



***Katana  
Shinobi Library  
Specification***





# ***Table of Contents***

---

<b>Shinobi System Functions .....</b>	<b>SLS-1</b>	
sbInit.....Initialize SHINOBI library .....	SLS-2	
sbExitSystem.....Terminate system .....	SLS-4	
<b>Shinobi File System Functions .....</b>	<b>SLS-5</b>	
Outline .....	SLS-5	
Data Specifications .....	SLS-5	
Constants .....	SLS-5	
Errors .....	SLS-5	
Data Structure .....	SLS-6	
Macros .....	SLS-6	
gdFsInit..... Initialize and mount file system .....	SLS-8	
gdFsCreateDirh .....	Get a handle to a directory .....	SLS-9
gdFsLoadDir .....	Get directory information .....	SLS-10
gdFsChangeDir .....	Change current directory .....	SLS-11
gdFsOpen .....	Open file .....	SLS-12
gdFsGetFileSize .....	Get file size .....	SLS-13
gdFsGetFileSctSize .....	Get file sector size .....	SLS-14
gdFsRead .....	Read file .....	SLS-15
gdFsSeek .....	Seek to sector .....	SLS-16
gdFsTell .....	Get current sector .....	SLS-17
gdFsClose .....	Close file .....	SLS-18

<b>Shinobi Memory Management Functions</b> .....	<b>SLS-19</b>
Outline .....	SLS-19
Function Specifications .....	SLS-19
Function List .....	SLS-19
syMallocInit .....	Get and declare memory area for memory management ..... SLS-20
syMalloc .....	Allocate memory ..... SLS-21
syFree .....	Release memory ..... SLS-22
<b>Shinobi Functions for Getting Peripheral Data</b> .....	<b>SLS-23</b>
Outline .....	SLS-23
Data Specifications .....	SLS-23
Structure Specifications .....	SLS-23
Device ID and Device Name .....	SLS-24
Buttons Supported by Device .....	SLS-25
Digital Button Data .....	SLS-26
Analog Axis Data .....	SLS-27
Data Acquiry Error .....	SLS-28
Function Specifications .....	SLS-28
Function List .....	SLS-28
pdGetPeripheral .....	Get controller button status ..... SLS-29
pdGetPeripheralError .....	Get controller error status ..... SLS-30
pdExecPeripheralServer .....	Get peripheral data from server ..... SLS-31



# ***1. Shinobi System Functions***

---

## **Outline**

Initializes the Katana hardware and makes the Shinobi library ready for use.

The Shinobi library contains several initialization functions. Calling these functions in a suitable sequence establishes the normal settings. The initialization function source code will be made public in future, allowing it to serve as a sample for library initialization.

---

◆ **Note:** The basic input/output parameters for Kamui remain the same in any of the above environments, except that some functions are unavailable in items a) and b) because the ARC1 has functions such as BumpMapping and Trilinear Filter, but the Holly does not.

---

The initialization source code is supplied in sbInit.c.

## sbInit

Initialize SHINOBI library

---

### Format

```
#include <Shinobi.h>
void sbInit( mode, frame, Count );
int mode
int frame
int count
```

### Parameters

#### mode

Screen mode (resolution)  
Specifies screen resolution.

#### frame

Frame buffer mode  
Sets the frame buffer color mode

#### count

Frame count  
Specifies the frame count in 1/60 second units.

### Return Value

None

### Example:

Initializes the hardware and makes the library ready for use. Internally, the function calls `njInitSystem` and specifies the screen resolution. The 2D clip area is set to the same size as the screen.

- Z clip is set to the range -1.0 – -60000.0.
- 3D screen projection distance is set to 500.
- Aspect ratio is set to 1.0 for X and Y.
- Color mode is set to `NJD_COLOR_MODE_NORMAL`.
- Frame count is set in 1/60 units.
- For example, if "2" is specified, frames are changed every 1/30 seconds.
- Frame change is performed with the `njWaitVSync` function.

The following screen modes are available.

Variable name	Screen mode
NJD_RESOLUTION_VGA	VGA
NJD_RESOLUTION_320x240_NTSCNI	NTSC non-interlaced 60 Hz
NJD_RESOLUTION_320x240_NTSCI	NTSC interlaced 30 Hz
NJD_RESOLUTION_640x240_NTSCNI	NTSC non-interlaced 60 Hz
NJD_RESOLUTION_640x240_NTSCI	NTSC interlaced 30 Hz
NJD_RESOLUTION_640x480_NTSCI	NTSC interlaced 30 Hz
NJD_RESOLUTION_320x240_PALNI	PAL non-interlaced 50 Hz
NJD_RESOLUTION_320x240_PALI	PAL interlaced 25 Hz
NJD_RESOLUTION_640x240_PALNI	PAL non-interlaced 50 Hz
NJD_RESOLUTION_640x240_PALI	PAL interlaced 25 Hz
NJD_RESOLUTION_640x480_PALI	PAL interlaced 25 Hz

The following frame buffer modes are available.

```
NJD_FRAMEBUFFER_MODE_RGB565
NJD_FRAMEBUFFER_MODE_RGB555
NJD_FRAMEBUFFER_MODE_ARGB4444
NJD_FRAMEBUFFER_MODE_ARGB1555
NJD_FRAMEBUFFER_MODE_RGB888
NJD_FRAMEBUFFER_MODE_ARGB8888
```

## Usage Example

```
sbInit( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1 );
```

Initializes the Shinobi library and sets the screen resolution to VGA (640x480), 1 frame, 1/60 seconds.

## Remarks

Always perform this at the start of a program.

Depending on the hardware configuration, some screen modes may not be available. Use this function in place of `njInitSystem`.

Because arguments are the same as for `njInitSystem`, it suffices to change only the function literal to achieve the same operation.

The source code for this function will be made public in a future release of the SDK. In the current release, this part is supplied in object file format, but the link object file differs, depending on whether the file system is used or not. Link the files as follows:

### File system is used

```
sbinit.obj
```

### File system is not used

```
sbinitn.obj
```

---

**Note:** We apologize for any temporary inconvenience caused by this approach.

---

# sbExitSystem

Terminate system

---

## Format

```
#inclue <shinobi.h>
```

```
sbExitSystem()
```

## Parameters

None

## Return Value

None

## Operation

Terminates the system.

Use in place of `njExitSystem`.



## ***2. Shinobi File System Functions***

---

### **Outline**

- Supplies GD-ROM access functions.
- Directory access
- File open/close
- File read (completion return)

### **Data Specifications**

#### **Constants**

##### *Seek mode*

GDFS\_SEEK\_SET

GDFS\_SEEK\_CUR

GDFS\_SEEK\_END

##### *Status*

GDFS\_STAT\_IDLE

GDFS\_STAT\_COMPLETE

GDFS\_STAT\_READ

GDFS\_STAT\_SEEK

GDFS\_STAT\_PLAY

GDFS\_STAT\_BUSY

GDFS\_STAT\_ERR

GDFS\_STAT\_FATAL

GDFS\_STAT\_UNDEF

#### **Errors**

GDFS\_ERR\_OK

GDFS\_ERR\_RESET

GDFS\_ERR\_INIT  
GDFS\_ERR\_DIRREC  
GDFS\_ERR\_CANTOPEN  
GDFS\_ERR\_NOTFOUND  
GDFS\_ERR\_NOHNDL  
GDFS\_ERR\_ILLHNDL  
GDFS\_ERR\_NOTDIR  
GDFS\_ERR\_DIROVER  
GDFS\_ERR\_BUSY  
GDFS\_ERR\_32ALIGN  
GDFS\_ERR\_SEEK  
GDFS\_ERR\_OFS  
GDFS\_ERR\_ILLTMODE  
GDFS\_ERR\_ILLREQ  
GDFS\_ERR\_READ  
GDFS\_ERR\_NOTREAD  
GDFS\_ERR\_RDTOUT  
GDFS\_ERR\_TOUT  
GDFS\_ERR\_EOF  
GDFS\_ERR\_TRAYOPEND  
GDFS\_ERR\_HWARE  
GDFS\_ERR\_MEDIA  
GDFS\_ERR\_RECOVER  
GDFS\_ERR\_PROTECT  
GDFS\_ERR\_ABORT  
GDFS\_ERR\_NOTREADY  
GDFS\_ERR\_UNITATTENT  
GDFS\_ERR\_FATAL  
GDFS\_ERR\_UNDEF

## Data Structure

GDFS	Filehandle
GDFS_DIRREC	Directory record handle

## Macros

GDFS\_WORK\_SIZE(x)

### Argument:

x ... Number of files to open simultaneously

### Meaning:

Required work buffer size

GDFS\_DIRREC\_SIZE(x)

**Argument:**

x ... Number of directory entries

**Meaning:**

Required directory record buffer size

---

**Note:** File system functions operate asynchronously. Make sure to wait on return value from `gdFsGetStat` of `GDFS_STAT_COMPLETE` before issuing other file operations.

---

## gdFsInit

## Initialize and mount file system

---

### Format

```
Sint32 gdFsInit(UInt32 max_open, void *gdFs_work, UInt32 max_dirent, void *gdFs_dirrec)
```

### Parameters

max_open	Number of files that can be opened simultaneously
gdFs_work	Work area pointer (supplied from user area)
max_dirent	Number of entries in current directory
gdFs_dirrec	Buffers for current directory (from user area)

### Return Value

GDFS_ERR_OK	... Normal end
Other	... Error

### Operation

Initializes the library.

### Example

```
UInt8 gdFswork[ GDFS_WORK_SIZE(8) ];
UInt8 gdFscurdir[ GDFS_DIRREC_SIZE(64) ];

gdFsInit(8, gdFswork, 64, gdFscurdir);
```

gdFs\_work must be 32-byte aligned. This function is executed from within the sbInit function. In the current SDK release, the source code of the sbInit function is not made public, but this is planned for a future release, in order to allow settings to be made when initializing the file system.

The default values in sbInit are as shown below.

- Number of files that can be opened simultaneously: 8
- Work area pointer: global variable gdFswork
- `UInt8 gdFswork [ GDFS_WORK_SIZE(8) ];`
- Number of entries for current directory: 1024
- Buffer for current directory: global variable gdFscurdir
- `UInt8 gdFscurdir[ GDFS_DIRREC_SIZE(1024) ];`

# gdFsCreateDirh

Get a handle to a directory

---

## Format

```
GDFS_DIRREC gdFsCreateDirhn(void *dirbuf, Sint32 max_dirent)
```

## Parameters

dirbuf... Directory buffer

max\_dirent... Number of entries in current directory

## Return Value

GDFS\_DIRREC.. Handle to the directory, Null if error

## Example:

```
Uint32 dirbuf[gdFsGetDirrecSize(64)/4];
GDFS_DIRREC gdFileHnd;

gdFileHnd = gdFsCreateDirhn((void *) dirbuf, 64);
ret = gdFsLoadDir("TEMP", gdFileHnd);

..open file..
```

## gdFsLoadDir

Get directory information

---

### Format

```
Uint32 gdFsLoadDir(Sint8 *dirname, GDFS_DIRREC gdfs_dirrec)
```

### Parameters

dirname ... Directory name

### Return Value

gdfs\_dirrec ... Directory record handle

(To current directory if NULL)

GDFS\_ERR\_OK ... Normal end

Other ... Error

### Function

Read directory record

### Example

```
/* Sample 1 */
/* Read MOVIE directory to g_dir */
Uin8 dirbuf[ GDFS_DIRREC_SIZE(64) ];
GDFS_DIRREC g_dir = (GDFS_DIRREC) dirbuf;

gdFsLoadDir("MOVIE", g_dir);

/* Sample 2 */
/* Move current directory to DATA directory */
gdFsLoadDir("DATA", NULL);
```

# gdFsChangeDir

Change current directory

---

## Format

```
Uint32 gdFsChangeDir(Sint8 *dirname)
```

## Parameters

dirname ... Directory name

## Return Value

GDFS\_ERR\_OK ... Normal end

Other ... Error

## Function

Change current directory

## Example

```
/* Move current directory to DATA directory */  
gdFsChangeDir("WORK");
```

## gdFsOpen

Open file

### Format

```
GDFS gdFsOpen(Sint8 *fname, GDFS_DIRREC gdfs_dirrec)
```

### Parameters

fname ... File name  
gdfs\_dirrec ... Directory record handle for file name search  
(Search current directory if NULL)

### Return Value

NULL ... Fail  
Other ... File handle

### Function

Open file

### Example

```
/* Sample 1 (accessing a file in the current directory) */  
GDFS gf;  
  
gf = gdFsOpen("A.BIN", NULL);  
  
/* Sample 2 (accessing a file in the current directory) */  
GDFS gf;  
Uint8 dirbuf[ GDFS_DIRREC_SIZE(64) ];  
GDFS_DIRREC g_dir;  
  
g_dir = gdFsCreateDirhn(dirbuf, 64);  
gdFsLoadDir("MOVIE", g_dir);  
gf = gdFsOpen("SMP.MOV", g_dir);
```

# gdFsGetFileSize

Get file size

---

## Format

```
Bool gdFsGetFileSize(GDFS gdfs, Uint32 *fsize);
```

## Parameters

gdfs ... File handle

\*fsize ... File size (bytes)

## Return Value

TRUE / FALSE

## Operation

Get file size

## Usage Example

```
GDFS gf;  
Uint32 flen;  
  
gf = gdFsOpen("TEST.BIN", NULL);  
gdFsGetFileSize(gdfs, &flen);
```

# gdFsGetFileSctSize

Get file sector size

---

## Format

```
Bool gdFsGetFileSctSize(GDFS gdfs, Uint32 *fsctsize);
```

## Parameters

gdfs ... File handle

## Return Value

\*fsctsize ... File sector size (bytes)

TRUE / FALSE: success of the operation

## Function

Get file sector size

## Example

```
GDFS gf;  
Uint32 flen;  
  
gf = gdFsOpen("TEST.BIN", NULL);  
gdFsGetFileSctSize(gdfs, &flen);
```

# gdFsRead

Read file

## Format

```
Sint32 gdFsRead(GDFS gdfs, Uint32 nsct, void *buf)
```

## Parameters

gdfs ... File handle  
nsct ... Number of sectors to read  
buf ... Storage buffer

## Return Value

GDFS\_ERR\_OK ... Normal end  
Other ... Error

## Function

Reads in sectors from a given file.

## Example

```
GDFS gf;  
Uint32 buf[32*2048/4];  
  
gf = gdFsOpen("TEST.BIN", NULL);  
gdFsRead(gf, 32, buf);  
gdFsClose(gf);
```

## gdFsSeek

Seek to sector

---

### Format

```
Sint32 gdFsSeek(GDFS gdfs, Uint32 sctno, Sint32 type)
```

### Parameters

gdfs ... File handle  
sctno ... Number of sector to read  
type ... GDFS\_SEEK\_SET  
          GDFS\_SEEK\_CUR  
                  GDFS\_SEEK\_END

### Return Value

GDFS\_ERR\_OK ... Normal end  
Other ... Error

### Function

Reads in sectors from a given file.

# gdFsTell

Get current sector

---

## Format

```
Sint32 gdFsTell(GDFS gdfs)
```

## Parameters

`gdfs` ... File handle

## Return Value

Current sector in file

## Function

Returns current file pointer sector.

# **gdFsClose**

Close file

---

## **Format**

```
void gdFsClose(GDFS gdfs);
```

## **Parameters**

gdfs ... File handle

## **Return Value**

None

## **Function**

Close file

## **Example**

```
GDFS gf;  
  
gf = gdFsOpen("TEST.BIN", NULL);  
gdFsClose(gf);
```



## ***3. Shinobi Memory Management Functions***

---

### **Outline**

The SHINOBI library performs memory management without relying on the operating system. Except for the initialization function, the conventions of the C standard malloc functions apply, only the name before the function is different.

### **Function Specifications**

#### **Function List**

<b>Function</b>	<b>Operation</b>
syMallocInit	Get and declare memory area for memory management
syMalloc	Allocate memory
syFree	Release memory*

# **syMallocInit**

Get and declare memory area for memory management

---

## **Format**

```
Void syMallocInit(Void *heap, Uint32 size);
```

## **Parameters**

heap ... Start address of memory area to be allocated

size ... Size of memory to be allocated

## **Return Value**

None

## **Function**

Specifies the memory area to be managed by the syMalloc library and initializes the library.

This function is executed from within the sbInit function.

In the current SDK release, the source code of the sbInit function is not made public, but this is planned for a future release, in order to allow changing the heap start address and size.

The default values in sbInit are as shown below.

- Heap area start address: end of section B
- Heap size: 0x00400000

# syMalloc

Allocate memory

---

## Format

```
Void *syMalloc(UInt32 size);
```

## Parameters

size ... Requested size (byte)

## Return Value

Pointer to allocated area

If NULL, allocation has failed

## Function

Same as standard C function `malloc(size)`.

# syFree

Release memory

---

## Format

```
Void syFree(void *ap);
```

## Parameters

Pointer to area to be released

## Return Value

None

## Function

Same as `free(*ap)`



## ***4. Shinobi Functions for Getting Peripheral Data***

---

### **Outline**

- Get Controller Data
- Get device data.
- To get device data, the function `pdGetPeripheral()` is used.
- To get the controller input, simply reference the members of this structure.

### **Data Specifications**

#### **Structure Specifications**

PDS_PERIPHERAL	Structure
Definition	<pre>typedef struct {     Uint32 id;     Uint32 support;     Uint32 on;     Uint32 off;     Uint32 press;     Uint32 release;     Uint16 r;     Uint16 l;     Sint16 x1;     Sint16 y1;     Sint16 x2;     Sint16 y2;     Sint8* name;     void* extend;     Uint32 old; } PDS_PERIPHERAL;</pre>

## Description

This structure serves for storing the control pad status.

## Members

id	Device ID
support	Button/lever support state
on	Digital button state
off	Digital button state (reversed)
press	Digital button down edge state
release	Digital button up edge state
r	Analog axis R value (0 - 255)
l	Analog axis L value (0 - 255)
x1	Analog axis X1 value (-128 - 127)
y1	Analog axis Y1 value (-128 - 127)
x2	Analog axis X2 value (-128 - 127)
y2	Analog axis Y2 value (-128 - 127)
name	Device name
extend	Extended data address (not used)
old	Reserved
Reference	<code>pdGetPeripheral()</code>

## Device ID and Device Name

The retrieved structure contains information about the device ID (type) and device name.

Because the format of the device ID according to the Maple bus specification is quite complex, it is not suitable for use by an application. In this library, it is simplified as follows.

Device	Device ID (id)	Device name string (name)
Standard control pad	PDD_DEV_CONTROLLER	Product name of the device

## 4. Shinobi Functions for Getting Peripheral Data

### Buttons Supported by Device

There are many different types of controller devices, which have various kinds of control buttons and levers, such as for example two cross-key pads, two analog channels, etc. The support member serves to check which kind of control the device use. The bit assign pattern is shown in the table below. 1 means that a button or lever is present, and 0 means that it is not present.

Controller devices must have a cross-key A, start button, A button and B button. By programming an application to use only these buttons, compatibility can be assured.

Button or Lever	Constants for Bit Position Specification
Cross-key A top	PDD_DEV_SUPPORT_KU
Cross-key A bottom	PDD_DEV_SUPPORT_KD
Cross-key A left	PDD_DEV_SUPPORT_KL
Cross-key A right	PDD_DEV_SUPPORT_KR
Cross-key B top	PDD_DEV_SUPPORT_KUB
Cross-key B bottom	PDD_DEV_SUPPORT_KDB
Cross-key B left	PDD_DEV_SUPPORT_KLB
Cross-key B right	PDD_DEV_SUPPORT_KRB
Start button	PDD_DEV_SUPPORT_ST
A button	PDD_DEV_SUPPORT_TA
B button	PDD_DEV_SUPPORT_TB
C button	PDD_DEV_SUPPORT_TC
X button	PDD_DEV_SUPPORT_TX
Y button	PDD_DEV_SUPPORT_TY
Z button	PDD_DEV_SUPPORT_TZ
D button	PDD_DEV_SUPPORT_TD
Analog axis R	PDD_DEV_SUPPORT_AR
Analog axis L	PDD_DEV_SUPPORT_AL
Analog axis X1	PDD_DEV_SUPPORT_AX1
Analog axis Y1	PDD_DEV_SUPPORT_AY1
Analog axis X2	PDD_DEV_SUPPORT_AX2
Analog axis Y2	PDD_DEV_SUPPORT_AY2

**Note:** The application should use the device ID and device specifications, or reference this button/lever information in order to ensure appropriate data handling.

### Digital Button Data

Digital button information comprises the four members "on", "off", "press", and "release", which should be used according to the application requirements. A button is assigned to each bit of the member. "1" means that the button is pressed and "0" means that it is released. During initialization with the function `pdInitPeripheral()`, reverse logic can be selected.

Digital Button	Constants for Bit Position Specification
Cross-key A top	PDD_DGT_KU
Cross-key A bottom	PDD_DGT_KD
Cross-key A left	PDD_DGT_KL
Cross-key A right	PDD_DGT_KR
A button	PDD_DGT_TA
B button	PDD_DGT_TB
C button	PDD_DGT_TC
D button	PDD_DGT_TD
X button	PDD_DGT_TX
Y button	PDD_DGT_TY
Z button	PDD_DGT_TZ
L button	PDD_DGT_TL
R button	PDD_DGT_TR
Start button	PDD_DGT_ST
Cross-key B top	PDD_DGT_KUB
Cross-key B bottom	PDD_DGT_KDB
Cross-key B left	PDD_DGT_KLB
Cross-key B right	PDD_DGT_KRB

## 4. Shinobi Functions for Getting Peripheral Data

Member	Meaning
Uint32 on	When button is pressed (button down), the corresponding bit is "1". When button is not pressed, the bit is "0".
Uint32 off	Bits are reversed from "on" member. When button is not pressed (button up), the corresponding bit is "1".
Uint32 press	When button has changed from not pressed to pressed (button down edge), the corresponding bit becomes "1". Other bits are "0".
Uint32 release	When button has changed from pressed to not pressed (button up edge), the corresponding bit becomes "1". Other bits are "0".

**Note:** When negative logic was selected, the bit status for button, on, off, press, release will be reversed.

Digital LR button information is simulated by the software based on analog LR information. Note that if a device does not have an analog LR button, the bit status will not change. (There are no devices with digital LR buttons in physical form.)

### Analog Axis Data

Analog axis information comprises the six members "r", "l", "x1", "y1", "x2", and "y2". For non-connected devices or devices without the respective button or lever, the data are the same as for center position.

Member	Sign	Range	Center position	Lever/ button
Uint16 r	Unsigned	0 - 255	0	Analog R button
Uint16 l	Unsigned	0 - 255	0	Analog L button
Sint16 x1	Signed	-128 - 0 - 127	0	Analog lever X1
Sint16 y1	Signed	-128 - 0 - 127	0	Analog lever Y1
Sint16 x2	Signed	-128 - 0 - 127	0	Analog lever X2
Sint16 y2	Signed	-128 - 0 - 127	0	Analog lever Y2

### Data Acquiry Error

If an error occurs during communication with the device according to the Maple bus protocol, correct button data will not be acquired. In such a case, the structure is set to the values shown below.

Member	Set Value
id,name,extend	Previous value
Digital button	All buttons not pressed
Analog axis	Center position

The function `pdGetPeripheralError()` can be used to check whether an error has occurred.

Error code	Contents
PDD_ERR_OK	No error
PDD_ERR_RETRY	Button data could not be acquired normally
PDD_ERR_GENERIC	Undefine error

The library does not contain steps for recovery when an error has occurred, Suitable routines must therefore be written into the application.

## Function Specifications

### Function List

Function	Operation
<code>pdGetPeripheral</code>	Get controller button status
<code>pdGetPeripheralError</code>	Get controller error status
<code>pdExecPeripheralServer</code>	Get peripheral data from server

# pdGetPeripheral

Get controller button status

---

## Format

```
const PDS_PERIPHERAL* pdGetPeripheral(UINT32 port)
```

## Parameters

port            port number (PDD\_PORT\_A0/B0/C0/D0)

## Return Value

PDS\_PERIPHERAL structure address

## Function

Gets peripheral data.

## Related Topics

PDS\_PERIPHERAL structure

## Example

```
const PDS_PERIPHERAL* per;  
  
per = pdGetPeripheral(PDD_PORT_A0);  
  
if (per->press & PDD_DGT_ST) {  
    /* Start button was pressed */  
        :  
        :  
    }  
}
```

# **pdGetPeripheralError**

Get controller error status

---

## **Format**

```
Sint32 pdGetPeripheralError(Uint32 port)
```

## **Parameters**

port ... Port number (PDD\_PORT\_A0/B0/C0/D0)

## **Return Value**

0 ... No error

Negative ... Error value (PDD\_ERR\_XXXX)

## **Function**

Gets error code for controller data get process.

## **Example**

```
Sint32 err;  
err = pdGetPeripheralError(PDD_PORT_B0);
```

# pdExecPeripheralServer

Get peripheral data from server

---

## Format

```
(void) pdExecPeripheralServer(void)
```

## Input

None

## Return Value

None

## Function

- Creates peripheral data for that frame.
- Call this function once at the loop start of one frame.

## Remarks

Must be used together with `njWaitVSync`.

