_NODEGRP_STR	Array of a node group ID (character string) (number of elements is *n*, equivalent to pjshowrsc --nodegrp) Node group IDs can also be used to express a range ("*ID1-ID2").* If this parameter is set in a handle, administrator privileges are required for calling the pjcmd_rscstat_execute() function.	char **
PJCMD_SCOPE_BOOTGRP	Array of a boot group ID (number of elements is *n*, equivalent to pjshowrsc --bootgr) If this parameter is set in a handle, administrator privileges are required for calling the pjcmd_rscstat_execute() function.	uint32_t *
PJCMD_SCOPE_BOOTGRP_STR	Array of a boot group ID (character string) (number of elements is *n*, equivalent to pjshowrsc --bootgr) Boot group IDs can also be used to express a range ("*ID1-ID2").* If this parameter is set in a handle, administrator privileges are required for calling the pjcmd_rscstat_execute() function.	char **
PJCMD_SCOPE_RSCUNIT	Array of a resource unit name (number of elements is *n*, equivalent to pjshowrsc --rscunit) If "*" is specified as a resource unit name, all resource units are targeted.	char **
PJCMD_SCOPE_RSCGRP	Array of a resource group name (number of elements is *n*, equivalent to pjshowrsc --rscgrp)	char **

scope	**val_p*	Type of **val_p*
	If "*" is specified as a resource group name, all resource groups are targeted.	
PJCMD_SCOPE_NODE	Array of a node ID (number of elements is *n*, equivalent to pjshowrsc -n)	uint32_t *
PJCMD_SCOPE_NODE_STR	Array of a node ID (character string) (number of elements is *n*, equivalent to pjshowrsc -n). Node IDs can also be used to express a range ("*ID1-ID2").*	char **

scope, the identifiers other than PJCMD_SCOPE_CLUSTER are exclusive of each other.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting the resource usage status.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *scope.*

PJCMD_ERROR_INVALID_PARAM

val_p or *n* is invalid.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

C.6.3 pjcmd_rscstat_get_scope()

```
pjcmd_result_t pjcmd_rscstat_get_scope(const PjcmdHandle_t *handle_p, pjcmd_scope_t scope, void
*val_p, uint32_t *n_p)
```

This function references the target range for getting the resource status that is set in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier of the target range to get information to be referenced. The identifiers that can be specified are the same as those for the pjcmd_rscstat_put_scope() function.

val_p

A value is stored in **val_p* based on the *scope* type. The caller needs to prepare an area of a sufficient size according to the value type.

n_p

The number of elements of *val_p* is stored in **n_p*. The caller needs to prepare the area.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting the resource usage status.

PJCMD_ERROR_INVALID_ARGUMENT

val_p or *n* is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *scope.*

PJCMD_ERROR_NODATA

A specified *scope* is not set in a handle.

C.6.4 pjcmd_rscstat_put_param()

```
pjcmd_result_t pjcmd_rscstat_put_param(PjcmdHandle_t *handle_p, pjcmd_rscstat_param_t param, const
void *val_p)
```

This function sets parameters in a handle that are related to the resource status acquisition.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of parameters related to the resource usage status to be obtained (See the following table.)

val_p

Pointer to the storage area for the value of parameter to be set. For example, if the value type to be set is int type, the caller must prepare a storage area for the int type value and specify a pointer (int *) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_RSCSTAT_VERBOSITY	Granularity of the resource status information to be obtained - PJCMD_RSCSTAT_VERBOSITY_SELF The information in the specified *scope* layer is obtained. (Default) - PJCMD_RSCSTAT_VERBOSITY_CHILD Information in the layers under the specified *scope* is also obtained. - PJCMD_RSCSTAT_VERBOSITY_NODE Even the node resource status is obtained.	int
PJCMD_RSCSTAT_INFO_LEVEL	Level of information to be obtained when acquisition of information in a node unit	int

param	**val_p*	Type of **val_p*
	(PJCMD_RSCSTAT_VERBOSITY_NODE) is specified in the PJCMD_RSCSTAT_VERBOSITY parameter (equivalent to pjshowrsc -v) 0: Obtain compute resource amount (number of nodes, number of CPU cores, memory amount, local file system size) (Default) 1: Also obtain job IDs of running jobs in addition to above information 2: Also obtain custom resource information in addition to above information 3: Also obtain job IDs that use nodes as communication routes, in addition to above information	
PJCMD_RSCSTAT_EXCLUSIVE	Whether or not to exclude jobs in other resource groups that share resources, from the resource usage amount of a resource group to be obtained (equivalent to pjshowrsc --exclusive) 0: Include resources of other resource groups (Default) 1: Exclude resources of other resource groups	int
PJCMD_RSCSTAT_CUSTOMRSC	Whether or not to get custom resource information is specified when a resource unit or resource group is the target (equivalent to pjshowrsc --custom-resource). 0: Do not obtain custom resource information (Default) 1: Obtain custom resource information	int
PJCMD_RSCSTAT_STATUS	Whether or not to get information for all nodes is specified when nodes are targeted. 0: Obtain only information for available nodes (Default) 1: Obtain information for all nodes	int
PJCMD_RSCSTAT_RAW	Equivalent to the specification of the --raw option in the pjshowrsc command 0: Not specified (Default) 1: Specified This parameter does not affect the information to be obtained but does affect the results output by the pjcmd_rscstat_print_resp() function.	int
PJCMD_RSCSTAT_DATA	Equivalent to the specification of the --data option in the pjshowrsc command 0: Not specified (Default) 1: Specified This parameter does not affect the information to be obtained but does affect the results output by the pjcmd_rscstat_print_resp() function.	int
PJCMD_RSCSTAT_DELIMITER	Equivalent to the specification of a character that is used to separate the information displayed using the --delimiter option in the pjshowrsc command This parameter does not affect the information to be obtained but does affect the results output by the pjcmd_rscstat_print_resp() function. If this parameter is not specified, a comma (,) is used.	char *
PJCMD_RSCSTAT_HELP	Equivalent to the specification of the --help option in the pjshowrsc command	int

param	**val_p*	Type of **val_p*
	0: Not specified (Default) 1: Specified This parameter does not affect the information to be obtained.	

Information on job resource usage is obtained in the following units based on *scope* specified in the pjcmd_rscstat_put_scope() function and the granularity of information specified in the PJCMD_RSCSTAT_VERBOSITY parameter of the pjcmd_rscstat_put_param() function.

scope	Granularity (*)	Unit of Information to be Obtained
Cluster	SELF	Cluster unit
	CHILD	In system with compute cluster sub management node: Node group In system without compute cluster sub management node: Boot group
	NODE	Node
Node group	SELF	Node group
	CHILD	Boot group
	NODE	Node
Boot group	SELF	Boot group
	CHILD	Node
	NODE	Node
Resource unit	SELF	Resource unit
	CHILD	Node
	NODE	Node
Resource group	SELF	Resource group
	CHILD	Node
	NODE	Node
Node	Any	Node

(*) "SELF" refers to PJCMD_RSCSTAT_VERBOSITY_SELF, "CHILD" refers to PJCMD_RSCSTAT_VERBOSITY_CHILD, and "NODE" refers to PJCMD_RSCSTAT_VERBOSITY_NODE.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting the resource usage status.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

C.6.5 pjcmd_rscstat_get_param()

```
pjcmd_result_t pjcmd_rscstat_get_param(const PjcmdHandle_t *handle_p, pjcmd_rscstat_param_t param,
void *val_p)
```

This function references the set parameters in a handle that are related to resource status acquisition.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of the parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_rscstat_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting the resource usage status.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

C.6.6 pjcmd_rscstat_execute()

```
PjcmdResp_t *pjcmd_rscstat_execute(const PjcmdHandle_t *handle_p)
```

This function requests the job operation management function to get the resource usage status based on a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about acquiring the resource usage status.
The caller must release the obtained response information by using pjcmd_destroy_resp(). If a request to get job information has failed, NULL is returned and the cause is set in pjcmd_errcode.
The response information indicates whether a request has succeeded or failed. Whether or not job information has been obtained successfully needs to be checked with a result code in the response information by using the pjcmd_get_result() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting the resource usage status.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node.
The function can be called from the login node, compute cluster management node and system management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_CONNECT

Communication with the daemon of the job operation management function has failed.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_BUSY

An operation cannot be requested because another operation request function is being processed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_SIGNAL

The process is interrupted because a signal has been received.

PJCMD_ERROR_INTERNAL

Internal error

C.6.7 pjcmd_rscstat_print_resp()

```
pjcmd_result_t pjcmd_rscstat_print_resp(const PjcmdResp_t *resp_p)
```

This function outputs the results of resource usage status acquisition to the standard output based on the specification of the pjshowrsc command.

[ARGUMENTS]

resp_p

Pointer to a response information

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring resource usage status.
- This is not response information that was successfully obtained.

C.6.8 pjcmd_rscstat_get_infogrp_num()

```
int pjcmd_rscstat_get_infogrp_num(const PjcmdResp_t *resp_p)
```

This function gets the number of information groups included in response information about acquiring the resource usage status.
An information group is information obtained in the information acquisition unit (cluster, node group, boot group, etc.) that is specified by the pjcmd_rscstat_put_scope() function when getting information.

[ARGUMENTS]

resp_p

Pointer to a response information

[RETURN VALUE]

Number of information groups.
If the function fails, -1 is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring resource usage status.
- This is not response information that was successfully obtained.

C.6.9 pjcmd_rscstat_get_infogrp_scope_type()

```
pjcmd_scope_t pjcmd_rscstat_get_infogrp_scope_type(const PjcmdResp_t *resp_p, int indx)
```

This function returns the type of information acquisition unit for a specific information group that is included in response information about acquiring the resource usage status.

[ARGUMENTS]

resp_p

Pointer to a response information

indx

Index of information groups. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the pjcmd_rscstat_get_infogrp_num() function.

[RETURN VALUE]

Identifier indicating a unit for getting information on an information group.

Identifier	Meaning
PJCMD_SCOPE_CLUSTER	Cluster unit

Identifier	Meaning
PJCMD_SCOPE_NODEGRP	Node group unit
PJCMD_SCOPE_BOOTGRP	Boot group unit
PJCMD_SCOPE_RSCUNIT	Resource unit unit
PJCMD_SCOPE_RSCGRP	Resource group unit
PJCMD_SCOPE_NODE	Node unit

If the function fails, -1 is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring resource usage status.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_PARAM

The *indx* value is out of range.

C.6.10 pjcmd_rscstat_get_infogrp_scope_value()

```
pjcmd_result_t pjcmd_rscstat_get_infogrp_scope_value(const PjcmdResp_t *resp_p, int indx,
pjcmd_scope_t scope, void *val_p)
```

This function gets the name (cluster name, resource unit name, or resource group name) or ID (node group ID, boot group ID, or node ID) of an information acquisition unit for a specific information group that is included in response information about acquiring the resource usage status.

[ARGUMENTS]

resp_p

Pointer to a response information

indx

Index of information groups. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the pjcmd_rscstat_get_infogrp_num() function.

scope

Type of information acquisition unit to get names or IDs. (See the following table.)
If the *scope* argument is different from the type of unit to get information groups, an error occurs.

val_p

A value is stored in **val_p* based on the *scope* type. The caller needs to prepare an area of a sufficient size according to the value type.

scope	**val_p*	Type of **val_p*
PJCMD_SCOPE_CLUSTER	Cluster name The area specified by **val_p* is undetermined after the release of response information.	char *
PJCMD_SCOPE_NODEGRP	Node group ID	uint32_t
PJCMD_SCOPE_BOOTGRP	Boot group ID	uint32_t

scope	**val_p*	Type of **val_p*
PJCMD_SCOPE_RSCUNIT	Resource unit name The area specified by **val_p* is undetermined after the release of response information.	char *
PJCMD_SCOPE_RSCGRP	Resource group name The area specified by **val_p* is undetermined after the release of response information.	char *
PJCMD_SCOPE_NODE	Node ID	uint32_t

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring resource usage status.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *scope.*

PJCMD_ERROR_INVALID_PARAM

- *scope* does not match the information group acquisition unit.
- The *indx* value is out of range.

C.6.11 pjcmd_rscstat_get_rscinfo_num()

```
int pjcmd_rscstat_get_rscinfo_num(const PjcmdResp_t *resp_p, int indx)
```

This function gets the amount of resource information included in a specific information group in response information about acquiring the resource usage status.

[ARGUMENTS]

resp_p

Pointer to a response information

indx

Index of information groups. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the pjcmd_rscstat_get_infogrp_num() function.

[RETURN VALUE]

Amount of resource information.
If the function fails, -1 is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring resource usage status.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_PARAM

The *indx* value is out of range.

C.6.12 pjcmd_rscstat_get_infogrp_customrsc_num()

```
pjcmd_result_t pjcmd_rscstat_get_infogrp_customrsc_num(const PjcmdResp_t *resp_p, int indx,
pjcmd_rscstat_customrsc_alloc_t type, uint32_t *num_p)
```

This function gets the number of custom resources for a specific information group.

[ARGUMENTS]

resp_p

Pointer to a response information

indx

Index of information groups. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the pjcmd_rscstat_get_infogrp_num() function.

type

Custom resource allocation type

Identifier	Meaning
PJCMD_RSCSTAT_CUSTOMRSC_ALLOC_RU_RG	Custom resources that are allocated to each resource unit or resource group
PJCMD_RSCSTAT_CUSTOMRSC_ALLOC_NODE	Custom resources that are allocated to each node

num_p

The number of custom resources is stored in **num_p*. The caller needs to prepare the area.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring resource usage status.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_PARAM

The *indx* value is out of range.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *type.*

C.6.13 pjcmd_rscstat_get_infogrp_customrscinfo()

```
PjcmdRscstatCustomRscinfo_t * pjcmd_rscstat_get_infogrp_customrscinfo(const PjcmdResp_t *resp_p, int
indx, pjcmd_rscstat_customrsc_alloc_t type, uint32_t cs_indx)
```

This function gets custom resource information for a specific information group.

[ARGUMENTS]

resp_p

Pointer to a response information

indx

Index of information groups. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the pjcmd_rscstat_get_infogrp_num() function.

type

Custom resource allocation type

Identifier	Meaning
PJCMD_RSCSTAT_CUSTOMRSC_ALLOC_RU_RG	Custom resources that are allocated to each resource unit or resource group
PJCMD_RSCSTAT_CUSTOMRSC_ALLOC_NODE	Custom resources that are allocated to each node

cs_indx

Index of custom resources. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the pjcmd_rscstat_get_infogrp_customrsc_num() function.

[RETURN VALUE]

Pointer to custom resource information. If information that can be obtained is referenced after the release of response information, operation is undetermined.
If the function fails, NULL is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring resource usage status.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_PARAM

The *indx* value is out of range.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *type.*

PJCMD_ERROR_NODATA

The value of *cs_indx* is out of range.

C.6.14 pjcmd_rscstat_get_rscinfo()

```
PjcmdRscstatRscinfo_t *pjcmd_rscstat_get_rscinfo(const PjcmdResp_t *resp_p, int infogrp_indx, int
rscinfo_indx)
```

This function gets one piece of resource information in a specific information group.

[ARGUMENTS]

resp_p

Pointer to a response information

infogrp_indx

Index of information groups. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the pjcmd_rscstat_get_infogrp_num() function.

rscinfo_indx

Index of resource information to be referenced in an information group. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the pjcmd_rscstat_get_rscinfo_num() function.

[RETURN VALUE]

Pointer to a resource information.
If the function fails, NULL is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not response information about acquiring resource usage status.
- This is not response information that was successfully obtained.

PJCMD_ERROR_INVALID_PARAM

The *infogrp_indx* or *rscinfo_indx* value is out of range.

C.6.15 pjcmd_rscstat_get_rscinfo_scope_type()

```
pjcmd_scope_t pjcmd_rscstat_get_rscinfo_scope_type(const PjcmdRscstatRscinfo_t *rscinfo_p)
```

This function gets the type of information acquisition unit for resource information.

[ARGUMENTS]

rscinfo_p

Pointer to resource information. This is a value returned by the pjcmd_rscstat_get_rscinfo() function.

[RETURN VALUE]

Identifier indicating a unit for getting information of resource.

Identifier	Meaning
PJCMD_SCOPE_CLUSTER	Cluster unit
PJCMD_SCOPE_NODEGRP	Node group unit
PJCMD_SCOPE_BOOTGRP	Boot group unit
PJCMD_SCOPE_RSCUNIT	Resource unit unit
PJCMD_SCOPE_RSCGRP	Resource group unit
PJCMD_SCOPE_NODE	Node unit

If the function fails, -1 is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

Resource information *rscinfo_p* is invalid (NULL).

C.6.16 pjcmd_rscstat_get_rscinfo_scope_value()

```
pjcmd_result_t pjcmd_rscstat_get_rscinfo_scope_value(const PjcmdRscstatRscinfo_t *rscinfo_p,
pjcmd_scope_t scope, void *val_p)
```

This function gets the name (cluster name, resource unit name, or resource group name) or ID (node group ID, boot group ID, or node ID) of a unit to get resource information.

[ARGUMENTS]

rscinfo_p

Pointer to resource information. This is a value returned by the pjcmd_rscstat_get_rscinfo() function.

scope

Type of information acquisition unit. (See the following table.)
If *scope* is different from the type of unit to get resource information, an error occurs.

val_p

A value is stored in **val_p* based on the *scope* type. The caller needs to prepare an area of a sufficient size according to the value type.

scope	**val_p*	Type of **val_p*
PJCMD_SCOPE_CLUSTER	Cluster name The area specified by *val_p is undetermined after the release of response information.	char *
PJCMD_SCOPE_NODEGRP	Node group ID	uint32_t
PJCMD_SCOPE_BOOTGRP	Boot group ID	uint32_t
PJCMD_SCOPE_RSCUNIT	Resource unit name The area specified by *val_p is undetermined after the release of response information.	char *
PJCMD_SCOPE_RSCGRP	Resource group name The area specified by *val_p is undetermined after the release of response information.	char *
PJCMD_SCOPE_NODE	Node ID	uint32_t

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

rscinfo_p or *val_p* is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *scope.*

PJCMD_ERROR_INVALID_PARAM

scope does not match the resource information acquisition unit.

C.6.17 pjcmd_rscstat_get_rscinfo_info()

```
pjcmd_result_t pjcmd_rscstat_get_rscinfo_info(const PjcmdRscstatRscinfo_t *rscinfo_p,
pjcmd_rscstat_rsc_name_t rscname, pjcmd_rscstat_rsc_value_t type, void *val_p)
```

This function references the total amount of specific resources, usage amount, or available amount from resource information.

[ARGUMENTS]

rscinfo_p

Pointer to a resource information

rscname

Identifier of a resource name to be referenced (See the following table.)

type

Identifier of a resource amount to be referenced (See the following table.)

val_p

A value is stored in **val_p* based on the *rscname* type. The caller needs to prepare an area of a sufficient size according to the value type.

rscname	**val_p*	Type of **val_p*
PJCMD_RSCSTAT_RSC_NODE	Number of compute nodes	uint32_t
PJCMD_RSCSTAT_RSC_CPU	Number of CPU cores of compute node	uint32_t
PJCMD_RSCSTAT_RSC_MEM	Memory amount of compute node	uint64_t

type	Description
PJCMD_RSCSTAT_RSC_TOTAL	Total amount of *rscname* resource
PJCMD_RSCSTAT_RSC_ALLOC	Usage amount of *rscname* resource
PJCMD_RSCSTAT_RSC_FREE	Unused amount of *rscname* resource

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

rscinfo_p or *val_p* is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *rscname* or *type.*

C.6.18 pjcmd_rscstat_read_jobinfo()

```
PjcmdSubjobid_t *pjcmd_rscstat_read_jobinfo(PjcmdRscstatRscinfo_t *rscinfo_p,
pjcmd_rscstat_jobtype_t type)
```

If resource information is the information in a node unit, this function returns one sub job ID structure of a job that is being executed using the node resource, or one sub job ID structure of a job that uses the node as a communication route. The next relevant job is returned every time this function is called.

[ARGUMENTS]

rscinfo_p

Pointer to a resource information

type

Type of job to be referenced

Identifier	Meaning
PJCMD_RSCSTAT_RUNNING_JOBS	Job that is being executed using a node resource
PJCMD_RSCSTAT_JOBS_USING_ROUTE	Job that uses a node as a communication route

[RETURN VALUE]

Pointer to a sub job ID structure.
The contents of the area specified by the obtained pointer are undetermined after calling this function next time.
If the function fails, NULL is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

rscinfo_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *type.*

PJCMD_ERROR_NODATA

There is no next job.

C.6.19 pjcmd_rscstat_get_rscinfo_customrsc_num()

```
pjcmd_result_t pjcmd_rscstat_get_rscinfo_customrsc_num(const PjcmdRscstatRscinfo_t *rscinfo_p,
pjcmd_rscstat_customrsc_alloc_t type, uint32_t *num_p)
```

This function gets the number of custom resources included in resource information.

[ARGUMENTS]

rscinfo_p

Pointer to a resource information

type

Custom resource allocation type

Identifier	Meaning
PJCMD_RSCSTAT_CUSTOMRSC_ALLOC_RU_RG	Custom resources that are allocated to each resource unit or resource group
PJCMD_RSCSTAT_CUSTOMRSC_ALLOC_NODE	Custom resources that are allocated to each node

num_p

The number of custom resources is stored in **num_p.* The caller needs to reserve the area.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

rscinfo_p or *val_p* is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *type.*

C.6.20 pjcmd_rscstat_get_rscinfo_customrscinfo()

```
PjcmdRscstatCustomRscinfo_t *pjcmd_rscstat_get_rscinfo_customrscinfo(const PjcmdRscstatRscinfo_t
*rscinfo_p, pjcmd_rscstat_customrsc_alloc_t type, uint32_t indx)
```

This function gets information on a specific custom resource that is included in resource information.

[ARGUMENTS]

rscinfo_p

Pointer to a resource information

type

Custom resource allocation type

indx

Index of custom resources. The specifiable value ranges from 0 to a value that is obtained by subtracting 1 from a value that is obtained by the pjcmd_rscstat_get_rscinfo_customrsc_num() function.

[RETURN VALUE]

Custom resource information.
If the function fails, NULL is returned, and the cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

rscinfo_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *type.*

PJCMD_ERROR_NODATA

The value of *indx* is out of range.

C.6.21 pjcmd_rscstat_get_customrscinfo_value()

```
pjcmd_result_t pjcmd_rscstat_get_customrscinfo_value(const PjcmdRscstatCustomRscinfo_t *rscinfo_p,
pjcmd_rscstat_customrsc_value_t type, void *val_p)
```

This function references custom resource information individually.

[ARGUMENTS]

rscinfo_p

Pointer to custom resource information. This is the information obtained by the pjcmd_rscstat_get_rscinfo_customrscinfo() function.

type

Identifier of information to be referenced (See the following table.)

val_p

A value is stored in **val_p* based on the *type* type. The caller needs to prepare an area of a sufficient size according to the value type.

type	**val_p*	Type of **val_p*
PJCMD_RSCSTAT_CUSTOMRSC_NAME	Custom resource name Operation is undetermined when the obtained value is referenced after the release of response information.	char *
PJCMD_RSCSTAT_CUSTOMRSC_TYPE	Custom resource amount type PJCMD_RSCSTAT_CUSTOMRSC_NUM Custom resources are of the numerical value type. PJCMD_RSCSTAT_CUSTOMRSC_KIND Custom resources are of the kind type.	int
PJCMD_RSCSTAT_CUSTOMRSC_TOTAL	Total amount of custom resources (for custom resources of numerical value type)	int64_t
PJCMD_RSCSTAT_CUSTOMRSC_ALLOC	Amount of custom resources in use (for custom resources of numerical value type)	int64_t
PJCMD_RSCSTAT_CUSTOMRSC_FREE	Amount of unused custom resources (for custom resources of numerical value type)	int64_t
PJCMD_RSCSTAT_CUSTOMRSC_KIND_NUM	Number of custom resources (for custom resources of kind type)	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

rscinfo_p or *val_p* is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *type*.

C.6.22 pjcmd_rscstat_get_customrscinfo_kind_value()

```
pjcmd_result_t pjcmd_rscstat_get_customrscinfo_kind_value(const PjcmdRscstatCustomRscinfo_t
*rscinfo_p, int indx, pjcmd_rscstat_customrsc_kind_value_t type, void *val_p)
```

This function references the resource information when the type of custom resource information is the kind type.

[ARGUMENTS]

info_p

Pointer to custom resource information (kind type) *indx*

Index of custom resource information (kind type) to be referenced. The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from a value that is obtained by the pjcmd_rscstat_get_customrscinfo_value() function.

type

Identifier of information to be referenced (See the following table.)

val_p

A value is stored in **val_p* based on the *type* type. The caller needs to prepare an area of a sufficient size according to the value type.

type	**val_p*	Type of **val_p*
PJCMD_RSCSTAT_CUSTOMRSC_KIND_NAME	Name of custom resource type	char *
PJCMD_RSCSTAT_CUSTOMRSC_KIND_TOTAL	Total amount of custom resources	int64_t
PJCMD_RSCSTAT_CUSTOMRSC_KIND_ALLOC	Amount of custom resources in use	int64_t
PJCMD_RSCSTAT_CUSTOMRSC_KIND_FREE	Amount of unused custom resources	int64_t

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_ARGUMENT

- *rscinfo_p* or *val_p* is invalid (NULL).
- *rscinfo_p* is not a custom resource of the kind type.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *type.*

PJCMD_ERROR_NODATA

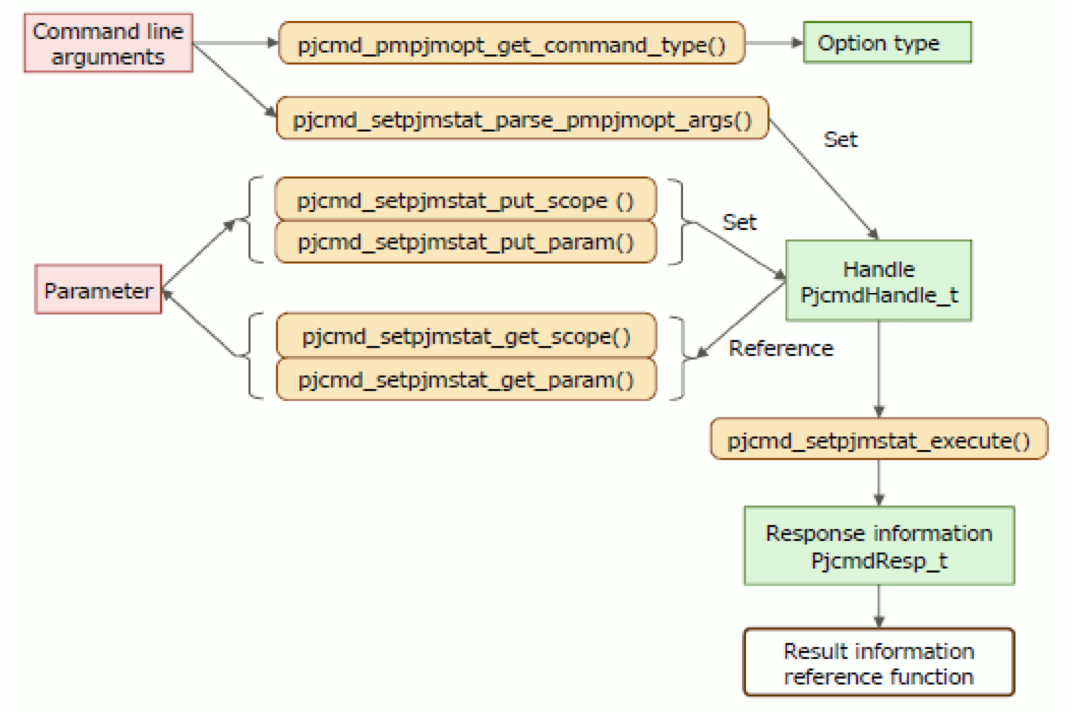
The value of *indx* is out of range.

Appendix D Job Operation Control API Reference

D.1 Setting of Job Submission/Job Execution Permissions

This section describes the functions for setting permissions for job submission and job execution in job operations.

Figure D.1 Requesting to Set Permissions for Job Submission and Job Execution


Command line arguments
pjcmd_pmpjmopt_get_command_type()
Option type
pjcmd_setpjmstat_parse_pmpjmopt_args()
Set
pjcmd_setpjmstat_put_scope ()
pjcmd_setpjmstat_put_param()
Set
Parameter
Handle PjcmdHandle_t
pjcmd_setpjmstat_get_scope()
pjcmd_setpjmstat_get_param()
Reference
pjcmd_setpjmstat_execute()
Response information PjcmdResp_t
Result information reference function

D.1.1 pjcmd_pmpjmopt_get_command_type()

```
pjcmd_pmpjmopt_command_type_t pjcmd_pmpjmopt_get_command_type(int argc, char **argv_pp)
```

This function analyzes command line arguments as arguments of the pmpjmopt command to determine whether --show-rsc-ug or --set-rsc-ug option is specified.

[ARGUMENTS]

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

Operation type of the pmpjmopt command

PJCMD_PMPJMOPT_SET_RSC_UG

The --set-rsc-ug option is specified.

PJCMD_PMPJMOPT_SHOW_RSC_UG

The --show-rsc-ug option is specified.

PCMD_PMPJMOPT_UNKNOWN_COMMAND_TYPE

Nothing could be determined (both options are specified, or either option is not specified).

[pjcmd_errcode]

PJCMD_SUCCESS

Success. This code is set when the return values are PJCMD_PMPJMOPT_SET_RSC_UG and PJCMD_PMPJMOPT_SHOW_RSC_UG.

PJCMD_ERROR_UNKNOWN_OPTION

Nothing could be determined. This code is set when the return value is PCMD_PMPJMOPT_UNKNOWN_COMMAND_TYPE.

D.1.2 pjcmd_setpjmstat_parse_pmpjmopt_args()

```
pjcmd_result_t pjcmd_setpjmstat_parse_pmpjmopt_args(PjcmdHandle_t *handle_p, int argc, char
**argv_pp)
```

This function analyzes command line arguments based on the specification used when the --set-rsc-ug option in the pmpjmopt command is specified, and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for setting permissions for job submission and execution.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option is not specified.
- An exclusive option is specified.

Calling this function moves arguments other than options to the end of the *argv_pp*[] array.
If an unrecognizable option is detected, analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.
This function recognizes only the --set-rsc-ug option in the pmpjmopt command and the options that can be specified with this option at the same time.

D.1.3 pjcmd_setpjmstat_put_scope()

```
pjcmd_result_t pjcmd_setpjmstat_put_scope(PjcmdHandle_t *handle_p, pjcmd_scope_t scope, void *val_p)
```

This function sets the target range for job submission/execution permission in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier indicating the target range (See the following table.)

val_p

Pointer to the storage area for the value indicating the target range. For example, if the value type to be set is char * type, the caller must prepare a storage area for the char * type value and specify a pointer (char **) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

scope	**val_p*	Type of **val_p*
PJCMD_SCOPE_CLUSTER	Cluster name (only one name, equivalent to pmpjmopt -c) If this parameter is not set, the PXMYCLST environment variable value is applied. If the parameter and the environment variable are not set, an error occurs in the pjcmd_setpjmstat_execute() function. This parameter is valid only when called from the system management node. When the parameter is called from a node other than the system management node, the cluster to which the calling node belongs is applied.	char *
PJCMD_SCOPE_RSCUNIT	Resource unit name (only one name; equivalent to pmpjmopt --rscunit) Specification of a resource unit name is required. If a resource unit name is not specified, an error occurs in the pjcmd_setpjmstat_execute() function.	char *
PJCMD_SCOPE_RSCGRP	Resource group name array (equivalent to pmpjmopt --rscgrp) The last element must be (char *)NULL. "*" in a resource group name indicates all resource groups (equivalent to pmpjmopt --all-rsc-groups). If a resource group is not specified, the setting of each resource group is based on the resource unit setting.	char **

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for setting permissions for job submission and execution.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown or unspecifiable value is specified in *scope.*

PJCMD_ERROR_INVALID_PARAM

val_p is invalid (NULL).

PJCMD_ERROR_NOMEM

Memory acquisition failed.

D.1.4 pjcmd_setpjmstat_get_scope()

```
pjcmd_result_t pjcmd_setpjmstat_get_scope(const PjcmdHandle_t *handle_p, pjcmd_scope_t scope, void
*val_p)
```

This function references the target range (cluster, resource unit, or resource group) of the set job submission/execution permission in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier of the target range to be referenced. The identifiers that can be specified are the same as those for the pjcmd_setpjmstat_put_scope() function.

val_p

A value is stored in **val_p* based on the *scope* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for setting permissions for job submission and execution.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown or unspecifiable value is specified in *scope.*

PJCMD_ERROR_NODATA

A specified *scope* is not set in a handle.

D.1.5 pjcmd_setpjmstat_put_param()

```
pjcmd_result_t pjcmd_setpjmstat_put_param(PjcmdHandle_t *handle_p, pjcmd_setpjmstat_param_t param,
const void *val_p)
```

This function sets parameters in a handle that are related to job submission/execution permission.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter related to job submission or execution permissions (See the following table.)

val_p

Pointer to the storage area for the value of parameter to be set. For example, if the value type to be set is int type, the caller must prepare a storage area for the int type value and specify a pointer (int *) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_SETPJMSTAT_JOB_SUBMIT	Job submission permission 0: Do not permit submission of new job 1: Permit submission of new job If this parameter is not set, the job submission permission does not change.	int
PJCMD_SETPJMSTAT_JOB_EXECUTE	Job execution permission 0: Do not permit execution of new job 1: Permit execution of new job If this parameter is not set, the job execution permission does not change.	int
PJCMD_SETPJMSTAT_HELP	Equivalent to the specification of the --help option in the pmpjmopt command 0: Not specified (Default) 1: Specified This parameter does not affect permissions for job submission and execution.	int

If neither the PJCMD_SETPJMSTAT_JOB_SUBMIT parameter nor PJCMD_SETPJMSTAT_JOB_EXECUTE parameter is set, an error occurs when the pjcmd_setpjmstat_execute() function is called.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for setting permissions for job submission and execution.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

D.1.6 pjcmd_setpjmstat_get_param()

```
pjcmd_result_t pjcmd_setpjmstat_get_param(const PjcmdHandle_t *handle_p, pjcmd_setpjmstat_param_t
param, void *val_p)
```

This function references the set parameters in a handle that are related to permissions for job submission and execution.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_setpjmstat_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for setting permissions for job submission and execution.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

D.1.7 pjcmd_setpjmstat_execute()

```
PjcmdResp_t *pjcmd_setpjmstat_execute(const PjcmdHandle_t *handle_p)
```

This function requests the job operation management function to permit job submission and execution based on a handle. Root privileges are required for calling the function.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about permissions for job submission and execution.
The caller must release the obtained response information by using pjcmd_destroy_resp(). If the request to permit job submission or execution fails, NULL is returned and pjcmd_errcode is set.
The response information indicates whether the request has succeeded or failed. The determination of whether the request to permit job submission or execution has been accepted successfully needs to be checked with a result code based on the response information by using the pjcmd_get_result() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for setting permissions for job submission and execution.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node. The function can only be called from the system management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_INTERNAL

Internal error

D.2 Referencing of Job Submission and Execution Permission Information

This section describes the functions for referencing job submission and execution permission information for job operations.

Figure D.2 Requesting to Get Job Submission and Execution Permission Information

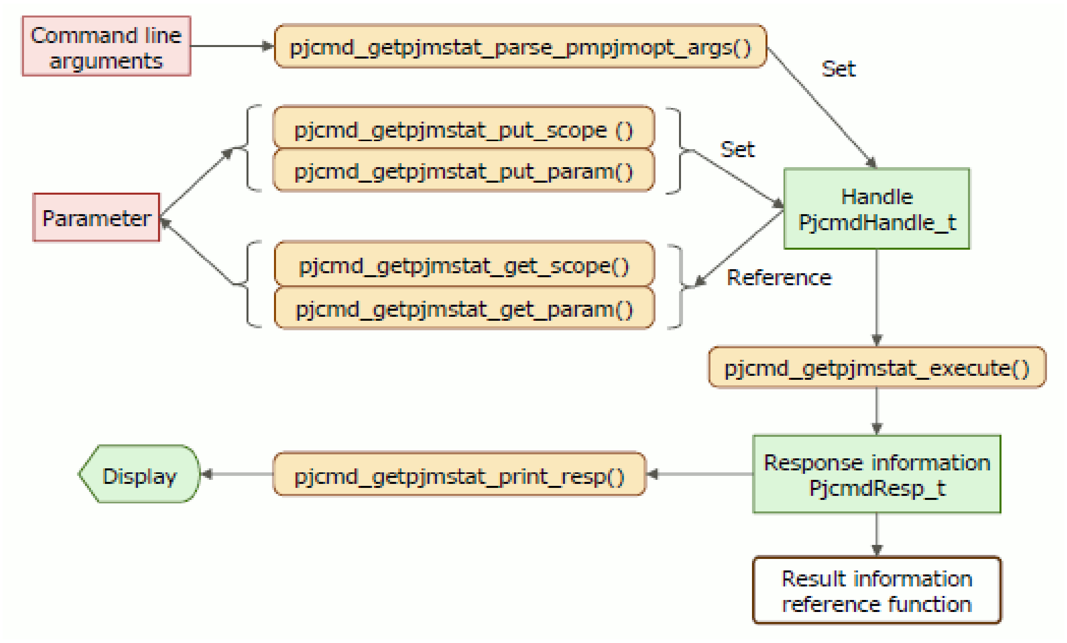
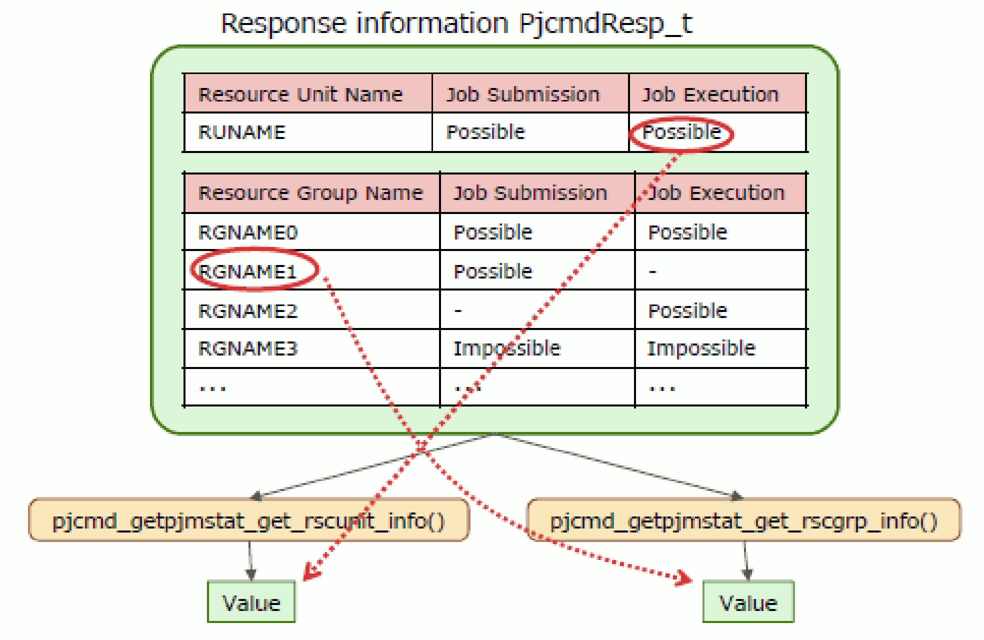

Command line arguments
pjcmd_getpjmstat_parse_pmpjmopt_args()
Set
pjcmd_getpjmstat_put_scope ()
pjcmd_getpjmstat_put_param()
Set
Parameter
Handle PjcmdHandle_t
pjcmd_getpjmstat_get_scope()
pjcmd_getpjmstat_get_param()
Reference
pjcmd_getpjmstat_execute()
Display
pjcmd_getpjmstat_print_resp()
Response information PjcmdResp_t
Result information reference function

Figure D.3 Referencing Job Submission and Execution Permission Information


Response information PjcmdResp_t
pjcmd_getpjmstat_get_rscunit_info()
pjcmd_getpjmstat_get_rscgrp_info()
Value
Value

Resource Unit Name	Job Submission	Job Execution
RUNAME	Possible	Possible

Resource Group Name	Job Submission	Job Execution
RGNAME0	Possible	Possible
RGNAME1	Possible	-
RGNAME2	-	Possible
RGNAME3	Impossible	Impossible
...	...	...

D.2.1 pjcmd_getpjmstat_parse_pmpjmopt_args()

```
pjcmd_result_t pjcmd_getpjmstat_parse_pmpjmopt_args(PjcmdHandle_t *handle_p, int argc, char
**argv_pp)
```

This function analyzes command line arguments based on the specification used when the --show-rsc-ug option in the pmpjmopt command is specified, and sets the specified details in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

argc

Number of arguments

argv_pp

Array of an argument

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting information on permissions for job submission and execution.

PJCMD_ERROR_INVALID_ARGUMENT

argc or *argv_pp* is invalid.

PJCMD_ERROR_UNKNOWN_OPTION

An unknown option has been detected.

PJCMD_ERROR_INVALID_OPTION

A method to specify an option is invalid.

- A method to specify an option argument is invalid.
- A required argument for the option is not specified.
- An exclusive option is specified.

Calling this function moves arguments other than options to the end of the *argv_pp*[] array.
If an unrecognizable option is detected, analysis of arguments stops, and *argv_pp*[pjcmd_optind-1] indicates the option.
This function recognizes only the --show-rsc-ug option in the pmpjmopt command and the options that can be specified with this option at the same time.

D.2.2 pjcmd_getpjmstat_put_scope()

```
pjcmd_result_t pjcmd_getpjmstat_put_scope(PjcmdHandle_t *handle_p, pjcmd_scope_t scope, const void
*val_p)
```

This function sets a target range to get the information on permissions for job submission and execution in a handle.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier indicating the target range (See the following table.)

val_p

Pointer to the storage area for the value indicating the target to get information. For example, if the value type to be set is char * type, the caller must prepare a storage area for the char * type value and specify a pointer (char **) to the area in *val_p*. If NULL is specified, the parameter value is initialized (not set).

scope	**val_p*	Type of **val_p*
PJCMD_SCOPE_CLUSTER	Cluster name (only one name, equivalent to pmpjmopt -c) If this parameter is not set, the PXMYCLST environment variable value is applied. If the parameter and the environment variable are not set, an error occurs in the pjcmd_getpjmstat_execute() function. This parameter is valid only when called from the system management node. When the parameter is called from a node other than the system management node, the cluster to which the calling node belongs is applied.	char *
PJCMD_SCOPE_RSCUNIT	Resource unit name (only one name, equivalent to pmpjmopt --rscunit) Specification of a resource unit name is required. If a resource unit name is not specified, an error occurs in the pjcmd_getpjmstat_execute() function.	char *
PJCMD_SCOPE_RSCGRP	Array of a resource group name (equivalent to pmpjmopt --rscgrp) The last element in the array must be (char *)NULL. "*" in a resource group name indicates all resource groups. If the resource group is not set, information on each resource unit is obtained.	char **

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting information on permissions for job submission and execution.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown or unspecifiable value is specified in *scope.*

PJCMD_ERROR_INVALID_PARAM

val_p is invalid (NULL).

PJCMD_ERROR_NOMEM

Memory acquisition failed.

D.2.3 pjcmd_getpjmstat_get_scope()

```
pjcmd_result_t pjcmd_getpjmstat_get_scope(const PjcmdHandle_t *handle_p, pjcmd_scope_t scope, void
*val_p)
```

This function references a target (cluster, resource unit, or resource group) that is set in a handle in order to get information on permissions for job submission and execution.

[ARGUMENTS]

handle_p

Pointer to a handle

scope

Identifier of the target range to be referenced. The identifiers that can be specified are the same as those for the pjcmd_getpjmstat_put_scope() function.

val_p

A value is stored in **val_p* based on the *scope* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting information on permissions for job submission and execution.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown or unspecifiable value is specified in *scope.*

PJCMD_ERROR_NODATA

A specified *scope* is not set in a handle.

D.2.4 pjcmd_getpjmstat_put_param()

```
pjcmd_result_t pjcmd_getpjmstat_put_param(PjcmdHandle_t *handle_p, pjcmd_getpjmstat_param_t param,
const void *val_p)
```

This function sets parameters in a handle that are related to acquisition of information on permissions for job submission and execution.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter related to acquisition of information on permissions for job submission and execution (See the following table.)

val_p

Pointer to the storage area for the value of parameter to be set. For example, if the value type to be set is int type, the caller must prepare a storage area for the int type value and specify a pointer (int *) to the area in *val_p.* If NULL is specified, the parameter value is initialized (not set).

param	**val_p*	Type of **val_p*
PJCMD_GETPJMSTAT_HELP	Equivalent to the --help option in the pmpjmopt command. 0: Not specified (Default) 1: Specified This parameter does not affect acquisition of information on permissions for job submission and execution.	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting information on permissions for job submission and execution.

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_INVALID_PARAM

A parameter value is invalid.

- A specification method is incorrect.
- A value is incorrect.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

D.2.5 pjcmd_getpjmstat_get_param()

```
pjcmd_result_t pjcmd_getpjmstat_get_param(const PjcmdHandle_t *handle_p, pjcmd_getpjmstat_param_t
param, void *val_p)
```

This function references the set parameters in a handle that are related to acquisition of information on permissions for job submission and execution.

[ARGUMENTS]

handle_p

Pointer to a handle

param

Identifier of a parameter to be referenced. The identifiers that can be specified are the same as those for the pjcmd_getpjmstat_put_param() function.

val_p

A value is stored in **val_p* based on the *param* type. The caller needs to prepare an area of a sufficient size according to the value type.

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting information on permissions for job submission and execution.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *param.*

PJCMD_ERROR_NODATA

A specified parameter is not set in a handle.

D.2.6 pjcmd_getpjmstat_execute()

```
PjcmdResp_t *pjcmd_getpjmstat_execute(const PjcmdHandle_t *handle_p)
```

This function requests the job operation management function to get the status of permissions for job submission and execution based on a handle. Root privileges are required for calling this function.

[ARGUMENTS]

handle_p

Pointer to a handle

[RETURN VALUE]

Response information about getting information on permissions for job submission and execution.
The caller must release the obtained response information by using pjcmd_destroy_resp(). If the request to permit job submission or execution fails, NULL is returned and pjcmd_errcode is set.
The response information indicates whether the request has succeeded or failed. The determination of whether the request to permit job submission or execution has been accepted successfully needs to be checked with a result code based on the response information by using the pjcmd_get_result() function.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Handle is invalid.

- *handle_p* is NULL.
- This is not a handle for getting information on permissions for job submission and execution.

PJCMD_ERROR_INVALID_NODE

This function cannot be called from this node. The function can only be called from the system management node.

PJCMD_ERROR_INVALID_PARAM

A parameter in a handle is invalid.

PJCMD_ERROR_NOMEM

Memory acquisition failed.

PJCMD_ERROR_NOPERM

Calling the function is not permitted.

PJCMD_ERROR_INTERNAL

Internal error

D.2.7 pjcmd_getpjmstat_print_resp()

```
pjcmd_result_t pjcmd_getpjmstat_print_resp(const PjcmdResp_t *resp_p)
```

This function outputs the results of getting job submission and execution permission information to the standard output based on the pmpjmopt command specifications.

[ARGUMENTS]

resp_p

Pointer to a response information

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response information is invalid.

- *resp_p* is NULL.
- This is not a response information for getting information on permissions for job submission and execution.
- This is not response information that was successfully obtained.

D.2.8 pjcmd_getpjmstat_get_rscunit_info()

```
pjcmd_result_t pjcmd_getpjmstat_get_rscunit_info(const PjcmdResp_t *resp_p, pjcmd_getpjmstat_info_t
type, void *val_p)
```

This function references information on a resource unit from the response information about getting job submission and execution permission information.

[ARGUMENTS]

resp_p

Pointer to a response information

type

Identifier of an information type to be referenced (See the following table.)

val_p

A value is stored in **val_p* based on the *type* type. The caller needs to prepare an area of a sufficient size according to the value type.

type	**val_p*	Type of **val_p*
PJCMD_GETPJMSTAT_INFO_RSCUNIT	Resource unit name The area specified by **val_p*is is undetermined after releasing response information.	char *

type	**val_p*	Type of **val_p*
PJCMD_GETPJMSTAT_INFO_JOB_SUBMIT	Job submission permission 0: Not permitted to submit new job 1: Permitted to submit new job -1: Unknown	int
PJCMD_GETPJMSTAT_INFO_JOB_EXECUTE	Job execution permission 0: Not permitted to execute new job 1: Permitted to execute new job -1: Unknown	int
PJCMD_GETPJMSTAT_INFO_RSCGRP_NUM	Number of resource groups in a resource unit If the target for getting information is a resource unit, 0 is set.	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_HANDLE

Response information is invalid.

- *resp_p* is NULL.
- This is not a response information for getting information on permissions for job submission and execution.
- The response information is not a result related to a resource unit but rather a result related to a resource group.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *type.*

PJCMD_ERROR_INVALID_PARAM

The value of *type* cannot be specified in this function.

D.2.9 pjcmd_getpjmstat_get_rscgrp_info()

```
pjcmd_result_t pjcmd_getpjmstat_get_rscgrp_info(const PjcmdResp_t *resp_p, int indx,
pjcmd_getpjmstat_info_t info, void *val_p)
```

This function references specific resource group information from response information about getting job submission and execution permission information.

[ARGUMENTS]

resp_p

Pointer to a response information

indx

Index of the resource group to be obtained.
The specifiable value ranges from 0 to a value that is calculated by subtracting 1 from the number of resource groups that is obtained by the pjcmd_getpjmstat_get_rscunit_info() function.

info

Identifier of information to be referenced (See the following table.)

val_p

A value is stored in * *val_p* based on the *info* type. The caller needs to prepare an area of a sufficient size according to the value type.

info	**val_p*	Type of **val_p*
PJCMD_GETPJMSTAT_INFO_RSCGRP	Resource group name	char *
PJCMD_GETPJMSTAT_INFO_JOB_SUBMIT	Job submission permission 0: Not permitted to submit new job 1: Permitted to submit new job -1: Unknown	int
PJCMD_GETPJMSTAT_INFO_JOB_EXECUTE	Job execution permission 0: Not permitted to execute new job 1: Permitted to execute new job -1: Unknown	int

[RETURN VALUE]

PJCMD_OK

Success

PJCMD_ERR

Failure. The cause is set in pjcmd_errcode.

[pjcmd_errcode]

PJCMD_ERROR_INVALID_RESP

Response inforamtion is invalid.

- *resp_p* is NULL.
- This is not a response information for getting information on permissions for job submission and execution.
- This result is not from getting the resource group status but rather from getting the resource unit status.

PJCMD_ERROR_INVALID_ARGUMENT

val_p is invalid (NULL).

PJCMD_ERROR_UNKNOWN_PARAM

An unknown value is specified in *info.*

PJCMD_ERROR_INVALID_PARAM

The value of *info* cannot be specified in this function.

PJCMD_ERROR_NODATA

The value of *indx* is out of range.

Appendix E Sample Programs

This appendix provides sample programs that call the command API.

The source files of the sample programs are in the following installation directory of the login node, compute cluster management node, and system management node.

```
/usr/src/FJSVtcs/pjm/pjcmd/
```

For details on how to compile a sample program, see documents in the above directory.

E.1 Submitting a Job

/usr/src/FJSVtcs/pjm/pjcmd/c/submit/pjcmd_submit.c

```
/*
 * Submit job
 */

#include <stdio.h>
#include <stdlib.h>
#include <FJSVtcs/pjm/pjcmd.h>

#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1

const char *CMD_NAME = "pjsub_custom";

int main(int argc, char **argv)
{
    PjcmdHandle_t *handle_p;
    PjcmdResp_t *resp_p;
    char *script_p;
    int i, code;
    int32_t line;
    char *detail_p;
    PjcmdSubjobid_t *subjobid_p;
    char subjobid_str[PJCMD_MAX_SUBJOBID_STR_LEN];

    /* Create handle to submit job */
    handle_p = pjcmd_create_handle(PJCMD_SUBMIT);
    if (handle_p == NULL) {
        fprintf(stderr,
                "%s: Failed in create_handle : %s\n", CMD_NAME, pjcmd_strerror(pjcmd_errcode));
        exit(EXIT_FAILURE);
    }

    /* Analyze command line arguments and set them in handle */
    if (pjcmd_submit_parse_pjsub_args(handle_p, argc, argv) == PJCMD_ERR) {
        /* Terminate after displaying arguments that failed to be analyzed */
        fprintf(stderr, "%s: Failed in parse_args : %s\n", CMD_NAME, argv[pjcmd_optind - 1]);
        exit(EXIT_FAILURE);
    }

    /* Analyze instruction lines in job script and set them in handle
     * pjcmd_optind indicates remaining arguments = scripts
     * (For simplicity, only 1 script used)
     */
    if (pjcmd_optind == argc) {
        fprintf(stderr, "%s: job script does not specified\n", CMD_NAME);
        pjcmd_destroy_handle(handle_p);
        exit(EXIT_FAILURE);
    }

    script_p = argv[pjcmd_optind];
    if (pjcmd_submit_parse_pjsub_scriptfile(handle_p, script_p, "#PJM", &line, &detail_p)
         == PJCMD_ERR) {
        fprintf(stderr, "%s: Failed to parse script %s (line=%d, arg=%s)\n",
                CMD_NAME, script_p, line, detail_p);
        pjcmd_destroy_handle(handle_p);
        exit(EXIT_FAILURE);
    }

    /* Set script file name in handle */
    if (pjcmd_submit_put_param(handle_p, PJCMD_SUBMIT_SCRIPTFILE, &script_p) == PJCMD_ERR) {
        fprintf(stderr, "%s: Failed in setting script name : %s\n", CMD_NAME, script_p);
        pjcmd_destroy_handle(handle_p);
        exit(EXIT_FAILURE);
    }

    /* Use set information in handle and submit job */
    resp_p = pjcmd_submit_execute(handle_p);
    if (resp_p == NULL) { /* If job submission failed */
        fprintf(stderr, "%s: Failed in submitting a job : %s\n", CMD_NAME, script_p);
        pjcmd_destroy_handle(handle_p);
        exit(EXIT_FAILURE);
    }

    /* Display submission results */
    int64_t jobnum[2];
    pjcmd_get_jobresult_num(resp_p, jobnum);
    for (i = 0; i < jobnum[0]; i++) {
        /* Obtain sub job ID structure and convert it to character string */
        pjcmd_get_jobresult_info(resp_p, PJCMD_JOBRESULT_ANY,
                         i, PJCMD_JOBRESULT_SUBJOBID, &subjobid_p);
        pjcmd_subjobid_to_str(subjobid_p, subjobid_str);
        /* Obtain result code */
        pjcmd_get_jobresult_info(resp_p, PJCMD_JOBRESULT_ANY, i, PJCMD_JOBRESULT_CODE, &code);
        /* Display result */
        printf("Job %s : %s\n", subjobid_str, (code == 0) ? "submitted" : "submit failed");
    }
    /* Release response information */
    pjcmd_destroy_resp(resp_p);
    /* Release handle */
    pjcmd_destroy_handle(handle_p);
    exit(EXIT_SUCCESS);
}
```

E.2 Getting Job Information

/usr/src/FJSVtcs/pjm/pjcmd/c/jobinfo/pjcmd_jobinfo.c

```
/*
 * Display job status
 */

#include <stdio.h>
#include <stdlib.h>
#include <FJSVtcs/pjm/pjcmd.h>

#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1

const char *CMD_NAME = "pjstat_custom";
```

```
int main(void)
{
    PjcmdHandle_t *handle_p;
    PjcmdResp_t *resp_p;

    /* Create handle for getting job information */
    handle_p = pjcmd_create_handle(PJCMD_JOBINFO);
    if (handle_p == NULL) {
        fprintf(stderr,
                "%s: Failed to create handle : %s\n", CMD_NAME, pjcmd_strerror(pjcmd_errcode));
        exit(EXIT_FAILURE);
    }

    /* Get information calculated in source unit or resource group unit */
    const char *grouping_items_p[2] = { "rscu", "rscg" };
    pjcmd_jobinfo_put_condition(handle_p, PJCMD_JOBINFO_GROUPING, grouping_items_p, 2);

    /* Set conditions as follows for job to be obtained:
     *  - Status is RUN (pjstat --filter st=RUN)
     *  - Job name begins with foo (pjstat --filter "jname=foo*")
     *  - Job ID is 10 characters or less (pjstat --filter jid=-10)
     *  - Priority is 5 or higher (pjstat --filter prio=5-)
     *  - elapse limit is 1 to 2 hours
     *    (pjstat --filter elpl=1:00:00-2:00:00)
     */
    const char *filter_exprs_p[5] = { "st=RUN", "jnam=foo*", "jid=-10", "prio=5-",
                                          "elpl=1:00:00-2:00:00" };
    pjcmd_jobinfo_put_condition(handle_p, PJCMD_JOBINFO_FILTER, filter_exprs_p, 5);

    /* Set job ID, job name, and job status as items to be obtained
     * (pjstat --choose jid,jnam,st)
     */
    const char *choose_items_p[3] = { "jid", "jnam", "st" };
    pjcmd_jobinfo_put_condition(handle_p, PJCMD_JOBINFO_CHOOSE, choose_items_p, 3);

    /* Sort jobs in ascending order of job submission time */
    const char *sort_items_p[] = { "adt:A" };
    pjcmd_jobinfo_put_condition(handle_p, PJCMD_JOBINFO_SORT, sort_items_p, 1);

    /* Get history (completed jobs)
     * information for past 5 days (pjstat -H=day)
     */
    int hist = 5;
    pjcmd_jobinfo_put_param(handle_p, PJCMD_JOBINFO_HISTORY_DAY, &hist);

    /* Get summary information altogether (equivalent to pjstat --with-summary) */
    int summary = PJCMD_JOBINFO_WITH_SUMMARY;
    pjcmd_jobinfo_put_param(handle_p, PJCMD_JOBINFO_SUMMARY, &summary);

    /* Get sub job information altogether (equivalent to pjstat -E) */
    int verbose = PJCMD_JOBINFO_VERBOSITY_SUBJOB;
    pjcmd_jobinfo_put_param(handle_p, PJCMD_JOBINFO_VERBOSITY, &verbose);

    /*  Get other users' job information altogether (equivalent to pjstat -A)
     *  (If command is called with general user privileges,
     *   items that cannot be referenced due to ACL settings are masked)
     */
    int othersjob = PJCMD_JOBINFO_OTHERSJOB_ALL;
    pjcmd_jobinfo_put_param(handle_p, PJCMD_JOBINFO_OTHERSJOB, &othersjob);

    /* Request acquisition of job information based on contents of handle */
    resp_p = pjcmd_jobinfo_execute(handle_p);
    if (resp_p == NULL) {
        fprintf(stderr, "%s: Request failed\n", CMD_NAME);
        pjcmd_destroy_handle(handle_p);
        exit(EXIT_FAILURE);
    }
    int code;
    int subcode;
    char *detail_p;
    pjcmd_get_result(resp_p, &code, &subcode, &detail_p);
    if (code != 0) {
        fprintf(stderr, "%s: Request failed (code=%d)\n", CMD_NAME, code);
        pjcmd_destroy_handle(handle_p);
        exit(EXIT_FAILURE);
    }

    /* Read information groups one by one from resp_p response information
     * (Job information group is read in unit of resource unit/resource group
     * based on acquisition conditions)
     */
    pjcmd_result_t ret;
    while (1) {
        ret = pjcmd_jobinfo_read_infogrp(resp_p);
        if (ret == PJCMD_ERR) {
            if (pjcmd_errcode == PJCMD_ERROR_NODATA) {
                break; /* All information groups are read */
            }
            fprintf(stderr, "%s: Cannot read infogrp\n", CMD_NAME);
            pjcmd_destroy_resp(resp_p);
            pjcmd_destroy_handle(handle_p);
            exit(EXIT_FAILURE);
        }

        /* Display summary line of information group */
        pjcmd_jobinfo_print_resp(resp_p, PJCMD_JOBINFO_PRINT_SUMMARY);
        /* Display job information in information group */
        pjcmd_jobinfo_print_resp(resp_p, PJCMD_JOBINFO_PRINT_JOBINFO);
    }

    pjcmd_destroy_resp(resp_p);
    pjcmd_destroy_handle(handle_p);
    exit(EXIT_SUCCESS);
}
```

E.3 Deleting a Job

/usr/src/FJSVtcs/pjm/pjcmd/c/kill/pjcmd_kill.c

```
/*
 * Delete job
 */

#include <stdio.h>
#include <stdlib.h>
#include <FJSVtcs/pjm/pjcmd.h>

#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1

const char *CMD_NAME = "pjdel_custom";

int main(int argc, char **argv)
{
    PjcmdHandle_t *handle_p;
    PjcmdResp_t *resp_p;
    PjcmdSubjobid_t *subjobid_p;
    char subjobid_str_p[PJCMD_MAX_SUBJOBID_STR_LEN];
    int i;
    int64_t num[2], total_num, ok_num, err_num, cnt;
    int help_flag;

    /* Create handle to delete job */
    handle_p = pjcmd_create_handle(PJCMD_KILL);
    if (handle_p == NULL) {
        fprintf(stderr, "Failed in create_handle : %s\n", pjcmd_strerror(pjcmd_errcode));
        exit(EXIT_FAILURE);
    }

    /* Analyze command line arguments */
    if (pjcmd_kill_parse_pjdel_args(handle_p, argc, argv) == PJCMD_ERR) {
        fprintf(stderr, "Failed in parse_args : %s : arg=%s\n",
                        pjcmd_strerror(pjcmd_errcode), argv[pjcmd_optind - 1]);
        /* Display the usage of pjdel */
        pjcmd_print_stdcmd_usage(PJCMD_STDCMD_PJDEL, CMD_NAME);
        pjcmd_destroy_handle(handle_p);
        exit(EXIT_FAILURE);
    }

    /* If --help option specified, exit after displaying method of use */
    pjcmd_kill_get_param(handle_p, PJCMD_KILL_HELP, &help_flag);
    if (help_flag != 0) {
        /* Display method of using pjdel */
        pjcmd_print_stdcmd_usage(PJCMD_STDCMD_PJDEL, CMD_NAME);
        pjcmd_destroy_handle(handle_p);
        exit(EXIT_SUCCESS);
    }

    /* Treat remaining arguments as job IDs of jobs to be deleted and set them in handle */
    if (pjcmd_optind == argc) { /* No job ID */
        fprintf(stderr, "Job id is not spesified.\n");
        pjcmd_destroy_handle(handle_p);
        exit(EXIT_FAILURE);
    }
    for (i = pjcmd_optind; i < argc; i++) {
        if (pjcmd_put_job_by_str(handle_p, argv[i]) == PJCMD_ERR) {
            fprintf(stderr, "Failed in put_job_by_str : %s\n", argv[i]);
            pjcmd_destroy_handle(handle_p);
            exit(EXIT_FAILURE);
        }
    }

    /* Request job deletion */
    resp_p = pjcmd_kill_execute(handle_p);
    if (resp_p == NULL) { /* If request fails */
        fprintf(stderr, "Failed in kill_execute : %s\n", pjcmd_strerror(pjcmd_errcode));
        pjcmd_destroy_handle(handle_p);
        exit(EXIT_FAILURE);
    }

    /* Display deletion result */
    pjcmd_get_jobresult_num(resp_p, num);
    total_num = num[0];
    ok_num    = num[1];
    err_num   = total_num - ok_num;

    if (err_num) { /* For error jobs */
```

```
        fprintf(stderr, "Operation failed for %ld jobs.\n", err_num);
    }
    /* Display only job IDs (sub job IDs) of failed jobs, one item per line */
    for (cnt = 0; cnt < err_num; cnt++) {
        pjcmd_get_jobresult_info(resp_p, PJCMD_JOBRESULT_ERR,
                                 cnt, PJCMD_JOBRESULT_SUBJOBID, &subjobid_p);
        pjcmd_subjobid_to_str(subjobid_p, subjobid_str_p);
        fprintf(stderr, "Failed for the job %s\n", subjobid_str_p);
    }

    pjcmd_destroy_resp(resp_p);
    pjcmd_destroy_handle(handle_p);
    exit((err_num == 0) ? EXIT_SUCCESS : EXIT_FAILURE);
}
```