FUJITSU

FUJITSU Software
Technical Computing Suite V4.0L20

Job Operation Software
API user's Guide for Scheduler API

J2UL-2463-02ENZ0(01)
June 2020

Preface

Purpose of This Manual

This document describes the scheduler plugin function (scheduler API), which is a part of the job operation management function of the "Job Operation Software" included in Technical Computing Suite.

Intended Readers

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

The manual assumes that readers have the following knowledge:

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

Organization of This Manual

This manual is organized as follows.

Notation Used in This Manual

Notation of model names

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

Administrators

The Job Operation Software has different types of administrators: system administrator, cluster administrator, and job operation administrator. However, they may all be represented as just "administrator" in this document. In such cases, an administrator who manages the system usually means the system administrator or cluster administrator. An administrator who manages job operations means the cluster administrator or job operation administrator.

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.
- All other trademarks are the property of their respective owners.

Date of Publication and Version

Version	Manual code
June 2020, Version 2.1	J2UL-2463-02ENZ0(01)
March 2020, Second version	J2UL-2463-02ENZ0(00)
January 2020, First version	J2UL-2463-01ENZ0(00)

Copyright

Update history

Changes	Location	Version
Fixed errata.	-	2.1
Added the member pjsplg_net_route_t net_route to the structure PjsplgJobSubmitParam_t that represents the job submission parameters. Also added the enumeration type pjsplg_net_route_t.	C.1.6 C.3.5	2
Changed the look according to product upgrades.	-	

Contents

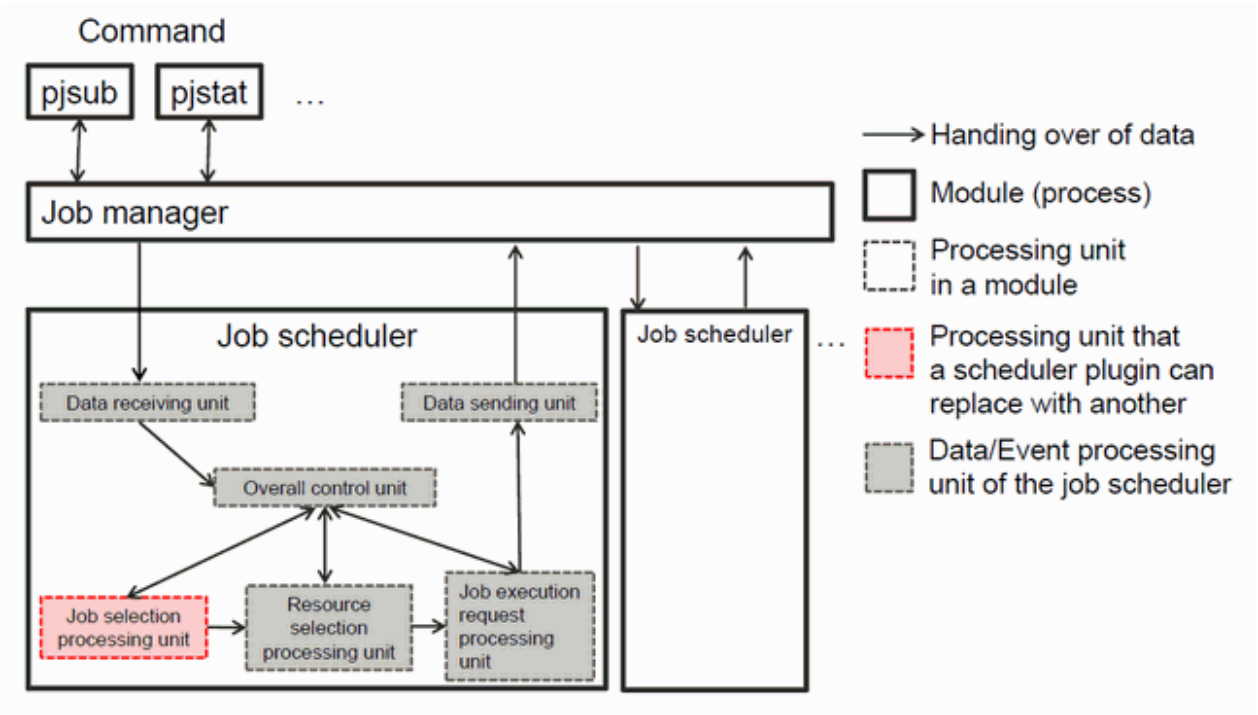
Chapter 1 Overview of Scheduler Plugin Function

The Job Operation Software provides a function that incorporates an original scheduling algorithm created by the job operation administrator into the job scheduler to replace the scheduling algorithm of the job operation management function with it. This function is called the "scheduler plugin function." Use of the scheduler plugin function allows you to apply an optimal scheduling algorithm to the job operation.

1.1 Scheduler Plugin Function

An original scheduling algorithm can be incorporated by replacing a part of the processing unit of the job scheduler with a module (shared library) created by the job operation administrator. The following figure shows the internal structure of the job scheduler and relations with the modules (job manager and commands).

Figure 1.1 Internal Structure of the Job Scheduler and the Processing Unit that a Plugin Can Replace With Another



The job scheduler is a module that allocates resources according to the job priority and determines the order of executing jobs (scheduled time of job execution start). It operates in coordination with the job manager that has interfaces with commands. The job scheduler consists of the following processing units.

- Data receiving unit
  A processing unit that receives data sent from the job manager. The types of received data include those indicating job submission, change in the node state, change in the system setting, etc. The received data is handed over to the overall control unit.

- Overall control unit
  A processing unit that controls the start and end of job scheduling. Triggered by data received from another processing unit, it discontinues the scheduling being performed or starts new scheduling. When discontinuing scheduling being performed, it retrieves the job being processed in each processing unit.

- Job selection processing unit
  A processing unit that prioritizes submitted jobs according to the job selection policy, and sorts them in descending order of the priority levels. It hands the jobs to the resource selection processing unit in sequence beginning with the one with the highest priority level. This processing unit and the resource selection processing unit cooperate with each other to implement the job scheduler function.

- Resource selection processing unit
  This unit, beginning with the job with the highest priority level determined by the job selection processing unit, allocates resources dedicated for each job (compute nodes, custom resources, etc.) while being aware of the number of requested nodes, elapse time limit

value, etc., and determines the scheduled time of job execution start. It hands over a job for which resources have been selected to the job execution request processing unit. This processing unit and the job selection processing unit cooperate with each other to implement the job scheduler function.

- Job execution request processing unit
  This unit requests the job manager to execute a job when it is the scheduled execution start time of the job determined by the resource selection processing unit.

The scheduler plugin function provides an API (Application Programming Interface) to replace the job selection processing unit, which is one of the above processing units, with a unique scheduling algorithm. This API is called the "scheduler API."

By using this API, the job operation administrator implements an original scheduling algorithm as a shared library, and incorporates this library into the job scheduler. Such a unique scheduling algorithm is incorporated in the job scheduler. The algorithm and a series of programs implementing the algorithm are together called the "job selection plugin." Also, shared libraries incorporated into the scheduler are called "plugin libraries." A process of the job scheduler is started for each plugin library.

With this library incorporated in the job scheduler, the job selection plugin determines the priority of jobs in the same way as the job selection processing unit of the job scheduler. The job scheduler controls the priority of jobs by preferentially allocating compute resources in descending order of job priority determined by the job selection plugin.

1.2 Programming Model of the Plugin Library

The scheduler API is designed by using an object-oriented programming model based on the class. The programming model for the plugin library has the concept called "class" which is implemented by the plugin library and "instance" which is generated based on the class.

- Class
  A class is a series of functions defined by a plugin library for the purpose of replacing specific processing of the job scheduler.
  The class is registered in the job scheduler during the initialization processing of a plugin.
  When the job scheduler calls the series of functions defined in the class during scheduling processing, the default scheduling algorithm is replaced with the original scheduling algorithm implemented by the plugin library. The class for replacing the scheduling algorithm is called a "job selection class."

- Instance
  An instance is a data structure of an arbitrary type. It is created for each resource unit or resource group that is a processing target of a plugin library, and holds data necessary for processing defined by the class.
  An instance is generated when the job scheduler calls the initialization function of the job selection class as defined in the class. (The function is create(). For details, see "Table 1.2 Functions Implemented by the Plugin Library (related to the job selection class).")
  Data held in an instance is used in the plugin library.
  For the job selection class, one instance is generated for each job scheduler process or each resource group.

See

A plugin library must be implemented according to this programming model. For information on how to implement a plugin library, see "2.1 How to Create a Plugin Library" and "2.2 Plugin Library Incorporation Settings."

1.3 Constitution of the Scheduler API

The scheduler API consists of the "plugin library" described in the preceding section and "built-in functions" provided for the plugin library by the job scheduler.
This section provides lists of the functions and classes that the plugin library must implement and the built-in functions that the plugin can use.

1.3.1 Functions That the Plugin Library Must Implement

The tables below provide lists of the functions that the plugin library must implement. To use the scheduler plugin function, the following functions are all required.

Table 1.1 Functions That the Plugin Library Implements (related to the job selection plugin)

Function Name	Description
plugin_init()	Initialization function of the job selection plugin This is called when the plugin library is read (at the time of dynamic loading). When called by the job scheduler, the initialization function of the job selection plugin registers a class for the job scheduler. The class registered here is called during the scheduling of the job scheduler.
plugin_fini()	End function of the job selection plugin This is called when the plugin library is released (unloaded). When called by the job scheduler, the end function of the job selection plugin deletes the registration of the class for the job scheduler.

Table 1.2 Functions Implemented by the Plugin Library (related to the job selection class)

Function Name	Description
create()	Initialization function of the job selection class This function is called at the start of a job scheduler process. The function builds the data (instance) required for the processing defined in the job selection class. The instance returned as the return value of this function is passed as an argument when the destroy(), push_many(), pop(), or remove_all() function is called.
destroy()	End function of the job selection class This function is called at the end of a job scheduler process. The function releases the data kept in an instance of the job selection class.
push_many()	Job registration function This function is called by the job scheduler when scheduling begins. The function registers jobs to be scheduled and passes them as arguments so that they can be managed by the job selection plugin. (*)
pop()	Function to retrieve the highest-priority job This function is called by the job scheduler during scheduling. The function returns the job that will select compute resources next (highest-priority job among jobs registered in the instance). The job is removed from the jobs managed by the job selection plugin and excluded from the jobs returned in subsequent pop() calls.
remove_all()	Function to remove registered jobs This function is called by the job scheduler when scheduling ends or is interrupted. The function removes all jobs from the jobs managed by the job selection plugin.

(*)
The following job information for scheduled jobs is passed as arguments of the push_many() function:

- Sub job ID
- Execution user ID
- Acceptance time (pjsub command execution time)
- Acceptance time of the first sub job (only for a step job)
- Job type (batch job or interactive job)
- Job model
- Execution group ID
- Job priority specified by the user
- Job priority within the resource unit
- Specified execution start time (value specified in the -at option of the pjsub command)
- Resource group ID

- Elapsed time limit value for job execution
- Total of the elapsed time limit values of preceding sub jobs (only for a step job)
  For the first sub job, 0 is passed. Note that the total does not include the elapsed time limit values of preceding sub jobs that ended when the step job was submitted by the pjsub command.
- Requested node shape (including information on whether :strict or :strict-io is specified in the -L node option of the pjsub command)
- Requested memory amount per node
- Priority of the job execution user
- Priority of the job execution group
- User priority within the group of the job execution user
- Job status (QUEUED only)
- Job execution wait time (difference between the job submission and scheduling start times)
- Re-execution count (cumulative count of job re-execution due to compute node failure, etc.)

See

For details of the functions, see "Appendix G Reference: Data Structures Relevant to the Job Selection Class" and "Appendix I Reference: Constants and Functions of the Job Selection Plugin."

1.3.2 Built-in Functions Provided by the Job Scheduler

The following provides a list of the built-in functions provided for the plugin library by the job scheduler.

Table 1.3 Built-in Functions Provided by the Job Scheduler

Name	Description
register_job_select_class()	Registers a job selection class.
deregister_job_select_class()	Deletes a job selection class.
get_rscunit()	Obtains the access right for resource unit information. The resource unit information includes also resource group information and node information. (*1)
put_rscunit()	Returns the access right for resource unit information.
get_user_fshare_value()	Obtains the fair share value of a specified user. (*2)
get_group_fshare_value()	Obtains the fair share value of a specified group. (*2)
get_user_in_group_fshare_value()	Obtains the user fair share value in the group of a specified user. (*2)

(*1)
The job selection plugin has the right to access resource unit information, whereas other threads of the job scheduler cannot access the information. Consequently, resource unit setting changes are not processed. After the access right is returned by the put_rscunit() function, setting changes are processed, and resource unit information is changed.
(*2)
The get_user_fshare_value(), get_group_fshare_value(), or get_user_in_group_fshare_value() function obtains a fair share value at the point in time when the function is called. The update timing of fair share values is, for example, the job execution start time.

See

For details of the built-in functions provided for the plugin library by the job scheduler, see "Appendix F Reference: API Functions That Can be Used From the Job Selection Class."

1.4 Scheduler API Types

The scheduler plugin function provides the following scheduler APIs.

- Types and definitions common to the scheduler APIs
- APIs relevant to job information
- APIs relevant to resource group information
- APIs relevant to resource unit information
- APIs relevant to the job selection plugin
- APIs relevant to the job selection plugin

See

For details of scheduler APIs, see the appendices from "Appendix B Reference: Common to the Scheduler APIs" to "Appendix I Reference: Constants and Functions of the Job Selection Plugin."

Chapter 2 Use of Scheduler Plugin Functions

This chapter describes how to use scheduler plugin functions.

2.1 How to Create a Plugin Library

The job selection plugin is implemented as a shared library (plugin library). The plugin library is loaded into the job scheduler process, and the functions implemented in the plugin library are called to allow scheduling by its own algorithm.

This section describes rules and knowledge necessary for creating a plugin library.

2.1.1 Rules for Creating a Plugin Library

Create a plugin library in compliance with the following rules.

- Name a file as follows.

```
lib plugin library name.so
```

  Specify the plugin name with a character string made up of single-byte alpha-numeric characters, hyphens ("-"), or underscores ("_").

Note

There is no limit on the number of the characters used for the file name of a plugin library. However, there is the limit of 512 characters in one line regarding the pmpjm.conf file, in which setting of incorporation of a plugin library in the job scheduler is made. While taking this limit into account, name the file.
For information on setting of incorporation of a plugin library in the job scheduler, see "2.2 Plugin Library Incorporation Settings."

- Use a share object (.so format) of ELF (Executable and Linkable Format) as a file format.
  The plugin library must be read when the job scheduler calls dlopen(3) (dynamic loading). At this time, libraries dynamically linked with the plugin library must be read even if the environment variable LD_LIBRARY_PATH is not specified. Especially, if the plugin library depends on a shared library placed in a non-standard path (path that is not /usr/lib or /usr/lib64, etc), you need to take some measures so that the path can be resolved. For example, you can specify RUNPATH when creating the plugin, or you can change the system setting by adding the path to /etc/ld.so.conf in the compute cluster management node, in which the job scheduler runs.

  The following is a compilation example.

```
# gcc -shared -fPIC -lpthread -lrt -o lib plugin library name.so source file
```

- Use C or a language supporting a C-compatible interface as a programming language.
  Since many jobs must be processed at a high speed, we recommend that the plugin library be created in a language such as C, a language that enables native compilation.

- To implement a scheduling algorithm in a plugin library, use the APIs provided by the job scheduler defined in the following header file, which is found in the system management node, compute cluster management node, and login node.

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

- The macro PJSPLG_DEFINE must be called.
  By calling this macro, the job scheduler is notified of the plugin name, plugin version, and version of the scheduler API used by the plugin library. This causes necessary information to be incorporated in the library, and the job scheduler becomes able to read the information. The job scheduler reads the information that has been read in the plugin library and uses it for checking the compatibility of the scheduler API and for the identifiers of log messages relevant to the plugin.

See

For details of the macro PJSPLG_DEFINE, see "Appendix I Reference: Constants and Functions of the Job Selection Plugin."

- The plugin library must implement the following functions and methods.
  - Initialization function of the job selection plugin plugin_init()
  - End function of the job selection plugin plugin_fini()
  - Initialization function of the job selection class create()
  - End function of the job selection class destroy()
  - Job registration function push_many()
  - Function to retrieve the highest-priority job pop()
  - Function to remove registered jobs remove_all()

Note

When all jobs are sorted at the push_many() function call time, job priority cannot change according to the fair share value, which changes based on resource allocation to the jobs retrieved by the pop() function. Also, since the start of resource selection processing waits until the sorting of all the jobs is completed, job sorting cannot be processed in parallel with the resource selection processing. Therefore, we recommend an implementation that searches for only the highest-priority job at the point in time when the pop() function is called.

See

For details of the functions the plugin library must implement, see "1.3.1 Functions That the Plugin Library Must Implement," "Appendix G Reference: Data Structures Relevant to the Job Selection Class," and "Appendix I Reference: Constants and Functions of the Job Selection Plugin."

- As to the functions to be implemented by the plugin library, make them thread-safe.
- Place the created plugin library in the compute cluster management node.

2.1.2 Implementation Example of the Plugin Library

The following is an implementation example of the initialization and end processing of the plugin library.

```
// include header file to declare API
#include <FJSVtcs/pjm/pjsplugin.h>

// Declare plugin name and version (displayed in pjsd log)
PJSPLG_DEFINE("custom-jobselect", "1.0.0");

// Declare group of functions (class) called from scheduler
const PjsplgJobSelectClass_t JOB_SELECT_CLASS = {
  .create     = create,     .destroy = destroy,
  .push_many  = push_many,  .pop     = pop,
  .remove_all = remove_all
};

// Function called when plugin is loaded
pjsplg_result_t plugin_init(PjsplgManager_t *mng_p) {
  // Register group of functions (class) related to job selection
  mng_p->register_job_select_class(&JOB_SELECT_CLASS);
}

// Function called when plugin is released
pjsplg_result_t plugin_fini(PjsplgManager_t *mng_p) {
  // Remove group of registered functions
  mng_p->deregister_job_select_class(&JOB_SELECT_CLASS);
}
```

```
// Declare internal data structure used inside plugin
typedef struct { ... } InternalData_t;

// Implement create function
static void *create(const PjsplgJobSelectPolicy_t *policy_p) {
  // Get data area used internally
  InternalData_t *data_p = calloc(1, sizeof(InternalData_t));

  // Initialize and return data used internally
  ...
  return data_p;
}

// Implement destroy function.
static void destroy (void *instance_p) {
  InternalData_t *data_p = (InternalData_t *)instance_p;

  // Release data area used internally
  ...
}

// Implement push_many function
static pjsplg_result_t push_many(void *instance_p,
                                 const PjsplgJobSelectClassApi_t *api_p,
                                 PjsplgSubjob_t **jobs_list_pp) {
  InternalData_t *data_p = (InternalData_t *)instance_p;
  // Register job information passed to internal data
  ...
  return PJSPLG_OK;
}

// Implement pop function
static pjsplg_result_t pop(void *instance_p,
                           const PjsplgJobSelectClassApi_t *api_p,
                           PjsplgSubjob_t **job_pp) {
  InternalData_t *data_p = (InternalData_t *)instance_p;

  if (resources targeted for any job) {
     // If no job targeted for resource selection, return NULL
     *job_pp = NULL;
     return PJSPLG_OK;
  }
  // Sort jobs in internal data in order of priority
  ...
  // Retrieve highest-priority job
  ...
  // Remove retrieved job from internal data
  ...
  return PJSPLG_OK;
}

// Implement remove_all function
static pjsplg_result_t remove_all(void *instance_p,
                                  const PjsplgJobSelectClassApi_t *api_p) {
  InternalData_t *data_p = (InternalData_t *)instance_p;

  // Remove all jobs from internal data
  ...

  return PJSPLG_OK;
}
```

2.1.3 Operation when Abnormality is Detected in the Plugin Library

The job scheduler operates as follows upon detection of abnormality caused by an error, etc. in the plugin library when loading or unloading the plugin library.

[Event 1]

Read error of the plugin library

[Cause]

- A call of dlopen(3) of the plugin library failed.
- Version incompatibility of the scheduler API was detected.
- A symbol necessary for the plugin library has not been defined.

[Operation]

The process of the job scheduler discontinues the start processing and terminates abnormally.

[Event 2]

Error return of the initialization function of the job selection plugin
Error return of the Initialization function of the job selection class

[Cause]

Error in the plugin library

[Operation]

The process of the job scheduler discontinues the start processing and terminates abnormally.

[Event 3]

Error return of the end function of the job selection class
Error return of the end function of the job selection plugin

[Cause]

Error in the plugin library

[Operation]

Since instance release or plugin end processing is performed at the end of the job scheduler process, a log message is output and the job scheduler process ends normally.

 See

For details of the processing performed at the start and end of the job scheduler process, the behavior of the job scheduler process at failure of the process, and log messages output at the failure of the processing, see "Appendix A Processing of the Job Scheduler Process and Log Messages Output at Failure of the Processing."

2.2 Plugin Library Incorporation Settings

The job operation administrator can configure plugin library incorporation with the pmpjm.conf configuration file for job operation management within a resource unit. Make these settings on the system management node. These settings make it possible to control operations, such as job scheduling, through the job selection plugin for the specified resource unit.

The pmpjm.conf file is placed in the following location in the system management node. When changing default setting values, edit this file.

```
/etc/opt/FJSVtcs/Rscunit.d/resource unit name/pmpjm.conf
```

 See

For details of the pmpjm.conf file and how to reflect the settings, see "Job Operation Software Administrator's Guide for Job Management."

Set to ResourceUnit section the path of the directory in which the plugin library to be incorporated into the job scheduler is placed.

Table 2.1 Definition Item Relevant to the Plugin Library in the ResourceUnit Section

Item Name	Definition Contents	Specifiable Values	Default Value
SchedulerPluginLoadPath	The absolute paths of the directories in which the plugin libraries in normal mode are placed	A list of directory names (To specify multiple items, delimit them with a colon ":".)	/etc/opt/FJSVtcs/plugin/pjm/pjsd/normal_mode

Configure settings for the plugin library to be incorporated into the job scheduler in the Scheduler sub section in the ResourceUnit section.

Table 2.2 Definition Items of the Scheduler Sub Section in the ResourceUnit Section

Item Name	Definition Contents	Specifiable Values	Default Value
Name	The name of the scheduler of the plugin library	A 1 to 63-character long character string consisting of single-byte alphanumeric characters, hyphens "-," and underscores "_"	Not omissible
Plugins	File names of the plugin libraries to be read	A list of file names (To specify multiple items, delimit them with a comma ",".)	None

Note

- One subsection scheduler can be defined in the ResourceUnit section.
- If the Plugins item is not specified, the plugin library is not loaded even when the file has the Scheduler section. In this case, the algorithm incorporated in the job scheduler is used.

When setting is reflected, a job scheduler process starts for each plugin library (each Scheduler sub section). The name of the process of the job scheduler in which a plugin library is incorporated is named as follows.

```
PJM pjsd_resource unit name_scheduler name
```

Log messages of this process are output to the following.

```
/var/log/FJSVtcs/pjm/pjsd_resource unit name_scheduler name.log
```

Information

If the pmpjmadm command is executed and setting change relevant to a scheduler plugin is reflected, the job scheduler process is ended, started, or restarted. At this time, since the other processes are not restarted, operations such as submitting jobs and checking job states can be performed as usual even while the setting change is being reflected.

The following example shows settings covering scheduling by the plugin library libplugin.so for all the resource groups in the resource unit rscunit000.
The underlined parts in the setting example are setting items for incorporating the plugin library. The other items that can be set in the pmpjm.conf file are omitted.

```
ResourceUnit {
    ResourceUnitName = rscunit000
    SchedulerPluginLoadPath = /plugin:/foo/bar
    ...
    Scheduler {
        Name = sched001
        Plugins = libplugin.so
```

```
    }
}
```

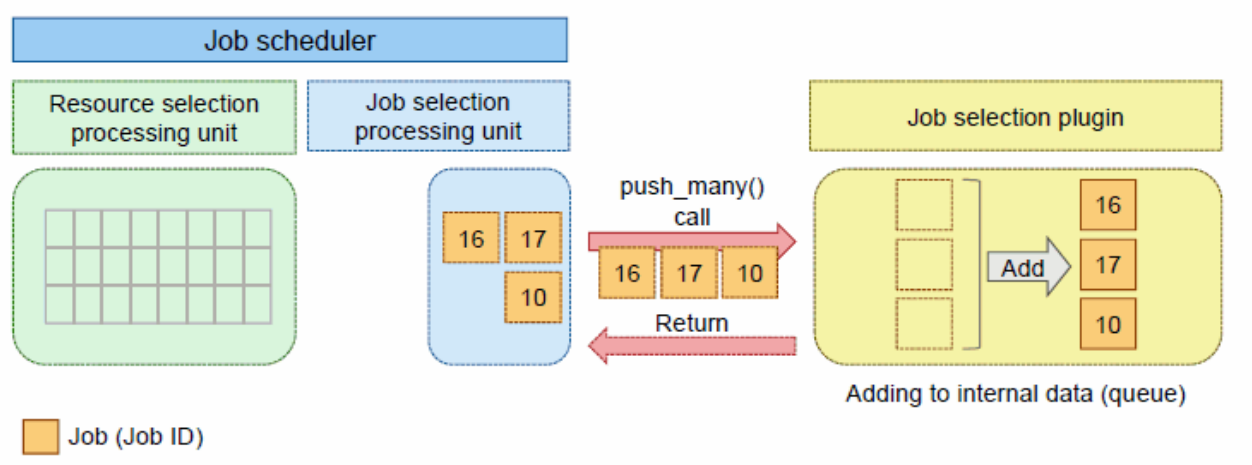
2.3 Scheduling Operation When the Scheduler API is Used

The job selection plugin implemented as a plugin library uses the scheduler API to process scheduling. This section describes scheduling processing (operation) for when the scheduler API is used.

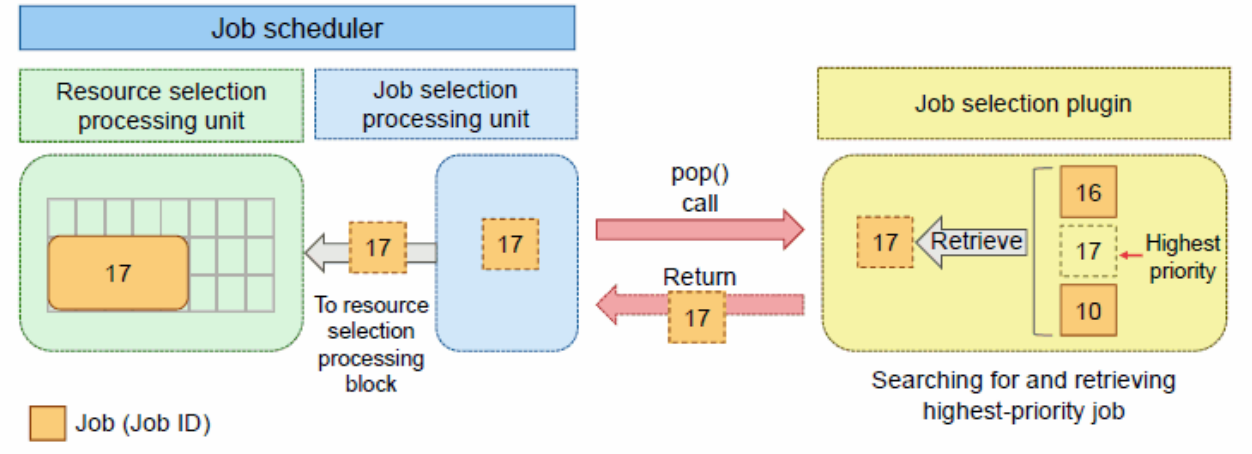
2.3.1 Basic scheduling behavior

The information required for determining the priority of jobs is passed when the job scheduler calls the job selection plugin.
The following figures show the functions of the job selection plugin that are called during scheduling and the order of the function calls.

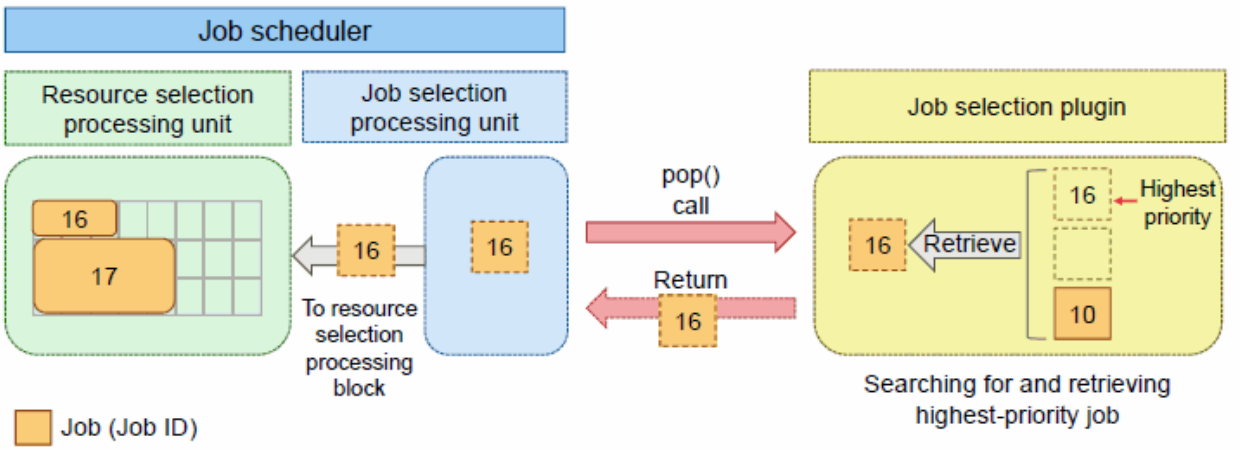
1. Registering jobs to be scheduled (when scheduling begins)
   The job scheduler calls the push_many() function and registers job information for scheduled jobs in the job selection plugin.



2. Retrieving the highest-priority job
   The job scheduler calls the pop() function and retrieves the highest-priority job from the job selection plugin. The retrieved job is passed to the resource selection processing unit, and allocated resources.



3. Retrieving the next job
   The job scheduler calls the pop() function again and retrieves the second highest-priority job from the job selection plugin. The retrieved job is passed to the resource selection processing unit of the job scheduler, and allocated resources.

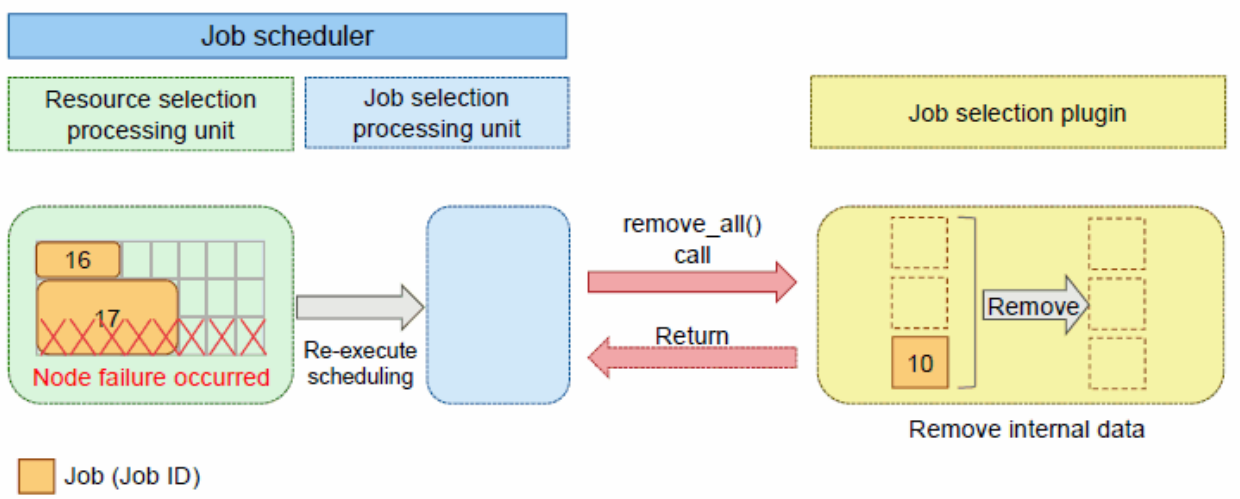


4. Repeating job selection
   The processes in steps 2 and 3 are repeated until the completion of resource selection for all jobs.

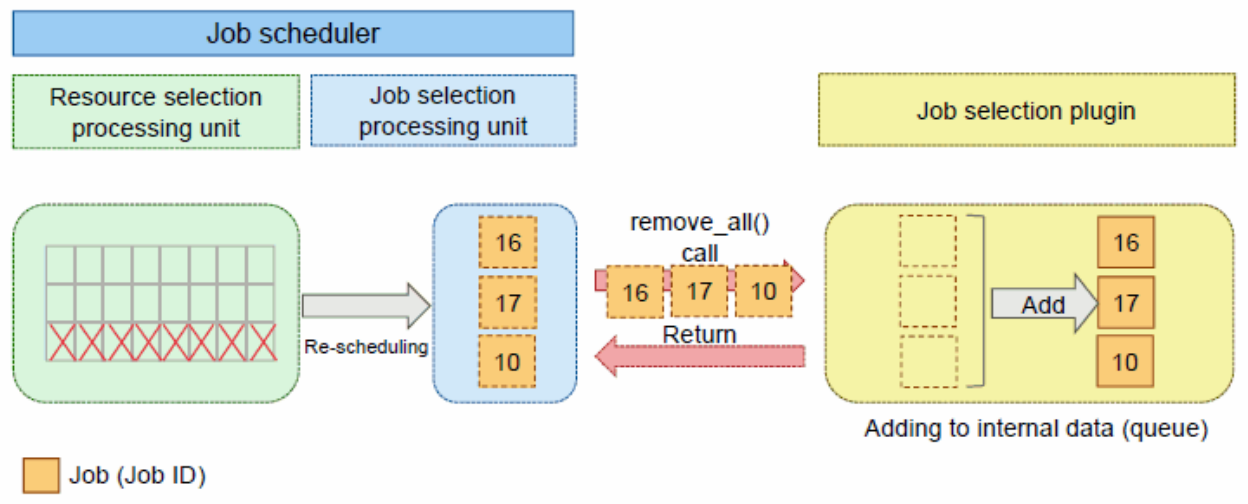
2.3.2 Scheduling behavior when an event occurrence affects scheduling results

If a new job submission, node failure, or other event that affects scheduling results occurs, the functions of the job selection plugin are called in the order shown below to interrupt and resume scheduling.
This section describes the behavior when a node failure occurs.

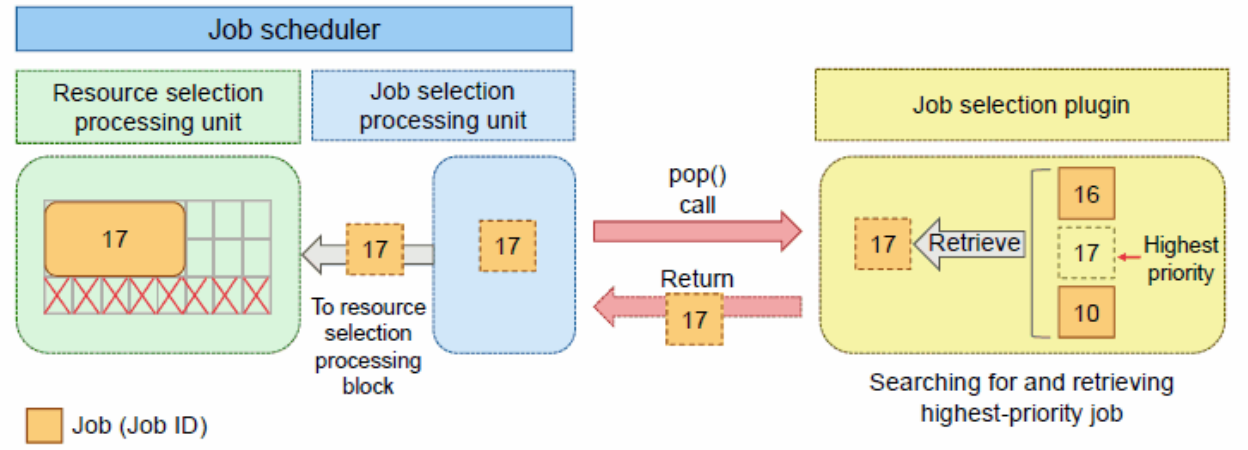
1. Resetting the internal information on the job selection plugin
   The job scheduler calls the remove_all() function to interrupt scheduling by resetting the internal information on the job selection plugin.



2. Resuming scheduling
   After interrupting scheduling, the job scheduler calls the push_many() function, and registers information on the job to be scheduled in the job selection plugin.



3. Retrieving the highest-priority job
   The job scheduler calls the pop() function, and retrieves the highest-priority job from the job selection plugin. The retrieved job is passed to the resource selection processing unit of the job scheduler, and allocated resources.



4. Repeating job selection
   Like in steps 2 and 3 in "2.3.1 Basic scheduling behavior," the pop() function call and resource selection for a job are repeated in subsequent behavior.

See

For details of the above functions and the method, see "Appendix G Reference: Data Structures Relevant to the Job Selection Class" and "Appendix I Reference: Constants and Functions of the Job Selection Plugin."

2.4 Notes on Using the Scheduler Plugin Function

If the scheduler plugin function is used in combination with a step job, simple sort logic such as sorting by the number of nodes cannot satisfy the step job requirement that job execution begin in ascending order of step number. Therefore, to determine the priority of the jobs (sub jobs), it is necessary to implement a plugin library to sort the sub jobs of a step job by criteria that enable resources to be selected in ascending order of step number.

Appendix A Processing of the Job Scheduler Process and Log Messages Output at Failure of the Processing

When using a job selection plugin, the job scheduler performs read processing of the plugin at start of a process, and release processing of the plugin at normal end of the process.
This appendix describes contents of processing performed when a process of the job scheduler is started and when it is ended, and describes the operation performed when the processing fails.

A.1 Process Start Processing of the Job Scheduler

When the job scheduler starts a process, it performs the following.

- Reading the plugin library (dynamic loading)
- Verifying the validity of the plugin library
- Calling the initialization function of the job selection plugin
  - Registering the class (only when the registration is requested in the initialization function)
  - Creating an instance of the job selection class (only when the class is registered in the initialization function)

If any of these operations fails, the job scheduler process discontinues the start processing to prevent operation based on abnormal system setting (plugin library) from being continued. Then, it outputs information used to identify the event of the failure to the job scheduler log, and terminates abnormally.

The following describes details of each of the process start processing operations of the job scheduler, and describes log messages to be output when an operation fails.

In the log output examples provided below, the header part is omitted. In addition, space characters are inserted in a log message instead of line feeds.

A.1.1 Reading the Plugin Library

When the job scheduler starts a process, it reads a plugin library to use based on the plugin setting information (see "2.2 Plugin Library Incorporation Settings") (dynamic loading). If the dynamic loading of the relevant plugin library, dlopen(3), fails, the job scheduler outputs the following error message to the log and terminates abnormally.

```
Failed to load scheduler plugin: dlopen failed.
plugin_path="path of the plugin library"
detail="detailed error cause"
```

The character string indicating the error cause returned by dlerror() is output to *detailed error cause*.

A.1.2 Verifying the Validity of the Plugin Library

After the job scheduler succeeds in dynamic loading of the plugin library, it verifies the validity of the loaded plugin library by checking the following points.

- Whether the required symbols have been defined
  Whether the symbols below have been defined is checked. These symbols are necessary for verifying the operation and validity of the

job selection plugin. These symbols are automatically incorporated in the plugin library by calling the PJSPLG_DEFINE macro. Therefore, a person who creates the plugin does not need to be aware of the symbols.

- PJSPLG_PLUGIN_API_VERSION
  Required API version of the plugin
  This is used to check whether the API version matches.
- PJSPLG_PLUGIN_TYPE
  Plugin type
  This is used to check whether the plugin type matches.
- PJSPLG_PLUGIN_NAME
  Plugin name
  This is output to the log of the job scheduler. This is used to check whether the intended plugin has been loaded.
- PJSPLG_PLUGIN_VERSION
  Plugin version
  This is output to the log of the job scheduler. This is used to check whether the intended plugin has been loaded.
- plugin_init()
  Initialization function of the job selection plugin
- plugin_fini()
  End function of the job selection plugin

If a symbol necessary for verifying the operation and validity of the job selection plugin has not been defined in the plugin library (dlsym(2) call fails), the job scheduler outputs the following error message to the log, and terminates abnormally.

```
Failed to load scheduler plugin: dlsym() failed.
plugin_path="path of the plugin library"
symbol="symbol that failed in being loaded"
detail="detailed error cause"
```

The character string indicating the error cause returned by dlerror() is output to *detailed error cause*.

- Whether the required API version of the plugin matches
  When a plugin library is created, whether PJSPLG_PLUGIN_API_VERSION, a symbol representing the API version of the header file used for compilation, matches the version of the API provided by the job scheduler process is checked.

  If the version of the API on which the plugin library depends and the version of the API provided by the job scheduler are incompatible because, for example, the library with an old version is deployed by mistake, the job scheduler outputs the following message to the log and terminates abnormally.

```
Failed to load scheduler plugin: incompatible scheduler plugin API version required.
plugin_path="path of the plugin library",
required_api_version="version of the API on which the plugin depends",
provided_api_version="version of the API provided by the scheduler"
```

- Whether the plugin type is a job selection plugin
  Whether the value of PJSPLG_PLUGIN_TYPE, a symbol representing the plugin type, matches the value meaning the job selection plugin is checked.

  If the plugin type read from the library does not match the plugin library, the job scheduler outputs the following message to the log and terminates abnormally.

```
Failed to load scheduler plugin: incompatible scheduler plugin identifier found.
plugin_path="path of the plugin library",
declared_identifier="plugin type declared by the plugin",
expected_identifier="plugin type the scheduler assumes"
```

A.1.3 Calling the Initialization Function of the Plugin Library

After completing the reading of the plugin library, the job scheduler calls the initialization function of the job selection plugin plugin_init(), which is loaded from the plugin library.

The plugin initialization function registers the job selection class by calling the API provided by the scheduler that is handed over as an argument of the relevant function (class registration function). The job scheduler internally maintains information of the registered class, and uses it at the time of instance creation, etc.

If the plugin initialization function terminates abnormally, the job scheduler outputs the following message to the log and terminates abnormally.

```
Failed to load scheduler plugin: plugin_init() failed.
plugin="[plugin ID](plugin name:plugin version)"
```

A.1.4 Creating an Instance of the job selection class

After completing the calling of the initialization function of the job selection plugin, the job scheduler calls, the initialization function of the job selection class create(), to create an instance of this class. The job scheduler maintains the instance that is returned as a return value of this function, and performs the scheduling processing unique to the plugin by calling functions of this instance.

If the instance creation method of the scheduling class returns abnormally, the job scheduler outputs the following error message to the log, and terminates abnormally.

```
Failed to load scheduler plugin: JobSelectClass_t::create() failed.
plugin="[plugin ID](plugin name:plugin version)"
```

A.2 Processing on Normal End of a Job Scheduler Process

When the job scheduler ends a process normally, it performs the following.

- Releasing the instance of the job selection class (only when it is registered in the initialization function)
- Calling the termination function of the job selection plugin
  - Deleting the class (only when the deletion is requested in the termination function)
- Releasing the plugin library

If any of these processing operations fails, the job scheduler process discontinues the subsequent termination processing of the job selection plugin. Then, it outputs information used to identify the event of the failure to the job scheduler log, and ends normally. The reason why the process ends normally is because the process itself can be continued even if the releasing processing of the job selection plugin terminates abnormally since this processing is performed when the process of the job scheduler ends.

A.2.1 Releasing the Instance of the Scheduling Class

When the job scheduler ends a process, it calls the end function of the job selection class destroy(), to release the instance of this class.

A.2.2 Calling the Termination Function of the Plugin Library

After completing the releasing of the instance, the job scheduler calls the end function of the job selection plugin plugin_fini(), which is loaded from the plugin library.
The end function of the job selection plugin is a function that has been defined in the plugin library by calling an API provided by the scheduler that is handed over as an argument of the relevant function (class deletion function). The plugin termination function is used to delete the class that the relevant plugin has registered.

If the job scheduler failed in the call of the end function of the job selection plugin, it outputs the following error message to the log, and ends normally.

```
Failed to unload scheduler plugin: plugin_fini() failed.
plugin="[plugin ID](plugin name:plugin version)"
```

After succeeding in calling the end function of the job selection plugin, if the class registered by the plugin to which this function belongs has not been deleted, the job scheduler outputs the following error message to the log, and ends normally.

```
Failed to unload scheduler plugin: plugin_fini() did not deregister job selection class.
plugin="[plugin ID](plugin name:plugin version)"
```

A.2.3 Releasing the Plugin Library

After the end function of the job selection plugin returns, the job scheduler calls dlclose(2) to release the plugin library.

If the job scheduler fails in releasing the plugin library, it outputs the following error message to the log, and ends normally.

```
Failed to unload scheduler plugin: dlclose() failed.
libid="plugin ID",
plugin_path="path of the plugin library",
detail="detailed error cause"
```

The character string indicating the error cause returned by dlerror() is output to *detailed error cause*.

A.3 Processing When a Process of the Job Scheduler Terminates Abnormally

If a process of the job scheduler terminates abnormally due to memory access violation, abort from error detection, etc., the releasing processing of the job selection plugin may not be called. In such a case, the end processing of the OS process releases memory resources, etc. used by the job selection plugin. However, as to the resources (files, IPC resources, etc.) that have not been released by the OS at the timing of the process end, they remain even after the process of the job scheduler terminates abnormally. Therefore, it is necessary to create a job selection plugin that operates normally even when there are undeleted resources.

Appendix B Reference: Common to the Scheduler APIs

This appendix mainly describes the structure and definitions common to the scheduler API.

B.1 Structure

This section describes the structure commonly used by the API functions and job selection plugins.

B.1.1 PjsplgSubjobId_t

PjsplgSubjobId_t is a structure used in common by API functions and the job selection plugin. A sub job ID is an ID used for identifying a sub job. Every sub job has only one sub job ID that is unique in the cluster.

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

typedef struct {
    uint32_t jobid;
    pjsplg_jobmodel_t jobmodel;
    uint32_t bulkno;
    uint32_t stepno;
} PjsplgSubjobId_t;
```

Table B.1 Members of PjsplgSubjobId_t

Members of	Members of	Members of
jobid	uint32_t	Job ID of the job including the relevant sub job
jobmodel	pjsplg_jobmodel_t	Job model of the job including the sub job
bulkno	uint32_t	Bulk number If a sub job is not a bulk job, 0 must be specified.
stepno	uint32_t	Step number If a sub job is not a step job, 0 must be specified.

B.2 Macro Definitions

This section describes the macro definitions commonly used by the API functions and job selection plugins.

B.2.1 PJSPLG_API_VERSION

PJSPLG_API_VERSION is a macro for identifying the version of the scheduler API defined by the header macro file of /usr/include/FJSVtcs/pjm/pjsplugin.h.

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

#define PJSPLG_API_VERSION "1.0.0"
```

B.2.2 PJSPLG_NUM_MAX_RSCGRP

PJSPLG_NUM_MAX_RSCGRP is a macro representing the number of resource groups that can be defined in the resource unit.

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

#define PJSPLG_NUM_MAX_RSCGRP 65535
```

B.3 Type Definitions

This section describes the type definitions commonly used by the API functions and job selection plugins.

B.3.1 pjsplg_ruid_t

pjsplg_ruid_t is a resource unit ID type.

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

typedef int32_t pjsplg_ruid_t;
```

Every resource unit has only one resource unit ID that is unique in the cluster.

B.3.2 pjsplg_rgid_t

pjsplg_rgid_t is a resource group ID type.

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

typedef int32_t pjsplg_rgid_t;
```

Every resource group has only one resource group ID that is unique in the resource unit.

B.4 Enumeration Types

This section describes the enumeration types commonly used by the API functions and job selection plugins.

B.4.1 pjsplg_result_t

pjsplg_result_t is an enumeration type representing the call result of the scheduler API function (success or failure).

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

typedef enum {
    PJSPLG_OK = 0,
    PJSPLG_ERR = -1
} pjsplg_result_t;
```

Table B.2 Codes and values of pjsplg_result_t

Code	Value	Description
PJSPLG_OK	0	Success
PJSPLG_ERR	-1	Failure

B.4.2 pjsplg_error_t

pjsplg_error_t is an enumeration type representing an error code of the scheduler API.

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

typedef enum {
    <Omitted>
    PJSPLG_SUCCESS = 0,
    PJSPLG_ERROR_INVALID_ARGUMENT,
    PJSPLG_ERROR_ALREADY_REGISTERED,
    PJSPLG_ERROR_NOT_REGISTERED,
    PJSPLG_ERROR_NOT_EXIST,
```

```
    PJSPLG_ERROR_ALREADY_GOT,
} pjsplg_error_t;
```

(*) Omitted enumeration types are unavailable.

Table B.3 Codes and values of pjsplg_error_t

Code	Value	Description
PJSPLG_SUCCESS	0	Success (no error has occurred.)
PJSPLG_ERROR_INVALID_ARGUMENT	1	The argument is invalid.
PJSPLG_ERROR_ALREADY_REGISTERED	2	The specified information has already been registered.
PJSPLG_ERROR_NOT_REGISTERED	3	The specified information has not been registered.
PJSPLG_ERROR_NOT_EXIST	4	The specified information does not exist.
PJSPLG_ERROR_ALREADY_GOT	6	The specified information has already been obtained.

B.4.3 pjsplg_jobtype_t

pjsplg_jobtype_t is an enumeration type representing a job type.

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

typedef enum {
    PJSPLG_JOBTYPE_BATCH = 1,
    PJSPLG_JOBTYPE_INTERACTIVE = 2,
} pjsplg_jobtype_t;
```

Table B.4 Codes and values of pjsplg_jobtype_t

Code	Value	Description
PJSPLG_JOBTYPE_BATCH	1	Batch job
PJSPLG_JOBTYPE_INTERACTIVE	2	Interactive job

B.4.4 pjsplg_jobmodel_t

pjsplg_jobmodel_t is an enumeration type representing a job model.

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

typedef enum {
    PJSPLG_JOBMODEL_NORMAL       = 1,
    PJSPLG_JOBMODEL_STEP         = 2,
    PJSPLG_JOBMODEL_BULK         = 3,
} pjsplg_jobmodel_t;
```

Table B.5 Codes and values of pjsplg_jobmodel_t

Code	Value	Description
PJSPLG_JOBMODEL_NORMAL	1	Normal job
PJSPLG_JOBMODEL_STEP	2	Step job
PJSPLG_JOBMODEL_BULK	3	Bulk job

B.5 Variable

This section describes the variable commonly used by the API functions and job selection plugins.

B.5.1 pjsplg_errcode

pjsplg_errcode is a variable for the error codes of the scheduler API. The error code set in this variable corresponds to the error that occurred at the time of the last API call.

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

extern __thread pjsplg_error_t pjsplg_errcode;
```

Appendix C Reference: Data Structures Relevant to Job Information

This appendix describes data structures relevant to job information.

C.1 Structure

This section describes structures relevant to job information.

C.1.1 PjsplgElapsedTimeLimit_t

PjsplgElapsedTimeLimit_t is a structure representing information of the elapsed time limit value (requested elapsed time).

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

typedef struct {
    pjsplg_job_elapsed_time_mode_t mode;
    union {
        uint64_t fixed;
        struct {
            uint64_t min;
            uint64_t max;
        } adaptive;
    } value
} PjsplgElapsedTimeLimit_t;
```

Table C.1 Members of PjsplgElapsedTimeLimit_t

Member			Type	Description
mode			pjsplg_job_elapsed_time_mode_t	Type of specification method of the elapsed time limit value.
value	-		union	Information of elapsed time limit values
	fixed		uint64_t	Elapsed time limit value of a job When the -L elapse=unlimited option is specified for the pjsub command, PJSPLG_ELAPSED_TIME_VALUE_UNLIMITED is set.
	adaptive	-	struct	Information of the elapsed time limit values of a job for which elapsed time limit values are specified as a range This member is valid when mode is PJSPLG_JOB_ELAPSED_TIME_ADAPTIVE.
		min	uint64_t	Minimum value of elapsed time limits
		max	uint64_t	Maximum value of elapsed time limits

C.1.2 PjsplgNodeShape_t

PjsplgNodeShape_t is a structure representing a requested node shape or the number of requested nodes.

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

typedef struct {
    int8_t dim;
    int32_t xsize;
    int32_t ysize;
```

```
    int32_t zsize;
} PjsplgNodeShape_t;
```

Table C.2 Members of PjsplgNodeShape_t

Member	Type	Description
dim	int8_t	The number of dimensions of the node shape (1 to 3)
xsize	int32_t	The X-axis size or the number of requested nodes
ysize	int32_t	The Y-axis size
zsize	int32_t	The Z-axis size

The following are setting values for the above members.

- For the shape of one-dimensional requested nodes or the number of requested nodes
  - dim: 1
  - xsize: Size of the number of requested nodes
  - ysize, zsize: Optional
- For the shape of two-dimensional requested nodes
  - dim: 2
  - xsize: X-axis size
  - ysize: Y-axis size
  - zsize: Optional
- For the shape of three-dimensional requested nodes
  - dim: 3
  - xsize: X-axis size
  - ysize: Y-axis size
  - zsize: Z-axis size

C.1.3 PjsplgNodeReq_t

PjsplgNodeReq_t is a structure representing information of the requested nodes.

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

typedef struct {
     pjsplg_node_req_type_t mode;
     PjsplgNodeShape_t node_shape;
     uint64_t mem_per_node;
} PjsplgNodeReq_t;
```

Table C.3 Members of PjsplgNodeReq_t

Member	Type	Description
mode	pjsplg_node_req_type_t	Node allocation mode Of the values of pjsplg_node_req_type_t, a valid value for the resource unit for which this job is submitted is set.
node_shape	PjsplgNodeShape_t	Shape of the requested nodes or the number of them
mem_per_node	uint64_t	Requested memory amount per node

C.1.4 PjsplgVnodeReq_t

PjsplgVnodeReq_t is a structure representing information on requested virtual nodes.

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

typedef struct {
    uint32_t num_vnode;
    uint64_t mem_per_vnode;
    uint32_t cpu_per_vnode;
} PjsplgVnodeReq_t;
```

Table C.4 Members of PjsplgVnodeReq_t

Member	Type	Description
num_vnode	uint32_t	Number of requested virtual nodes
mem_per_vnode	uint64_t	Amount of requested memory per virtual node
cpu_per_vnode	uint32_t	Number of requested CPU cores per virtual node

C.1.5 PjsplgRscReq_t

PjsplgRscReq_t is a structure representing information of the resources requested by a job.

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

typedef struct {
    pjsplg_job_rsc_type_t type;
    union {
        PjsplgNodeReq_t node;
        PjsplgVnodeReq_t vnode;
    } value;
} PjsplgRscReq_t;
```

Table C.5 Members of PjsplgRscReq_t

Member		Type	Description
type		pjsplg_job_rsc_type_t	Requested resource type
value	-	union	Information of the requested resource amount
	node	PjsplgNodeReq_t	Information of the requested nodes
	vnode	PjsplgVnodeReq_t	Information of the requested virtual nodes

C.1.6 PjsplgJobSubmitParam_t

PjsplgJobSubmitParam_t is a structure representing job submission parameters. This information is invariable during execution of the job unless the pjalter command, etc. is executed.

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

typedef struct {
    <Omitted>
    pjsplg_jobtype_t jobtype;

    uid_t uid;
    gid_t gid;

    int16_t apriority;
    int16_t priority;

    int16_t user_priority;
```

```
    int16_t group_priority;
    int16_t user_in_group_priority;

    pjsplg_rgid_t rgid;

    uint64_t attribute;

    PjsplgElapsedTimeLimit_t elapsed_time_limit;
    uint64_t total_preceding_elapsed_time_limit;

    PjsplgRscReq_t req_rsc;

    struct timespec accept_date;
    struct timespec first_subjob_accept_date;
    time_t specified_start_date;
    pjsplg_net_route_t net_route;
} PjsplgJobSubmitParam_t;
```

(*) Omitted enumeration types are unavailable.

Table C.6 Members of PjsplgJobSubmitParam_t

Member	Type	Description
jobtype	pjsplg_jobtype_t	Job type If the --interact option of the pjsub command is specified, PJSPLG_JOBTYPE_INTERACTIVE is set. If the option is not specified, PJSPLG_JOBTYPE_BATCH is set.
uid	uid_t	Job execution user ID The user ID of the user who executed the pjsub command is set.
gid	gid_t	Job execution group ID The group ID of the group specified in the -g option of the pjsub command or the group ID of the user who executed the pjsub command is set.
apriority	int16_t	Job priority level within the resource unit The value specified in the --apriority option of the pmalter command or the system default value of 127 is set.
priority	int16_t	Job priority level specified by the user The value specified in the -p option of the pjsub, pjalter, or pmalter command or the default value specified in the joblimit priv-pri definition item in the job ACL is set.
user_priority	int16_t	Priority level of the job execution user The value specified in the define pri definition item in the job ACL is set.
group_priority	int16_t	Priority level of the job execution group The value specified in the define pri-g definition item in the job ACL is set.
user_in_group_priority	int16_t	Priority level of the job execution user within the group The value specified in the define ingroup-pri definition item in the job ACL is set.
rgid	pjsplg_rgid_t	Resource group ID The value specified in the -L rscgrp= option of the pjsub, pjalter, or pmalter command or the default value

Member	Type	Description
		specified in the define rscgroup definition item in the job ACL is set.
attribute	uint64_t	Job attributes The logical addition of the valid macros in the macro named PJSPLG _ JOB _ ATTR _ * is set.
elapsed_time_limit	PjsplgElapsedTimeLimit_t	Information of the elapsed time limit value (requested elapsed time) The value specified in the -L elapse= option of the pjsub command or the default value specified in the joblimit elapse definition item in the job ACL is set.
total_preceding_elapsed_time_limit	uint64_t	Total of the elapsed time limit values (requested elapsed time) of preceding sub jobs The total of the elapsed time limit values of preceding sub jobs that had not ended when this sub job was submitted (when the pjsub command was executed) is set. The total includes the elapsed time limit values of preceding sub jobs that ended after this sub job was submitted but before it was executed. For a non-step job or the first sub job, 0 is set.
req_rsc	PjsplgRscReq_t	Information of the requested resource The value specified in the -L node= or node-mem= option of the pjsub command or the -L vnode= or vnode-mem= option of the pjsub command or the default value specified in the joblimit node, joblimit nodemem, joblimit vnode, or joblimit vnode-mem definition item in the job ACL is set.
accept_date	struct timespec	Job acceptance time The execution time of the pjsub command is set.
first_subjob_accept_date	struct timespec	Acceptance time of the first sub job of a step job For a non-step job or the first sub job, 0 is set.
specified_start_date	time_t	Job execution start specified time The execution start time specified in the --at option of the pjsub command is set. For the execution start time is not specified, 0 is set.
net_route	pjsplg_net_route_t	Communication path change of a job when a Tofu interconnect link goes down The value specified in the --net-route option of the pjsub command or the default value specified in the define net-route item of the job ACL function is set.

C.1.7 PjsplgJobVariableParam_t

PjsplgJobVariableParam_t is a structure representing the variable parameters of a job. A variable parameter of a job is a parameter whose content varies with the state transition during execution of the job.

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

typedef struct {
    <Omitted>
    pjsplg_job_state_t state;
    uint64_t wait_time;
    uint32_t num_restart;

    struct timespec all_prec_subjob_exit_date;
```

```
    struct timespec last_released_date;
} PjsplgJobVariableParam_t;
```

(*) Omitted enumeration types are unavailable.

Table C.7 Members of PjsplgJobVariableParam_t

Member	Type	Description
state	pjsplg_job_state_t	Current job state
wait_time	uint64_t	Job execution wait time The elapsed time from the job acceptance time (execution time of the pjsub command) to the scheduling start time is set. For a job submitted after scheduling start, 0 is set.
num_restart	uint32_t	The number of times of re-execution The cumulative number of job re-executions for such a reason as node down. This is the cumulative number of times of job execution requests, and this number does not increase when re-queueing is performed.
all_prec_subjob_exit_date	struct timespec	End time of the preceding sub job For a job that is not a step job, or a step job whose preceding sub job has not completed, 0 is set.
last_released_date	struct timespec	Last release time The time when the end user executes the pjrls command for this job or the preceding sub job (only in case of a step job) and a transition the QUEUED state occurs is set. For a job for which the pjrls command has never been executed, 0 is set. When the administrator executes the pjrls command, this time is not changed. According to the scheduling policy of FCFS (First-Come First-Serve), job priority levels are determined based on the job acceptance time. For example, end users a and b submit jobs at the following timing. a. When creating a job script, the program to be executed by the job script, and input data is completed, and the job becomes executable b. At the time of completion of creation of the job script After the job is submitted, the job is fixed by using the pjhold command so that is not executed, and then the program and input data are created. After completing their creation, the fixed job is released by using the pjrls command. In such a case, if the FCFS policy based on the job acceptance time is observed, job b takes priority over job a even if they become executable at the same time. In such a condition, to realize control that makes the job priority levels of a and b equal, you can use this information.

C.1.8 PjsplgSubjob_t

PjsplgSubjob_t is a structure representing job information (a structure that keeps information (on jobs in the case of normal jobs or on sub jobs in the case of step or bulk jobs)). The job scheduler uses this structure to notify the job selection plugin of job (or sub job) information.

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

typedef struct {
    <Omitted>
    PjsplgSubjob_t *prev_p;
    PjsplgSubjob_t *next_p;

    PjsplgSubjobId_t sjid;
    PjsplgJobSubmitParam_t submit_param;
    PjsplgJobVariableParam_t variable_param;
} PjsplgSubjob_t;
```

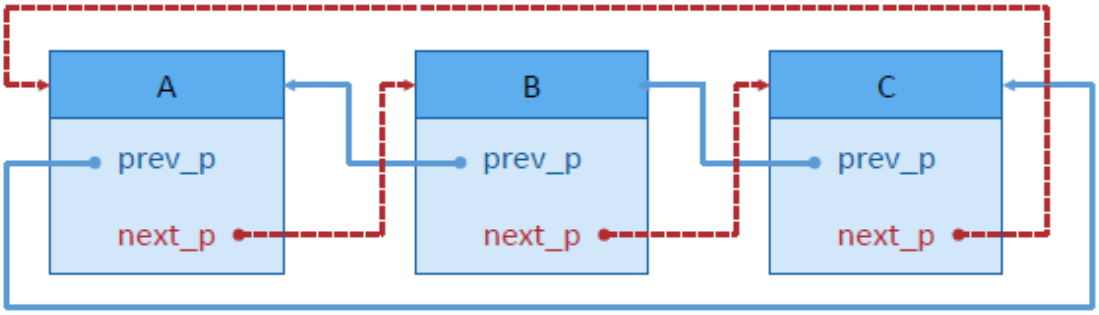
(*) Omitted enumeration types are unavailable.

Table C.8 Members of PjsplgSubjob_t

Member	Type	Description
prev_p	PjsplgSubjob_t *	Previous job information
next_p	PjsplgSubjob_t *	Next job information
sjid	PjsplgSubjobId_t	Sub job ID
submit_param	PjsplgJobSubmitParam_t	Submission parameter
variable_param	PjsplgJobVariableParam_t	Variable parameter

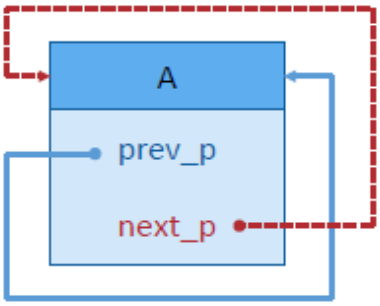
By setting the pointers to other job information in prev_p and next_p, a bidirectional circular link list of job information can be made up. If multiple sets of job information are handed over through the scheduler API, the job scheduler makes up a bidirectional circular link list structure that includes all the job information to be handed over with prev_p and next_p. Then, the job scheduler hands over all the job information to the job selection plugin by handing over the pointer to the first job information set to it.
If job information A, B, and C are arranged in sequence, the connection of the bidirectional circular link list is as follows.

Figure C.1 Connection of the Bidirectional Circular Link List when Job Information A, B, and C are Arranged in Sequence



The connection of a bidirectional circular link list that includes only job information A is as follows.

Figure C.2 Connection of a Bidirectional Circular Link List That Includes Only Job Information A



C.2 Macro Definitions

This section describes macro definitions relevant to job information.

C.2.1 PJSPLG_ELAPSED_TIME_VALUE_UNLIMITED

PJSPLG_ELAPSED_TIME_VALUE_UNLIMITED is a macro representing the elapsed time for when unlimited is specified.

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

#define PJSPLG_ELAPSED_TIME_VALUE_UNLIMITED (~((uint64_t)0))
```

This is set for jobs that have the -L elapse=unlimited option or the like specified in the pjsub command.

C.2.2 PJSPLG_JOB_ATTR_*

PJSPLG_JOB_ATTR_* is a macro representing job attributes.

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

<Omitted>
#define PJSPLG_JOB_ATTR_NODE_STRICT       0x00000001LLU
#define PJSPLG_JOB_ATTR_MPI_ASSIGN_ONLINE 0x00000100LLU
#define PJSPLG_JOB_ATTR_RESTART           0x00010000LLU
```

(*) Omitted enumeration types are unavailable.

Table C.9 Macros of PJSPLG_JOB_ATTR_*

Macro name	Value	Description
PJSPLG_JOB_ATTR_NODE_STRICT	0x00000001LLU	Attribute that represents jobs that have resources allocated as per the specified node shape without rotation This flag is set for jobs submitted with :strict specified in the -L node option of the pjsub command.
PJSPLG_JOB_ATTR_MPI_ASSIGN_ONLINE	0x00000100LLU	Attribute that represents jobs whose allocation range does not include failed nodes This flag is set for jobs submitted with the --mpi assign-online-node option specified in in the pjsub command.
PJSPLG_JOB_ATTR_RESTART	0x00010000LLU	Attribute that represents re-executable jobs This flag is set in the following cases. The flag is set for jobs submitted with the --restart option specified in the pjsub command. It is also set for job models that have re-execution enabled in the

Macro name	Value	Description
		papjm.conf or pmpjm.conf file and do not have the --no-restart option specified in the pjsub command.

C.3 Enumeration Types

This section describes enumeration types relevant to job information.

C.3.1 pjsplg_job_elapsed_time_mode_t

pjsplg_job_elapsed_time_mode_t is an enumeration type to specify specification method types of elapsed time limit values of a job.

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

typedef enum {
    PJSPLG_JOB_ELAPSED_TIME_FIXED =1,
    PJSPLG_JOB_ELAPSED_TIME_ADAPTIVE,
} pjsplg_job_elapsed_time_mode_t;
```

Table C.10 Codes and values of pjsplg_job_elapsed_time_mode_t

Code	Value	Description
PJSPLG_JOB_ELAPSED_TIME_FIXED	1	Job for which only the upper limit of the elapsed time limit values is specified
PJSPLG_JOB_ELAPSED_TIME_ADAPTIVE	2	Job for which a range of elapsed time limit values is specified

C.3.2 pjsplg_job_rsc_type_t

pjsplg_job_rsc_type_t is an enumeration type representing requested resource types of a job.

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

typedef enum {
    PJSPLG_JOB_RSC_TYPE_NODE = 1,
    PJSPLG_JOB_RSC_TYPE_VNODE,
} pjsplg_job_rsc_type_t;
```

Table C.11 Codes and values of pjsplg_job_rsc_type_t

Code	Value	Description
PJSPLG_JOB_RSC_TYPE_NODE	1	Node allocation job
PJSPLG_JOB_RSC_TYPE_VNODE	2	Virtual node allocation job

C.3.3 pjsplg_node_req_type_t

pjsplg_node_req_type_t is an enumeration type representing node allocation modes.

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

typedef enum {
    PJSPLG_NODE_REQ_TORUS = 1,
    PJSPLG_NODE_REQ_MESH,
    PJSPLG_NODE_REQ_NONCONT,
    PJSPLG_NODE_REQ_NODEID,
} pjsplg_node_req_type_t;
```

Table C.12 Codes and values of pjsplg_node_req_type_t

Code	Value	Description
PJSPLG_NODE_REQ_TORUS	1	Torus mode (Tofu unit is exclusively used.)
PJSPLG_NODE_REQ_MESH	2	Mesh mode
PJSPLG_NODE_REQ_NONCONT	3	Non-contiguous mode
PJSPLG_NODE_REQ_NODEID	4	Node allocation to a job submitted to a PRIMERGY server resource unit

C.3.4 pjsplg_job_state_t

pjsplg_job_state_t is an enumeration type representing the state of a job. This job state refers to the state of a job internally managed by the job scheduler. Therefore, the set state may be different from the job state shown by the pjstat command.

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

typedef enum {
    <Omitted>
    PJSPLG_JOB_STATE_QUEUED    = 1,
} pjsplg_job_state_t;
```

(*) Omitted enumeration types are unavailable.

Table C.13 Codes and values of pjsplg_job_state_t

Code	Value	Description
PJSPLG_JOB_STATE_QUEUED	1	Job execution wait state A request to start job execution causes a transition to the execution start wait state.

C.3.5 pjsplg_net_route_t

pjsplg_net_route_t is an enumeration type representing whether to change the communication path of a job when a Tofu interconnect link goes down.

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

typedef enum {
    PJSPLG_NET_ROUTE_DYNAMIC  = 1,
    PJSPLG_NET_ROUTE_STATIC,
} pjsplg_net_route_t;
```

Table C.14 Codes and Values of pjsplg_net_route_t

Code	Value	Description
PJSPLG_NET_ROUTE_DYNAMIC	1	Dynamically changes the communication path when a Tofu interconnect link goes down. Job execution continues.
PJSPLG_NET_ROUTE_STATIC	2	Does not change the communication path when a Tofu interconnect link goes down. The job ends abnormally.

Appendix D Reference: Data Structures Relevant to Resource Group Information

This appendix describes data structures relevant to resource group information.

D.1 Structure

This section describes structures relevant to resource group information.

D.1.1 PjsplgRscgrpConf_t

PjsplgRscgrpConf_t is a structure representing setting information of a resource group.

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

typedef struct {
    int16_t priority;
} PjsplgRscgrpConf_t;
```

Table D.1 Members of PjsplgRscgrpConf_t

Member	Type	Description
priority	int16_t	Job selection priority level of the resource group

D.1.2 PjsplgRscgrp_t

PjsplgRscgrp_t is a structure representing resource group information. This includes the resource group ID, resource group name, and job selection priority of the resource group.

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

typedef struct {
    <Omitted>
    pjsplg_rgid_t rgid;
    char name[PJSPLG_MAX_RSCGRP_NAME_LEN + 1];
    PjsplgRscgrpConf_t conf;
} PjsplgRscgrp_t;
```

(*) Omitted enumeration types are unavailable.

Table D.2 Members of PjsplgRscgrp_t

Member	Type	Description
rgid	pjsplg_rgid_t	Resource group ID
name	char[]	Resource group name
conf	PjsplgRscgrpConf_t	Resource group setting information

D.2 Macro Definitions

This section describes macro definitions relevant to resource group information.

D.2.1 PJSPLG_MAX_RSCGRP_NAME_LEN

PJSPLG_MAX_RSCGRP_NAME_LEN is a macro representing the maximum number of characters in a resource group name.

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

#define PJSPLG_MAX_RSCGRP_NAME_LEN 63
```

The 63 characters do not include null characters.

Appendix E Reference: Data Structures Relevant to Resource Unit Information

This appendix describes data structures relevant to resource unit information.

E.1 Structure

This section describes structures relevant to resource unit information.

E.1.1 PjsplgRscunit_t

PjsplgRscunit_t is a structure representing resource unit information. This includes information of resource unit configurations, status, and the nodes and resource groups belonging to the resource unit.

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

typedef struct {
    <Omitted>
    pjsplg_ruid_t ruid;
    char name[PJSPLG_MAX_RSCUNIT_NAME_LEN + 1];
    PjsplgRscgrp_t * rscgrp_map_p[PJSPLG_NUM_MAX_RSCGRP];
} PjsplgRscunit_t;
```

(*) Omitted enumeration types are unavailable.

Table E.1 Members of PjsplgRscunit_t

Member	Type	Description
ruid	pjsplg_ruid_t	Resource unit ID
name	char[]	Resource unit name
rscgrp_map_p	PjsplgRscgrp_t *[]	Array of the resource groups included in this resource unit By accessing this array with the specification of a resource group ID as a subscript, the corresponding resource group information can be obtained. If the resource group corresponding to the resource group ID does not exit, NULL is returned.

E.2 Macro Definitions

This section describes macro definitions relevant to resource unit information.

E.2.1 PJSPLG_MAX_RSCUNIT_NAME_LEN

PJSPLG_MAX_RSCUNIT_NAME_LEN is a macro representing the maximum number of characters in a resource unit name.

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

#define PJSPLG_MAX_RSCUNIT_NAME_LEN 63
```

The 63 characters do not include null characters.

Appendix F Reference: API Functions That Can be Used From the Job Selection Class

This appendix describes API functions that can be used from the job selection class and are provided by the job scheduler function.

F.1 Structure

This section describes structures for representing API functions that can be used from the job selection class.

F.1.1 PjsplgJobSelectClassApi_t

PjsplgJobSelectClassApi_t is a structure representing API functions that can be used from the job selection class and are provided by the job scheduler. This structure is handed over as an argument to each function in PjsplgJobSelectClass _ t. The functions provided by this structure can be used only from the inside of this function.

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

typedef struct {
    pjsplg_result_t (*get_rscunit)(const PjsplgRscunit_t **rscunit_pp);
    pjsplg_result_t (*put_rscunit)(const PjsplgRscunit_t **rscunit_pp);
   pjsplg_result_t (*get_user_fshare_value)(pjsplg_fshare_set_id_t fshare_set_id, uid_t uid, int64_t
*value_p);
    pjsplg_result_t (*get_group_fshare_value)(pjsplg_fshare_set_id_t fshare_set_id, gid_t gid,
int64_t *value_p);
    pjsplg_result_t (*get_user_in_group_fshare_value)
                    (pjsplg_fshare_set_id_t fshare_set_id, uid_t uid, gid_t gid, int64_t *value_p);
} PjsplgJobSelectClassApi_t;
```

Table F.1 Members of PjsplgJobSelectClassApi_t

Member	Type	Description
get_rscunit()	Function pointer	Obtains the right to access resource unit information.
put_rscunit()		Returns the right to access resource unit information.
get_user_fshare_value()		Obtains the fair share value of the specified user.
get_group_fshare_value()		Obtains the fair share value of the specified group.
get_user_in_group_fshare_value()		Obtains the user fair share value in the group of the specified user.

The following details each member.

get_rscunit()

Obtains the right to access resource unit information.

```
pjsplg_result_t (*get_rscunit)(const PjsplgRscunit_t **rscunit_pp);
```

The processing below relevant to the resource unit is not performed until the right to access the resource unit information obtained by calling this function is returned (`put_rscunit()` is called or the function that called this function returns).

- Resource unit setting change
- Node state transition

This is because the other threads of the job scheduler main body cannot access the resource unit information while the job selection plugin maintains the access right.

Table F.2 Argument of get_rscunit()

Argument	Description
rscunit_pp	Pointer to the resource unit information to be obtained

Table F.3 Return values of get_rscunit()

Return value	Description
PJSPLG_OK	Acquisition success.
PJSPLG_ERR	Acquisition failure When this function failed in obtaining resource unit information, it returns an error. The cause of the error is set in pjsplg_errcode, the error code variable of the scheduler API. For details of pjsplg_errcode, see "pjsplg_errcode."

In the case of an error, this function sets the following error code in pjsplg_errcode.

Table F.4 Error code set in pjsplg_errcode

Error code	Description
PJSPLG_ERROR_INVALID_ARGUMENT	The argument is invalid.

put_rscunit()

Returns the right to access resource unit information.

```
pjsplg_result_t (*put_rscunit)(const PjsplgRscunit_t **rscunit_pp);
```

As to the resource unit information whose access right is returned by this function, its access right can be obtained again by calling get_rscunit(). After the call of this function, NULL is set in *rscunit_pp.

Table F.5 Argument of put_rscunit()

Argument	Description
rscunit_pp	Pointer to the pointer to the resource unit information to be returned

Table F.6 Return values of put_rscunit()

Return value	Description
PJSPLG_OK	Return success.
PJSPLG_ERR	Return failure When this function failed in returning resource unit information, it returns an error. The cause of the error is set in pjsplg_errcode, the error code variable of the scheduler API. For details of pjsplg_errcode, see "pjsplg_errcode."

In the case of an error, this function sets the following error code in pjsplg_errcode.

Table F.7 Error code set in pjsplg_errcode

Error code	Description
PJSPLG_ERROR_INVALID_ARGUMENT	The argument is invalid.

get_user_fshare_value()

Obtains the fair share value of the specified user.

```
pjsplg_result_t (*get_user_fshare_value)(fshare_set_id_t fshare_set_id, uid_t uid, int64_t
*value_p);
```

Table F.8 Argument of get_user_fshare_value()

Argument	Description
fshare_set_id	Fair share set ID whose fair share value is to be obtained
uid	User ID of the user to be obtained

Argument	Description
value_p	Fair share value

Table F.9 Return values of get_user_fshare_value()

Return value	Description
PJSPLG_OK	Acquisition success.
PJSPLG_ERR	Acquisition failure When this function failed in obtaining the fair share value of the specified user, it returns an error. The cause of the error is set in pjsplg_errcode, the error code variable of the scheduler API. For details of pjsplg_errcode, see "pjsplg_errcode."

In the case of an error, this function sets the following error code in pjsplg_errcode.

Table F.10 Error codes set in pjsplg_errcode

Error code	Description
PJSPLG_ERROR_NOT_EXIST	The fair share set corresponding to the fair share set ID specified by the argument does not exist.
PJSPLG_ERROR_INVALID_ARGUMENT	The argument is invalid.

get_group_fshare_value()

Obtains the fair share value of the specified group.

```
pjsplg_result_t (*get_group_fshare_value)(pjsplg_fshare_set_id_t fshare_set_id, gid_t gid, int64_t *value_p);
```

Table F.11 Argument of get_group_fshare_value()

Argument	Description
fshare_set_id	Fair share set ID whose fair share value is to be obtained
gid	Group ID of the group to be obtained
value_p	Fair share value

Table F.12 Return values of get_group_fshare_value()

Return value	Description
PJSPLG_OK	Acquisition success.
PJSPLG_ERR	Acquisition failure If this function fails in obtaining the fair share value of the specified group, it returns an error. The cause of the error is set in pjsplg_errcode, the error code variable of the scheduler API. For details of pjsplg_errcode, see "pjsplg_errcode."

In the case of an error, this function sets the following error code in pjsplg_errcode.

Table F.13 Error codes set in pjsplg_errcode

Error code	Description
PJSPLG_ERROR_NOT_EXIST	The fair share set corresponding to the fair share set ID specified by the argument does not exist.
PJSPLG_ERROR_INVALID_ARGUMENT	The argument is invalid.

get_user_in_group_fshare_value

Obtains the user fair share value in the group of the specified user.

```
pjsplg_result_t (*get_user_in_group_fshare_value)
                (pjsplg_fshare_set_id_t fshare_set_id, uid_t uid, gid_t gid, int64_t *value_p);
```

Table F.14 Argument of get_user_in_group_fshare_value

Argument	Description
fshare_set_id	Fair share set ID whose fair share value is to be obtained
uid	User ID of the user to be obtained
gid	Group ID of the group to be obtained
value_p	Fair share value

Table F.15 Return values of get_user_in_group_fshare_value

Return value	Description
PJSPLG_OK	Acquisition success.
PJSPLG_ERR	Acquisition failure If this function fails in obtaining the user fair share value in the group of the specified user, it returns an error. The cause of the error is set in pjsplg_errcode, the error code variable of the scheduler API. For details of pjsplg_errcode, see "pjsplg_errcode."

In the case of an error, this function sets the following error code in pjsplg_errcode.

Table F.16 Error codes set in pjsplg_errcode

Error code	Description
PJSPLG_ERROR_NOT_EXIST	The fair share set corresponding to the fair share set ID specified by the argument does not exist.
PJSPLG_ERROR_INVALID_ARGUMENT	The argument is invalid.

Appendix G Reference: Data Structures Relevant to the Job Selection Class

This appendix describes data structures relevant to the job selection class.

G.1 Structure

This section describes structures relevant to the job selection class.

G.1.1 PjsplgJobSelectClass_t

PjsplgJobSelectClass_t is a structure representing the job selection class. This structure is a collection of functions that the job selection plugin must implement to replace the job selection process. To implement the job selection plugin, the functions required by this class must be implemented in the job selection plugin.

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

typedef struct {
    void *(*create)(void);
    void (*destory)(void *instance_p);
    pjsplg_result_t (*push_many)
        (void *instance_p, const PjsplgJobSelectClassApi_t *api_p,
        PjsplgSubjob_t *jobs_list_p);
    pjsplg_result_t (*pop)
        (void *instance_p, const PjsplgJobSelectClassApi_t *api_p,
        PjsplgSubjob_t **job_p);
    pjsplg_result_t (*remove_all)
        (void *instance_p, const PjsplgJobSelectClassApi_t *api_p);
} PjsplgJobSelectClass_t;
```

Table G.1 Members of PjsplgJobSelectClass_t

Member	Type	Description
create()	Function pointer	Create an instance (internal data) of the job selection class.
destroy()		Release an instance (Internal data generated by the create () function) of the job selection class.
push_many()		Registers jobs as targets in job selection.
pop()		Retrieves the highest-priority job among registered jobs.
remove_all()		Removes all jobs.

The following details each member.

create()

This member generates an instance of the job selection class.

```
void *(*create)(void);
```

It generates and returns an instance of the job selection class as the return value.

Table G.2 Return codes of create()

Return code	Description
!NULL	Pointer to an instance of the job selection class (Generation success)
NULL	Generation failure

The function returns the pointer to the instance when successfully generated. If generation fails, the function returns NULL, and the scheduling processing in progress is interrupted.

destroy()

This member releases an instance of the job selection class.

```
void (*destory)(void *instance_p);
```

The member releases the instance of the job selection class given by the argument instance_p.

Table G.3 Argument of destroy()

Argument	Description
instance_p	Instance of the job selection class

push_many()

This member registers jobs as targets in job selection.

```
pjsplg_result_t (*push_many)
    (void *instance_p, const PjsplgJobSelectClassApi_t *api_p,
    PjsplgSubjob_t *jobs_list_p);
```

Table G.4 Arguments of push_many()

Argument	Description
instance_p	Instance of the job selection class
api_p	Job scheduler-provided API function that can be used from this function
jobs_list_p	List of job information to be registered

Table G.5 Return codes of push_many()

Return code	Description
PJSPLG_OK	Registration success
PJSPLG_ERR	Registration failure

If registration fails, the function returns PJSPLG_ERR, and the scheduling processing in progress is interrupted.

pop()

This member retrieves the highest-priority job among registered jobs.

```
pjsplg_result_t (*pop)
    (void *instance_p, const PjsplgJobSelectClassApi_t *api_p,
    PjsplgSubjob_t **job_pp);
```

Table G.6 Arguments of pop()

Argument	Description
instance_p	Instance of the job selection class
api_p	Job scheduler-provided API function that can be used from this function
job_pp	Pointer to the pointer to the highest-priority job information

Table G.7 Return codes of pop()

Return code	Description
PJSPLG_OK	Acquisition success
PJSPLG_ERR	Acquisition failure

If no job information is registered, the function sets NULL as *job_pp, returns PJSPLG_OK, and returns. If retrieval fails, the function returns PJSPLG_ERR, and the scheduling processing in progress is interrupted.

remove_all()

This member removes all jobs.

```
pjsplg_result_t (*remove_all)
    (void *instance_p, const PjsplgJobSelectClassApi_t *api_p);
```

Table G.8 Arguments of remove_all()

Argument	Description
instance_p	Instance of the job selection class
api_p	Job scheduler-provided API function that can be used from this function

Table G.9 Return codes of remove_all()

Return code	Description
PJSPLG_OK	Remove success
PJSPLG_ERR	Remove failure

If removal fails, the function returns PJSPLG_ERR, and the scheduling processing in progress is interrupted.

Appendix H Reference: Data Structure Relevant to the Scheduler Plugin Manager

Use the scheduler plugin manager to register or release the job selection class so that the job scheduler use it.
This appendix describes a data structure relevant to the scheduler plugin manager.

H.1 Structure

This section describes a structure relevant to the scheduler plugin manager.

H.1.1 PjsplgManager_t

PjsplgManager_t is a structure representing the scheduler plugin manager.

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

typedef struct {
    <Omitted>
    pjsplg_result_t (*register_job_select_class)
         (PjsplgManager_t *mng_p, const PjsplgJobSelectClass_t *class_p);
    pjsplg_result_t (*deregister_job_select_class)
         (PjsplgManager_t *mng_p, const PjsplgJobSelectClass_t *class_p);
} PjsplgManager_t;
```

(*) Omitted enumeration types are unavailable.

Table H.1 Members of PjsplgManager_t

Member	Description
register_job_select_class()	Registers the job selection class.
deregister_job_select_class()	Removes the job selection class.

The following details each member.

register_job_select_class()

This member registers the job selection class.

```
pjsplg_result_t (*register_job_select_class)
    (PjsplgManager_t *mng_p, const PjsplgJobSelectClass_t *class_p);
```

The class registered by this function is called when a job is selected. You can register up to one job selection class.

Table H.2 Arguments of register_job_select_class()

Argument	Description
mng_p	Scheduler plugin manager
class_p	Job selection class to be registered

Table H.3 Return values of register_job_select_class()

Return value	Description
PJSPLG_OK	Registration success
PJSPLG_ERR	Registration failure If the job selection class is already registered, an error is returned. The cause of the error is set in the pjsplg_errcode variable for a scheduler API error code. For details on pjsplg_errcode, see "pjsplg_errcode."

If an error occurs, this function sets the following error code in pjsplg_errcode.

Table H.4 Error code set in pjsplg_errcode

Error code	Description
PJSPLG_ERROR_ALREADY_REGISTERED	The job selection class is already registered.

deregister_job_select_class()

This member removes the class registered by the register_job_select_class.

Table H.5 Arguments of deregister_job_select_class()

Argument	Description
mng_p	Scheduler plugin manager
class_p	Job selection class to be removed

Table H.6 Return values of deregister_job_select_class()

Return value	Description
PJSPLG_OK	Deletion success
PJSPLG_ERR	Deletion failure If the job selection class is not registered, an error is returned. The cause of the error is set in the pjsplg_errcode variable for a scheduler API error code. For details on pjsplg_errcode, see "pjsplg_errcode."

If an error occurs, this function sets the following error code in pjsplg_errcode.

Table H.7 Error code set in pjsplg_errcode

Error code	Description
PJSPLG_ERROR_NOT_REGISTERED	The specified job selection class is not registered.

Appendix I Reference: Constants and Functions of the Job Selection Plugin

This appendix describes the constants and functions that every job selection plugin must define.

I.1 Macro Definitions

This section describes a macro definition used by job selection plugins.

I.1.1 PJSPLG_DEFINE

PJSPLG_DEFINE is a macro for defining a plugin. It is used to define a name and version of a plugin.

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

#define PJSPLG_DEFINE(name, version)
```

Table I.1 Arguments of PJSPLG_DEFINE

Argument	Description
name	Arbitrary character string representing a plugin name
version	Arbitrary character string representing a version of a plugin

The name and version defined by this macro are output to the log file of the job scheduler.

When creating a plugin library, you need to call PJSPLG_DEFINE() from one of the source code files (c files) to embed the plugin name and plugin into the library as shown below.

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

PJSPLG_DEFINE("custom_sched", "1.0.0");
```

I.2 Functions

This section describes functions used by job selection plugins.

I.2.1 plugin_init()

plugin_init() is a function that initializes the job selection plugin.

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

pjsplg_result_t plugin_init(PjsplgManager_t *mng_p);
```

This function is called when the job selection plugin is loaded.
You can perform processing for registering the class by calling a function of the scheduler plugin manager from the inside of this function.
Every job selection plugin must implement this function.

Table I.2 Argument of plugin_init()

Argument	Description
mng_p	Scheduler plugin manager

Whether the initialization of the job selection plugin was successful is returned as a return value of this function.

Table I.3 Return values of plugin_init()

Return value	Description
PJSPLG_OK	Initialization success
PJSPLG_ERR	Initialization failure

If failure is returned, starting of the job scheduler process fails.

I.2.2 plugin_fini()

plugin_fini() is a function that discards the job selection plugin.

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

pjsplg_result_t plugin_fini(PjsplgManager_t *mng_p);
```

This function is called when the job selection plugin is unloaded. The unloading processing is performed when the job scheduler process ends. At the time of call of this function, it is guaranteed that all the instances created by the relevant job selection plugin have been discarded.
You can perform deletion processing of the class by calling a function of the scheduler plugin manager from the inside of this function.
Every job selection plugin must implement this function.

Table I.4 Argument of plugin_fini()

Argument	Description
mng_p	Scheduler plugin manager

Whether or not the job selection plugin was discarded is returned as a return value of this function.

Table I.5 Return values of plugin_fini()

Return value	Description
PJSPLG_OK	Discarding success
PJSPLG_ERR	Discarding failure

Regardless that either of the value is returned, the job scheduler continues the end processing. However, the return value is output to the log file of the job scheduler.