FUJITSU

FUJITSU Software
Technical Computing Suite V4.0L20

Job Operation Software
API user's Guide for Power API

J2UL-2545-01ENZ0(02)
March 2022

Preface

Purpose of This Manual

This manual describes how to use the Power API provided by the Job Operation Software of Technical Computing Suite.

Intended Readers

This manual is intended for administrators who operate and manage jobs and the end users who actually use the Power API. The manual assumes readers have the following knowledge:

- Basic Linux knowledge
- Knowledge of usage of job, obtained from the "Job Operation Software End-user's Guide"
- Knowledge of the Sandia Power API specifications

Organization of This Manual

This manual is organized as follows.

Chapter 1 What is the Power API?

This chapter provides an overview of the Power API.

Chapter 2 How to Use the Power API

This chapter describes how to create, compile, and execute a Power API program.

Chapter 3 Items Specific to a System Consisting of the FX or PRIMERGY Server

This chapter describes items specific to the Power API in the system.

Appendix A Functions Available in the Job Operating Software

This appendix describes the range of supported Power API functions on the Job Operation Software.

Appendix B Sample Programs

This appendix describes sample programs that use the Power API.

Notation Used in This Manual

Representation of Units

The following table lists the prefixes representing units in this manual. As a rule, disk size is expressed as a power of 10, and memory size is expressed as a power of 2. Be careful about specifying sizes 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}

Representation 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.

Representation of the Path Name of a Command

In operation examples, a command in the /bin, /usr/bin, /sbin, or /usr/sbin directory may not be indicated by an absolute path in some cases.

Symbols in This Manual

This manual uses the following symbols.

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

The See symbol indicates the reference source of detailed information.

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
March 2022, Version 1.2	J2UL-2545-01ENZ0(02)
June 2020, Version 1.1	J2UL-2545-01ENZ0(01)
February 2020, First version	J2UL-2545-01ENZ0(00)

Copyright

Update History

Changes	Location	Version
Added description of electrical energy on FX server.	3.2.2	1.2
Changed power measurement and control target of Power API.	Chapter 1	1.1
Changed query function for upper of lower limit of power knob.	2.1.4, 3.2.3	
Changed cross compiler command name.	2.2, B.1	
Clarified the behavior of the function when retrieving non existing Object.	3.2.1	
Clarified the data type which varies on Attribute.	3.2.2, 3.2.3, 3.3.2	
Changed sample program code.	B.4	

Contents

Chapter 1 What is the Power API? 1

Chapter 2 How to Use the Power API 2
2.1 How to Create a Power API Program 2
2.1.1 Power API Program Flow 2
2.1.2 Initialization 3
2.1.3 Getting Objects 3
2.1.4 Power Measurement and Control 4
2.1.5 Termination Process 4
2.2 Compile Method 4
2.3 Execution Method 5

Chapter 3 Items Specific to a System Consisting of the FX or PRIMERGY Server 6
3.1 Initialization Function 6
3.2 Power Control and Measurement on the FX Server 6
3.2.1 Expression of the Object Tree Structure of the FX Server 6
3.2.2 Power Measurement on the FX Server 8
3.2.3 Power Control on the FX Server 9
3.3 Power Control and Measurement on the PRIMERGY Server 10
3.3.1 Expression of the Object Tree Structure of the PRIMERGY Server 11
3.3.2 Power Measuring on the PRIMERGY Server 12
3.3.3 Power Control on the PRIMERGY Server 12
3.4 Statistical Information 12

Appendix A Functions Available in the Job Operating Software 13

Appendix B Sample Programs 16
B.1 Sample Programs Included in the Package 16
B.2 Example of a Program of Electric Energy Measurement 16
B.3 Example of a Program of Power Control 17
B.4 Example of a Program to Get Statistical Information 18

Chapter 1 What is the Power API?

The Sandia Power API (Power Application Programming Interface) defines a library interface proposed by Sandia National Laboratories to measure and control power.
The Sandia Power API provides the necessary measurement and control capabilities for every HPC (High Performance Computing) system.

The Power API used with the Job Operation Software can run on the compute nodes of the FX server and PRIMERGY server. End users can perform the following operations without special privileges by executing a program that uses the Power API as a job.

Table 1.1 Operations Executable by the Power API

Model	Compute Node Power Measurement	Compute Node Power Control
FX server	Yes	Yes
PRIMERGY server	Yes	No

Yes: Executable, No: Not executable

The Power API can be executed from an application program written in C. In Job Operation Software, the Power API can be also used from Fortran. Sample programs are included in the package for this purpose. In this manual, a program that uses the Power API is called a Power API program.

The Power API available in this system is based on the Power API specification of version 2.0 that Sandia National Laboratories made open to the public. The following operations are available.

- End users can use the provided functions in "Appendix A Functions Available in the Job Operating Software."
- Power can be measured and controlled on the compute nodes which the Power API is executed.

See

- For details on the Sandia Power API, see the webpages of Sandia National Laboratories.

  http://powerapi.sandia.gov/

- For details on sample programs, see "Appendix B Sample Programs."

Note

The Power API is not available in KVM mode in the job execution environment. For more information about KVM mode, see "Job Operation Software End-user's Guide."

Chapter 2 How to Use the Power API

This chapter describes how to use the Power API from a job.

Use the Power API in the following step on a system consisting of the FX server or PRIMERGY server.

1. Create a program that uses the Power API.
2. Compile the Power API program.
3. Submit and execute the created program as a job.

The following sections describe each step of the procedure.

2.1 How to Create a Power API Program

This section describes how to create a Power API program so that end users can measure and control power on individual hardware.

See

For details on items specific to the system, see "Chapter 3 Items Specific to a System Consisting of the FX or PRIMERGY Server."

For details on the Power API specifications, see the webpages of Sandia National Laboratories.

http://powerapi.sandia.gov/

2.1.1 Power API Program Flow

The basic steps of a Power API program are as follows.

1. Initialize
2. Get the target Object for power measurement and control.

   In the Power API, hardware such as a CPU and memory is called "Object." For details on acquisition of Objects, see "2.1.3 Getting Objects."
3. Measure and control power to the target program section.

   In the Power API, Attributes are used to express the power measurement and control types that are available for Objects. End users can measure and control power on the target hardware by specifying Objects and Attributes. For details on power measurement and control, see "2.1.4 Power Measurement and Control."
4. Perform the termination process.

The following example shows a power measurement program written based on the above steps.

```
#include <stdio.h>
#include "pwr.h"


int main()
{
    PWR_Cntxt cntxt = NULL;
    PWR_Obj obj = NULL;
    double energy0 = 0.0;
    double energy1 = 0.0;
    PWR_Time ts0 = 0;
    PWR_Time ts1 = 0;
    double ave_power = 0.0;

    // 1. Initialize Power API
    PWR_CntxtInit(PWR_CNTXT_DEFAULT, PWR_ROLE_APP, "app", &cntxt);
```

```
    // 2. Get Object (In this step, get an Object that indicates the entire compute node.)
    PWR_CntxtGetObjByName(cntxt, "plat.node", &obj);

    // 3. Get electric energy at start and end points of program section to be measured,
    //    and calculate average power in program section using obtained electric energy
    //    (In this step, PWR_ATTR_ENERGY is specified as an Attribute.)
    PWR_ObjAttrGetValue(obj, PWR_ATTR_ENERGY, &energy0, &ts0);

    ... // Arbitrary program section

    PWR_ObjAttrGetValue(obj, PWR_ATTR_ENERGY, &energy1, &ts1);

         // Calculate the average electric power from the two electric power quantities,
         // energy-0 and energy-1.

    // 4. Terminate processing of Power API
    PWR_CntxtDestroy(cntxt);

    return 0;
}
```

The following sections describe each step of the procedure.

2.1.2 Initialization

End users initialize the Power API by calling the PWR_CntxtInit() initialization function.

The following identifiers are specified in arguments for the initialization function.

- PWR_CntxtType *type*

  This identifier specifies the function to use the Power API.

- PWR_Role *role*

  This identifier indicates the privileges of the Power API user. The available range of the function that can be used by the specified PWR_CntxtType varies depending on specified privileges.

For details on PWR_CntxtType and PWR_Role that can be specified in this system, see "3.1 Initialization Function."

2.1.3 Getting Objects

In the Sandia Power API, the hardware whose power is measured and controlled is called an Object. The system targeted by the Power API is expressed by an Object tree structure. An end user can retrieve the Object by specifying its unique name using the PWR_CntxtGetObjByName function.

Table 2.1 Function to Get an Object by Specifying a Unique Name

Function	Description
PWR_CntxtGetObjByName	Gets the Object with the specified unique name.

For details on Objects and Object trees in this system, see "3.2.1 Expression of the Object Tree Structure of the FX Server" and "3.3.1 Expression of the Object Tree Structure of the PRIMERGY Server."

2.1.4 Power Measurement and Control

End users can measure and control power by specifying Objects and Attributes.

The following functions are used for power measurement and control.

Table 2.2 Functions to Get and Set Attributes

Function	Description
PWR_ObjAttrGetValue	Gets a single Attribute value for an Object.
PWR_ObjAttrSetValue	Sets a single Attribute value to an Object.

To measure power, specify PWR_ATTR_ENERGY in the PWR_ObjAttrGetValue function. To change the power control frequency, specify PWR_ATTR_FREQ in the PWR_ObjAttrSetValue function. To get the currently set frequency, specify PWR_ATTR_FREQ in the PWR_ObjAttrGetValue function.

Note

On FX servers, the per-BoB power capping function, which is designed to reduce power consumption, may limit CPU frequency changes (Conditions are set by the administrator). In this state, the Power API cannot change the CPU frequency and the function returns the error code PWR_RET_FAILURE.
However, the upper and lower limits of the frequency (Attribute PWR_MD_MAX and PWR_MD_MIN that can be retrieved with the PWR_ObjAttrGetMeta function) will be the values before this limit is activated. See the Sandia National Laboratories Web page for information on how to retrieve these values using the PWR_ObjAttrGetMeta function.

See

- The Attributes that can be specified vary depending on the target Object. For the correspondence relationship of Objects and Attributes that can be specified in this system, see "3.2 Power Control and Measurement on the FX Server" and "3.3 Power Control and Measurement on the PRIMERGY Server" based on the compute node model.
- In the Power API, there are some functions that collectively set and get multiple Objects and Attributes. For details on the Power API specifications, see the webpages of Sandia National Laboratories.

2.1.5 Termination Process

The end user can terminate the Power API using the following function.

Table 2.3 Termination Function

Function	Description
PWR_CntxtDestroy	Terminates the Power API.

2.2 Compile Method

To create an execution file for a Power API program, an end user needs to compile the Power API program on the login node. When compiling the program, use the header file of the Power API, and link the Power API library.

The Power API header file is pwr.h, and the library is libpwr.so.
The following table lists the locations of the Power API header files and libraries. The locations of the header file and library vary depending on the compute node on which the Power API is executed.

Table 2.4 Locations of Header Files and Libraries

File Type	Location
Power API header file for FX server	/opt/FJSVtcs/pwrm/aarch64/include/
Power API header file for PRIMERGY server	/opt/FJSVtcs/pwrm/x86_64/include/

File Type	Location
Power API library for FX server	/opt/FJSVtcs/pwrm/aarch64/lib64/
Power API library for PRIMERGY server	/opt/FJSVtcs/pwrm/x86_64/lib64/

The following command line shows an example to compile the Power API program sample.c for the FX server using a cross compiler (the command name is fccpx).

```
$ fccpx sample,c -L /opt/FJSVtcs/pwrm/aarch64/lib64 \
-I /opt/FJSVtcs/pwrm/aarch64/include -lpwr
```

2.3 Execution Method

An end user writes a job script to execute a Power API program and submit a job.

For example, the job.sh job script to execute a.out of a Power API program is written as follows.

```
#!/bin/bash
#PJM -L node=1

./a.out
```

Submit the job as follows:.

```
$ pjsub job.sh
```

See

For details on how to write a job script of the Job Operation Software and how to submit jobs, see "Job Operation Software End-user's Guide."

Chapter 3 Items Specific to a System Consisting of the FX or PRIMERGY Server

This chapter describes the items specific to the Power API in a system consisting of the FX or PRIMERGY server.

3.1 Initialization Function

This section describes PWR_CntxtType and PWR_Role that can be specified in arguments of the PWR_CntxtInit function to initialize a Power API program.

The following types can be specified in the PWR_CntxtType type argument.

Table 3.1 PWR_CntxtType That Can be Specified

PWR_CntxtType	Description
PWR_CNTXT_DEFAULT	Power API standard functions can be used.
PWR_CNTXT_FX1000 [FX]	A function that is extended for the FX server can be used.

The Power API used in this system originally defines the Attributes corresponding to power measurement and control specific to the FX server by extending the Attributes of the Power API. End users can use the Attributes specific to the FX server by specifying PWR_CNTXT_FX1000 at initialization.

The following role can be specified in the PWR_Role role argument.

Table 3.2 PWR_Role That Can be Specified

PWR_Role	Description
PWR_ROLE_APP	Refers to application users.

3.2 Power Control and Measurement on the FX Server

This section describes the Object tree structure of the FX server and also power measurement and control that can be performed on the FX server.

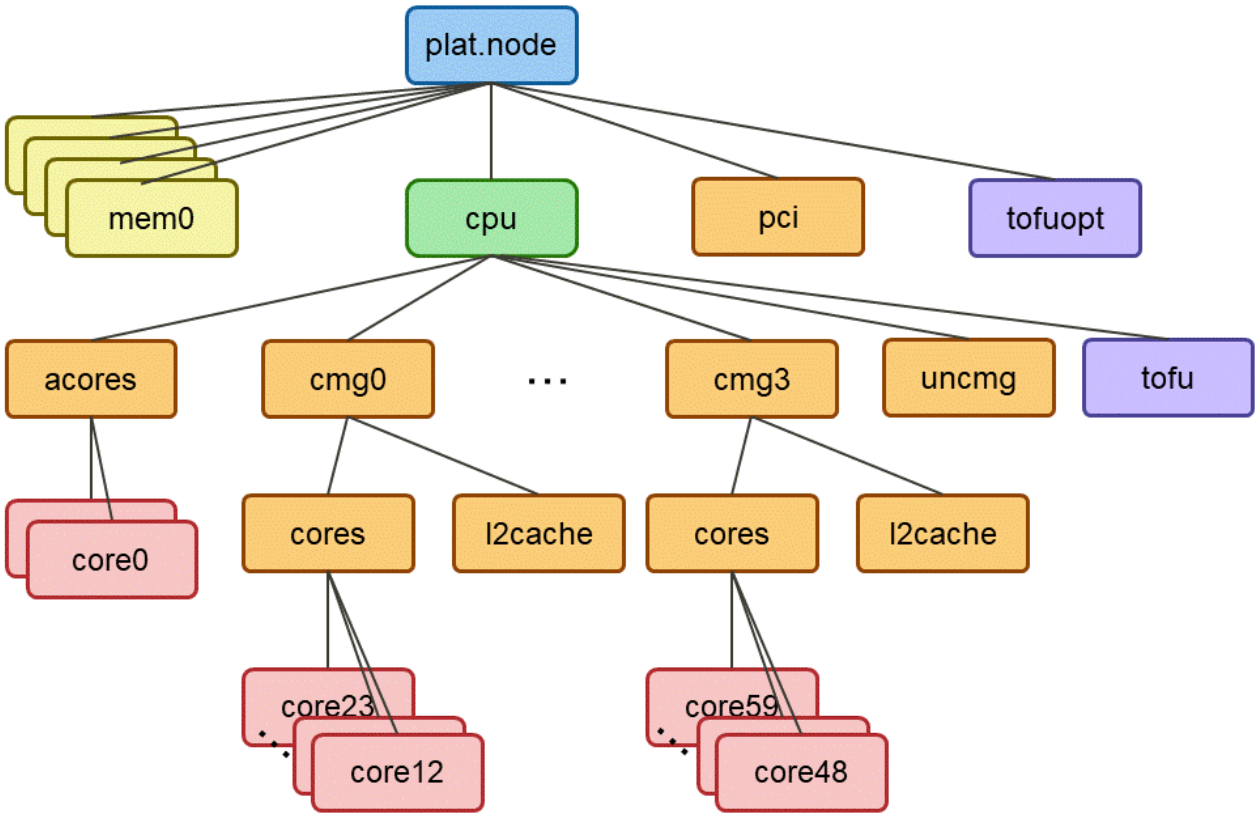
3.2.1 Expression of the Object Tree Structure of the FX Server

The Power API on this system can get only Objects of the compute node on which The Power API is running. The Objects of other compute nodes cannot be obtained.

The compute node on which The Power API is running is expressed as a tree structure as shown in the following figure.

Each node indicates an individual Object.

Figure 3.1 Object Tree of the FX Server



In the above figure, the Objects shown in the same color indicate that they are the same PWR_ObjType.

The following table provides details on each Object.

Table 3.3 Objects of the FX Server

Unique Name	PWR_ObjType	Description
plat.node	PWR_OBJ_NODE	Entire node
plat.node.cpu	PWR_OBJ_SOCKET	CPU Socket
plat.node.mem*N* (*N* = 0 to 3)	PWR_OBJ_MEM	Memory
plat.node.pci	PWR_OBJ_POWER_PLANE	PCI express
plat.node.tofuopt	PWR_OBJ_NIC	Optical module
plat.node.cpu.uncmg	PWR_OBJ_POWER_PLANE	Parts other than assistant core group in CPU, CMG, and Tofu
plat.node.cpu.acores (*)See note below for McKernel mode.	PWR_OBJ_POWER_PLANE	Assistant core group
plat.node.cpu.cmg*N* (*N* = 0 to 3)	PWR_OBJ_POWER_PLANE	CMG
plat.node.cpu.tofu	PWR_OBJ_NIC	Tofu
plat.node.cpu.acores.core*L* - Compute node: *L*=0,1 - Compute node and I/O node: *L*=0 to 3 (*)See note below for McKernel mode.	PWR_OBJ_CORE	Assistant core

Unique Name	PWR_ObjType	Description
plat.node.cpu.cmg*N*.cores	PWR_OBJ_POWER_PLANE	Compute core group in CMG
plat.node.cpu.cmg*N*.cores.l2cache	PWR_OBJ_POWER_PLANE	L2 cache
plat.node.cpu.cmg*N*.cores.core*M* The range of M for N is as follows: - When *N*=0, *M*=12 - 23 - When *N*=1, *M*=24 - 35 - When *N*=2, *M*=36 - 47 - When *N*=3, *M*=48 - 59 (*)See note below for McKernel mode.	PWR_OBJ_CORE	Compute core

Caution must be exercised so that a unique Object name used by the Power API is expressed by separating the name of each node that is passed when tracing a tree, starting from the tree root (plat.node).

For example, when an Object of a core under cmg0 in Figure 3.1 Object Tree of the FX Server is expressed with a unique name, the name is plat.node.cmg0.cores.

Note

- An Object with a unique name of plat.node.cpu.acores.core*L* (L=2 or 3) exists only in the compute and I/O node, not in the compute node. Specifying these objects in the PWR_CntxtGetObjByName function for a compute node returns error.
- In McKernel mode, objects for assistant cores and assistant core group cannot be retrieved.
  And, in McKernel mode, the values *N* and *M* for the unique name of the compute core Object (plat.node.cpu.cmg*N*.cores.core*M*) are:
  - When *N*=0, *M*=0 - 11
  - When *N*=1, *M*=12 - 23
  - When *N*=2, *M*=24 - 35
  - When *N*=3, *M*=36 - 47

  For more information on McKernel mode, see "Job Operation Software End-user's Guide."

3.2.2 Power Measurement on the FX Server

The FX server can measure the following two types of electrical energy (Unit: J).

Electrical Energy	Description
Estimated electrical energy (IDEAL)	Electrical energy that is estimated based on the number of CPU instructions issued and other information. This value does not take into account variations in electrical energy due to individual differences in computing nodes, and the estimated electrical energy is the same for the same job. The value is estimated by hardware and updated every millisecond.
Measured electrical energy (MEASURED)	Electrical energy that the job actually consumed. This value takes into account variations in electrical energy due to individual differences in computing nodes and/or individual differences of data patterns to calculation, therefore the measured electrical energy is different for the same job. The value is collected by hardware from an electric energy measurement element and updated every 5 milliseconds.

The following table lists the Power API Attribute and whose data type corresponding to each type of electrical energy and Objects that can be measured.

Table 3.4 Attribute Corresponding to Each Type of Electrical Energy and Objects That Can be Measured

Electrical Energy	Attribute (data type)	Object That Can be Measured
Estimated electrical energy	PWR_ATTR_ENERGY (double)	Entire node Compute core group in CMG L2 cache Memory Tofu Part other than assistant core group in CPU, CMG, and Tofu Assistant core Optical module PCI Express
Measured electrical energy	PWR_ATTR_MEASURED_ENERGY (double)	Entire node

Note

- The estimated electrical energy of an <entire node> on the FX server can be calculated using the following calculation formula:

```
<Compute core group in CMG> + <L2 cache> + <Memory> + <Tofu> + <parts other than assistant core in CPU, CMG, and Tofu>
```

  The <assistant core> is used for processes other than jobs. The estimated electrical energy of an <optical module> and <PCI Express> is changed based on the node to which a job is assigned. Therefore, these three estimated electric energies are not included in the calculation of the estimated electrical energy of an entire node.

- The measured electrical energy of an <entire node> on the FX server includes <assistant core>, <optical module>, and <PCI Express>, which are configured differently for computing and I/O nodes.

- When the same job is performed on different nodes, the estimated electric energy for the <entire node> is the same however the measured electric energy varies depending on computing nodes. There may be a +/- 40% difference between two types of the electrical energy of <entire node> for same nodes.

- PWR_ATTR_MEASURED_ENERGY is an extended Attribute of the FX server. Estimated electrical energy can be measured only when the PWR_CntxtType type argument is specified in PWR_CNTXT_FX1000 at initialization.

3.2.3 Power Control on the FX Server

Power can be controlled on the FX server using the following power knob functions.

Table 3.5 Power Knob Functions Available on the FX Server

Power Knob Function	Description
Changing frequency	Controls the frequency of the CPU.
Memory access limit	Controls the bus usage rate between the memory access controller and memory.
Instruction issuance limit	Controls the number of instructions simultaneously processed by the core.
EXA only	Controls the number of pipes of the core that can be used by an instruction that uses a general-purpose register.
Eco mode, FLA only state	FLA only controls the number of pipes of the core that can use FP and SIMD registers. Eco mode increases the power reduction effect when FLA only is enabled.
Retention state	Determines whether or not to transit to a low power state (Retention state) when no process is running on the core.

The following table lists the Power API Attribute and whose data type corresponding to each power knob function, Objects available for the power knob function, and values that can be set.

Table 3.6 Attributes Corresponding to the Power Knob Function, Available Objects, and Values that Can be Set

Power Knob Function	Attribute (data type)	Available Object	Value That Can be Set
Changing frequency	PWR_ATTR_FREQ (double)	CPU Socket	Set in unit of Hz. 2200000000 2000000000 1600000000 Hardware configurable values may differ from those listed.
Memory access limit	PWR_ATTR_THROTTLING_STATE (uint64_t)	Memory	0: 100% bus usage rate 1: 90% bus usage rate 2: 80% bus usage rate 3: 70% bus usage rate 4: 60% bus usage rate 5: 50% bus usage rate 6: 40% bus usage rate 7: 30% bus usage rate 8: 20% bus usage rate 9: 10% bus usage rate
Instruction issuance number limit	PWR_ATTR_ISSUE_STATE (uint64_t)	Compute core	0: 4 instructions 1: 2 instructions
EXA only	PWR_ATTR_EX_PIPE_STATE (uint64_t)	Compute core	0: Use pipe A and pipe B 1: Use only pipe A
Eco mode, FLA only state	PWR_ATTR_ECO_STATE (uint64_t)	Compute core	0: Disable Eco mode and FLA only 1: Disable Eco mode and enable FLA only 2: Enable Eco mode and FLA only
Retention state	PWR_ATTR_RETENTION_STATE (uint64_t)	Compute core	0: Do not transition to Retention state 1: Transition to Retention state

The range that end users can set for power control is restricted by administrator settings. The following method can be used to know the value range that can be set for power control.

- Make an inquiry by specifying Attribute and Metadata in the PWR_ObjAttrGetMeta function of the Power API. Set Metadata as PWR_MD_MIN or PWR_MD_MAX to obtain the lowest or highest value of Attribute, respectively. The data type of return value depends on the Attribute, as shown in "Table 3.6 Attributes Corresponding to the Power Knob Function, Available Objects, and Values that Can be Set."

See

For details on the PWR_ObjAttrGetMeta function specifications, see the webpages of Sandia National Laboratories.

http://powerapi.sandia.gov/

Note

PWR_ATTR_THROTTLING_STATE, PWR_ATTR_ISSUE_STATE, PWR_ATTR_EX_PIPE_STATE, PWR_ATTR_ECO_STATE, and PWR_ATTR_RETENTION_STATE are extended Attributes of the FX server. Power can be controlled only when the PWR_CntxtType type argument is specified in PWR_CNTXT_FX1000.

3.3 Power Control and Measurement on the PRIMERGY Server

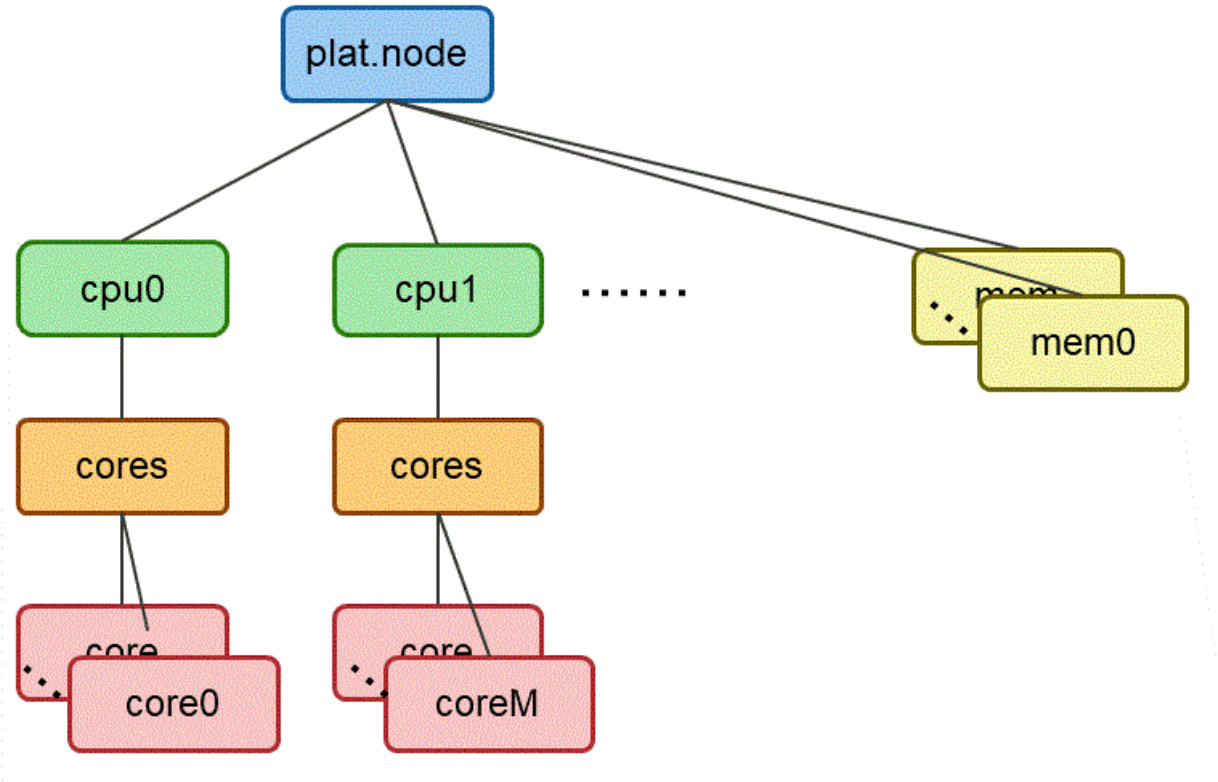
This section describes the Object tree structure of the PRIMERGY server and also power measurement and control that can be performed on the PRIMERGY server.

3.3.1 Expression of the Object Tree Structure of the PRIMERGY Server

The Power API on this system can get only Objects of the compute node on which The Power API is running. The Objects of other compute nodes cannot be obtained.

The compute node on which The Power API is running is expressed as a tree structure as shown in the following figure. Each node in the tree indicates an individual Object.

Figure 3.2 Object Tree of the PRIMERGY Server



In the above figure, the Objects shown in the same color indicate that they are the same PWR_ObjType.

The following table provides details on each Object.

Table 3.7 Objects of the PRIMERGY Server

Unique Name	PWR_ObjType	Description
plat.node	PWR_OBJ_NODE	Entire node
plat.node.cpu*N* (*N* is hardware-dependent.)	PWR_OBJ_SOCKET	CPU Socket
plat.node.mem*N* (*N* is hardware-dependent.)	PWR_OBJ_MEM	Memory
plat.node.cpu*N*.cores (*N* is hardware-dependent.)	PWR_OBJ_POWER_PLANE	CPU core group
plat.node.cpu*N*.cores.core*M* (*N* and *M* are hardware-dependent.)	PWR_OBJ_CORE	CPU core

Note that the unique name of Object used in the Power API is the name of the node that has traversed the tree from the root of the tree (plat.node) , separated by ".".

For example, when an Object of the cores under cpu0 in Object Tree of the PRIMERGY Compute Node is expressed with a unique name, the name is plat.node.cpu0.cores.

3.3.2 Power Measuring on the PRIMERGY Server

The PRIMERGY server can use the following electrical energy (Unit: J).

Electrical Energy	Description
Measured electrical energy	Actually measured electrical energy. Power that was actually consumed by a job can be obtained using measured electrical energy.

The following table lists the Power API Attribute and whose data type corresponding to each type of electrical energy and Objects that can be measured.

Table 3.8 Attribute Corresponding to Each Type of Electrical Energy and Objects That Can be Measured

Electrical Energy	Attribute (data type)	Objects That Can be Measured
Measured electrical energy	PWR_ATTR_ENERGY (double)	CPU Socket Memory CPU core group

Note

The objects whose electrical energy can be measured vary depending on the CPU mounted on the PRIMERGY server. The Power API function returns a return value indicating an error when the end user specifies an unmeasurable Object at electrical energy measurement. For details, see the webpages of Sandia National Laboratories.

3.3.3 Power Control on the PRIMERGY Server

Power cannot be controlled from a job on the PRIMERGY server.

3.4 Statistical Information

The Sandia Power API defines the functions that get statistical information. In this system, statistical information on power (PWR_ATTR_POWER) can be obtained for Objects that can use PWR_ATTR_ENERGY.

The following table lists the type of statistical information that can be obtained by the Power API and the corresponding Attribute. Power consumption is calculated periodically based on changes in electric energy and statistics are performed.

Table 3.9 Type of Statistical Information That Can be Obtained

Item	Attribute
Power	PWR_ATTR_POWER

See

- For details on the Objects available for PWR_ATTR_ENERGY, see "3.2.2 Power Measurement on the FX Server" or "3.3.2 Power Measuring on the PRIMERGY Server" according to the compute node type.
- For more information on statistics and the functions that retrieve them, see the webpages of Sandia National Laboratories.

  See also "B.4 Example of a Program to Get Statistical Information" that has a sample program of the statistical information functions.

Appendix A Functions Available in the Job Operating Software

End users can use the following functions with the Power API provided by the Job Operation Software.

Table A.1 List of Functions Available to End Users

Function Name	Availability	Description
PWR_CntxtInit	Yes	Initialization function of the Power API
PWR_CntxtDestroy	Yes	Termination function of the Power API
PWR_CntxtGetEntryPoint	Yes	Function to get the Objects of the Object tree root
PWR_ObjGetType	Yes	Function to get PWR_ObjType of the specified Object
PWR_ObjGetName	Yes	Function to inquire about the unique name of the specified Object
PWR_ObjGetParent	Yes	Function to get the parent Object of the specified Object
PWR_ObjGetChildren	Yes	Function to get the child Objects of the specified Object
PWR_CntxtGetObjByName	Yes	Function to get the Object corresponding to the specified unique name
PWR_GrpCreate	Yes	Function to generate a Group to store multiple Objects
PWR_GrpDestroy	Yes	Function to discard the specified Group
PWR_GrpAddObj	Yes	Function to add the specified Objects to the specified Group
PWR_GrpRemoveObj	Yes	Function to delete the specified Objects from the specified Group
PWR_GrpGetNumObjs	Yes	Function to get the number of Objects contained in the specified Group
PWR_GrpGetObjByIndx	Yes	Function to get the Objects corresponding to the specified element numbers from the specified Group
PWR_GrpDuplicate	Yes	Function to duplicate the specified Group
PWR_GrpUnion	Yes	Function to generate a Group by performing the union operation of two specified Groups
PWR_GrpIntersection	Yes	Function to generate a Group by performing the intersection operation of two specified Groups
PWR_GrpDifference	Yes	Function to generate a Group by performing the difference operation of two specified Groups
PWR_GrpSymDifference	Yes	Function to generate a Group by performing the symmetric difference operation of two specified Groups
PWR_CntxtGetGrpByName	No	Function to get the system-defined Groups corresponding to the specified unique names. This function cannot be used because Group is not defined in this system.
PWR_ObjAttrGetValue	Yes	Function to get the specified Attribute values for the specified Objects
PWR_ObjAttrSetValue	Yes [FX]	Function to set the specified Attribute values to the specified Objects. This function can be used only on the FX server.
PWR_StatusCreate	Yes	Function to generate the status to hold individual error information when collectively getting and setting multiple Objects and Attributes
PWR_StatusDestroy	Yes	Function to discard the specified status
PWR_StatusPopError	Yes	Function to get error information held in the specified status
PWR_StatusClear	Yes	Function to clear error information held in the specified status
PWR_ObjAttrGetValues	Yes	Function to collectively get multiple Attribute values for the specified Objects

Function Name	Availability	Description
PWR_ObjAttrSetValues	Yes [FX]	Function to collectively get multiple specified Attribute values for the specified Objects. This function can be used only on the FX server.
PWR_ObjAttrIsValid	Yes	Function to inquire whether or not the specified Attribute is available for the specified Object
PWR_GrpAttrGetValue	Yes	Function to collectively get multiple specified Attribute values for all Objects belonging to the specified Group
PWR_GrpAttrSetValue	Yes [FX]	Function to collectively set specified Attribute values to all Objects belonging to the specified Group. This function can be used only on the FX server.
PWR_GrpAttrGetValues	Yes	Function to collectively get multiple specified Attribute values from all Objects belonging to the specified Group
PWR_GrpAttrSetValues	Yes [FX]	Function to collectively set multiple specified Attribute values to all the Objects belonging to the specified Group. This function can be used only on the FX server.
PWR_ObjAttrGetMeta	Yes	Function to get detailed description information related to the specified Objects and Attributes
PWR_ObjAttrSetMeta	No	Function to set detailed setting information on the specified Objects and Attributes. End users are prohibited from setting detailed setting information in this system. For this reason, end users cannot use this function.
PWR_MetaValueAtIndex	Yes	Function to inquire about the values that can be set in the specified Objects and Attributes
PWR_ObjGetStat	No	Function to get statistical information on the specified Objects and Attributes from the data accumulated in the system. This function is not supported by the Power API on this system.
PWR_GrpGetStats	No	Function to get statistical information on each Object from data accumulated in the system for all Objects and Attributes belonging to the specified Group. This function is not supported by the Power API on this system.
PWR_ObjCreateStat	Yes	Function to generate objstat to collect statistical information on the specified Objects and Attributes in real time
PWR_GrpCreateStat	Yes	Function to generate grpstat to collect statistical information on each Object in real time for all Objects and Attributes belonging to the specified Group
PWR_StatStart	Yes	Function to start collecting the specified objstat or grpstat statistical information
PWR_StatStop	Yes	Function to stop collecting the specified objstat or grpstat statistical information
PWR_StatClear	Yes	Function to clear the collected data of the specified objstat or grpstat statistical information
PWR_StatGetValue	Yes	Function to get statistical values from the specified objstat
PWR_StatGetValues	Yes	Function to get the statistical value of each Object included in the specified grpstat
PWR_StatGetReduce	Yes	Function to perform a reduction calculation for the statistical value of each Object included in the specified grpstat
PWR_GrpGetReduce	No	Function to get statistical information on each Object from data accumulated in the system for all Objects and Attributes belonging to the specified Group, and to perform a reduction calculation. This function is not supported by the Power API on this system.
PWR_StatDestroy	Yes	Function to discard the specified stat
PWR_GetMajorVersion	Yes	Function to get the major version of the Power API
PWR_GetMinorVersion	Yes	Function to get the minor version of the Power API

Function Name	Availability	Description
PWR_GetReportByID	No	Function to get statistical information on the specified Attributes. Due to the Sandia Power API specifications, end users cannot use this function.
PWR_StateTransitDelay	No	Function to get the time taken for a transition between the two specified power management-related states of the specified Object. This function is not supported by the Power API of this system.
PWR_AppHintCreate	No	Function to create application tuning tip information for a program section. This function is not supported by the Power API of this system.
PWR_AppHintDestroy	No	Function to discard tuning tip information. This function is not supported by the Power API of this system.
PWR_AppHintStart	No	Function to report to the OS that the program process that is the target of tuning tip information has started. This function is not supported by the Power API of this system.
PWR_AppHintStop	No	Function to report to the OS that the program process that is the target of tuning tip information has ended. This function is not supported by the Power API of this system.
PWR_AppHintProgress	No	Function to indicate the calculation progress rate in the specified program section. This function is not supported by the Power API of this system.
PWR_SetSleepStateLimit	No	Function to set the deepest sleep state to which the OS is allowed to transition. This function is not supported by the Power API of this system.
PWR_WakeUpLatency	No	Function to get the transition time for recovery from the specified sleep mode. This function is not supported by the Power API of this system.
PWR_RecommendSleepState	No	Function to get the deepest sleep state within td by the Power API of this system.
PWR_SetPerfState	No	Function to set the specified Object the specified recovery transition time. This function is not supported the performance level defined for Objects. This function is not supported by the Power API of this system.
PWR_GetPerfState	No	Function to get the current performance level of the specified Object. This function is not supported by the Power API of this system.
PWR_GetSleepState	No	Function to get the current sleep state of the specified Object. This function is not supported by the Power API of this system.

Yes: Available, No: Not available

If a function that is not available is executed, an error is returned.

See

For details on each function, see the webpages of Sandia National Laboratories.

http://powerapi.sandia.gov/

Appendix B Sample Programs

This appendix describes sample programs that use the Power API.

B.1 Sample Programs Included in the Package

Two types of sample programs use the Power API: sample programs in C language and sample program in Fortran language. Sample programs in each language are placed in the following directory under the login node:

- Sample programs in C language

```
/usr/src/FJSVtcs/pwrm/powerapi/c/
```

- Sample programs in Fortran language

```
/usr/src/FJSVtcs/pwrm/powerapi/fortran/
```

The following sample programs are placed in individual directories.

Table B.1 Sample Programs Contained in Directories

Sample Program	File Name (C Language)	File Name (Fortran Language)
Electrical energy measurement	pwrget.c	pwrget.f03
Power control	pwrset.c	pwrset.f03
Statistical information acquisition	pwrstat.c	pwrstat.f03
Electrical energy measurement of multiple Objects	pwrget_multi.c	pwrget_multi.f03
Power control of multiple Objects	pwrset_multi.c	pwrset_multi.f03
Definition of Power API functions, variables, and types in Fortran (corresponding to header file)		pwrf.f03
		pwrtypesf.f03

End users change and compile these sample programs as appropriate after copying them to their own directory on the login node.

When compiling sample programs in C language, end users specify the file name of the sample program to be compiled first. For details on the necessary header files and libraries for compilation, see "2.2 Compile Method." When compiling sample programs in Fortran language, end users specify the file name of the sample program to be compiled and pwrf.f03 first. Be sure that pwrf.f03 comes first in the order of specifying files.

The following example shows the command line for compiling a sample program (pwrget.f03) for electrical energy measurement using a cross compiler (the command name is frtpx). For details on specifying the necessary Power API libraries when compiling, see "2.2 Compile Method."

```
$ frtpx pwrf.f03 pwrget.f03 -L /opt/FJSVtcs/pwrm/aarch64/lib64 -lpwr
```

The following sections, the following information is written in a sample program in C language:

- Electric energy measurement
- Power control
- Statistical information acquisition

B.2 Example of a Program of Electric Energy Measurement

The following example is a program to get an Object by specifying the unique name of the Object on the FX server, and to get electrical energy.

```
#include <stdio.h>
#include <unistd.h>
#include "pwr.h"
```

```
int main()
{
    PWR_Cntxt cntxt = NULL;
    PWR_Obj obj = NULL;
    int rc;
    double energy1 = 0.0;
    double energy2 = 0.0;
    double ave_power = 0.0;
    PWR_Time ts1 = 0;
    PWR_Time ts2 = 0;

    // Get context of Power API
    rc = PWR_CntxtInit(PWR_CNTXT_DEFAULT, PWR_ROLE_APP, "app", &cntxt);
    if (rc != PWR_RET_SUCCESS) {
        printf("CntxtInit Failed\n");
        return 1;
    }

    // Get the Object for which the electric energy is measured
    rc = PWR_CntxtGetObjByName(cntxt, "plat.node", &obj);
    if (rc != PWR_RET_SUCCESS) {
        printf("CntxtGetObjByName Failed\n");
        return 1;
    }

    // Get the estimated electric energy of Object
    rc = PWR_ObjAttrGetValue(obj, PWR_ATTR_ENERGY, &energy1, &ts1);
    if (rc != PWR_RET_SUCCESS) {
        printf("ObjAttrGetValue Failed (rc = %d)\n", rc);
        return 1;
    }

    sleep(3);

    rc = PWR_ObjAttrGetValue(obj, PWR_ATTR_ENERGY, &energy2, &ts2);
    if (rc != PWR_RET_SUCCESS) {
        printf("ObjAttrGetValue Failed (rc = %d)\n", rc);
        return 1;
    }

    // Calculate the average power from the electric energy of the two measurement points
    ave_power = (energy2 - energy1) / ((ts2 - ts1) / 1000000000.0);
    printf("ave_power = %lf\n", ave_power);

    // Destroy the context
    PWR_CntxtDestroy(cntxt);

    return 0;
}
```

B.3 Example of a Program of Power Control

The following example is a program to get an Object by specifying the unique name of the Object on the FX server and to set a frequency.

```
#include <stdio.h>
#include "pwr.h"

int main()
{
    PWR_Cntxt cntxt = NULL;
    PWR_Obj obj = NULL;
```

```
    int rc;
    double freq = 0.0;

    // Get context of Power API
    rc = PWR_CntxtInit(PWR_CNTXT_DEFAULT, PWR_ROLE_APP, "app", &cntxt);
    if (rc != PWR_RET_SUCCESS) {
        printf("CntxtInit Failed\n");
        return 1;
    }

    // Get the Object for which the frequency is set
    rc = PWR_CntxtGetObjByName(cntxt, "plat.node.cpu", &obj);
    if (rc != PWR_RET_SUCCESS) {
        printf("CntxtGetObjByName Failed\n");
        return 1;
    }

    // Specify frequency to set
    freq = 2000000000.0;
    // Set the frequency to Object
    rc = PWR_ObjAttrSetValue(obj, PWR_ATTR_FREQ, &freq);
    if (rc != PWR_RET_SUCCESS) {
        printf("ObjAttrSetValue Failed (rc = %d)\n", rc);
        return 1;
    }

    // Destroy the context
    PWR_CntxtDestroy(cntxt);

    return 0;
}
```

B.4 Example of a Program to Get Statistical Information

The following example is a program to get the minimum value of the power used in a section on the FX server.

```
#include <stdio.h>
#include <unistd.h>
#include "pwr.h"

int main()
{
    PWR_Cntxt cntxt = NULL;
    PWR_Obj obj = NULL;
    PWR_Stat stat = NULL;
    int rc;
    double min_power = 0.0;
    PWR_TimePeriod period = { PWR_TIME_UNINIT, PWR_TIME_UNINIT, PWR_TIME_UNINIT };

    // Get context of Power API
    rc = PWR_CntxtInit(PWR_CNTXT_DEFAULT, PWR_ROLE_APP, "app", &cntxt);
    if (rc != PWR_RET_SUCCESS) {
        printf("CntxtInit Failed\n");
        return 1;
    }

    // Get the Object for which statistics are to be retrieved
    rc = PWR_CntxtGetObjByName(cntxt, "plat.node", &obj);
    if (rc != PWR_RET_SUCCESS) {
        printf("CntxtGetObjByName Failed\n");
        return 1;
    }
```

```
    // Get Statistical Object to obtain minimum power value
    rc = PWR_ObjCreateStat(obj, PWR_ATTR_POWER, PWR_ATTR_STAT_MIN, &stat);
    if (rc != PWR_RET_SUCCESS) {
        printf("ObjCreateStat Failed (rc = %d)\n", rc);
        return 1;
    }

    // Start to get statistics
    rc = PWR_StatStart(stat);

    sleep(3);

    // Stop to get statistics
    rc = PWR_StatStop(stat);

    // Get amount of statistics
    rc = PWR_StatGetValue(stat, &min_power, &period);
    if (rc != PWR_RET_SUCCESS) {
        printf("StatGetValue Failed (rc = %d)\n", rc);
        return 1;
    }

    printf("minimum power : %lf\n", min_power);
    printf("start : %lu\n", period.start);
    printf("stop : %lu\n", period.stop);
    printf("instant : %lu\n", period.instant);

    // Destroy statistical Object
    PWR_StatDestroy(stat);

    // Destroy context
    PWR_CntxtDestroy(cntxt);

    return 0;
}
```