Notes on This Document

Notes on Using This Document

- The contents provided by this document (PDF files) are intended only for supplying information.

- The contents provided by this document (PDF files) are subject to change without notice.

- We are not responsible for errors or omissions in technical or editorial aspects concerning the contents described in this document. We also are not responsible for technical measures, correspondence, execution according to this document, as well as for the results occurred by them such as inevitable, indirect or incidental damages.

Notes on Copyright

- Sony Corporation is the copyright holder of this document.

- No information in this document may be duplicated, reproduced or modified. It is also prohibited to publish the contents of this document on the Web or other public media without the express written permission of Sony Corporation.

About Registered Trademarks

- OPEN-R is a trademark or a registered trademark of Sony Corporation.

- "Memory Stick" is a trademark of Sony Corporation. "TM" is not described in this document.

- Adobe Acrobat and Adobe Reader are registered trademarks of Adobe Systems Incorporated.

- Other system names, product names, service names and firm names contained in this document are generally trademarks or registered trademarks of respective makers.
Index

Notes on This Document................................................................. 1
Notes on Using This Document...................................................... 1
Notes on Copyright........................................................................... 1
About Registered Trademarks.......................................................... 1

Chapter 1 Base Class........................................................................... 8

1.1 Class OObject ................................................................................ 8
Init()........................................................................................................ 9
Start()..................................................................................................... 9
Stop()..................................................................................................... 9
Destroy() .............................................................................................. 10
DoInit()............................................................................................... 10
DoStart()............................................................................................. 10
DoStop()............................................................................................... 11
DoDestroy()......................................................................................... 11
RegisterServiceEntry()...................................................................... 11

Chapter 2 Inter-object communication .......................................... 12

2.1 OSubject class ............................................................................... 12
OSubject() .......................................................................................... 12
~OSubject() ........................................................................................ 12
SetReadyEntry()................................................................................ 12
GetID() ............................................................................................... 12
SetBufferSize().................................................................................. 13
GetBufferSize().................................................................................. 13
SetNotifyUnitSize()............................................................................ 14
GetNotifyUnitSize()............................................................................ 14
SetData() ............................................................................................. 14
SetData() ............................................................................................. 14
SetData() ............................................................................................. 15
SetData() ............................................................................................. 15
SetData() ............................................................................................. 15
SetData() ............................................................................................. 16
SetData() ............................................................................................. 16
SetData() ............................................................................................. 17
SetData() ............................................................................................. 17
SetData() ............................................................................................. 17
NotifyObserver()................................................................................ 18
NotifyObserver()................................................................................ 18
NotifyObservers()............................................................................... 18
RemainBuffer()................................................................................... 19
RemainBuffer()................................................................................... 19
RemainBuffer() ....................................................................................19
ClearBuffer() ........................................................................................20
ClearBuffer() ........................................................................................20
ClearBuffer() ........................................................................................20
NumberOfObservers() ........................................................................20
begin() ..................................................................................................21
d() .....................................................................................................21
FindObserver() .....................................................................................21
IsAllReady() .........................................................................................22
IsAnyReady() .......................................................................................22
IsReady() ..............................................................................................22
IsReady() ..............................................................................................22
IsReady() ..............................................................................................23
ReadyStatus() .......................................................................................23
ReadyStatus() .......................................................................................23
ControlHandler() ..................................................................................24
ReadyHandler() ....................................................................................24

2.2 OReadyEvent class ............................................................................... 25
SbjIndex() .............................................................................................25
SenderID() ............................................................................................25
IsAssert() ..............................................................................................25
IsDeassert() ..........................................................................................25

2.3 OObserver class ................................................................................... 26
OObserver() ..........................................................................................26
~OObserver() .......................................................................................26
SetNotifyEntry()...................................................................................26
GetID() .................................................................................................26
SetBufCtrlParam()................................................................................27
SetSkip()...............................................................................................27
AssertReady().......................................................................................27
AssertReady().......................................................................................27
AssertReady().......................................................................................28
DeassertReady() ...................................................................................28
DeassertReady() ...................................................................................28
DeassertReady() ...................................................................................28
NumberOfSubjects() ............................................................................29
begin() ..................................................................................................29
d() .....................................................................................................29
ConnectHandler() .................................................................................30
NotifyHandler()....................................................................................30

2.4 ONotifyEvent class .............................................................................. 31
ObsIndex() ............................................................................................ 31
SenderID() ............................................................................................ 31
NumOfData() .......................................................................................... 31
NumOfNotify() ..................................................................................... 31
Data() .................................................................................................... 32
RCData() .............................................................................................. 32

2.5 RCRRegion class .............................................................................. 33
RCRegion() .......................................................................................... 33
RCRegion() .......................................................................................... 33
RCRegion() .......................................................................................... 33
~RCRegion() ........................................................................................ 34
AddReference() .................................................................................... 34
RemoveReference() ............................................................................... 34
NumberOfReference() ........................................................................... 35
Base() ................................................................................................... 35
Size() .................................................................................................... 35
MemID() .............................................................................................. 35
Offset() ................................................................................................. 36
SetSize() ................................................................................................ 36
ReserveSharedMemory() ....................................................................... 36

2.6 OShmPtrBase class ........................................................................... 37
OShmPtrBase() ..................................................................................... 37
OShmPtrBase() ..................................................................................... 37
OShmPtrBase() ..................................................................................... 37
~OShmPtrBase() .................................................................................. 37
operator=() .......................................................................................... 38
Deallocate() .......................................................................................... 38
Base() ................................................................................................... 38
Size() .................................................................................................... 38
MemID() .............................................................................................. 39
Offset() ................................................................................................. 39
RCRPtr() .............................................................................................. 39

2.7 OShmPtr class .................................................................................. 40
OShmPtr() ............................................................................................ 40
OShmPtr() ............................................................................................ 40
OShmPtr() ............................................................................................ 40
OShmPtr() ............................................................................................ 41
~OShmPtr() .......................................................................................... 41
operator=() .......................................................................................... 41
Chapter 3 Service ................................................................. 44

3.1 OVirtualRobotComm ......................................................... 44

3.2 OVirtualRobotAudioComm ............................................. 44

Chapter 4 Data Format ....................................................... 45

4.1 Common header ............................................................ 45
   ODataVectorInfo .............................................................. 45

4.2 Communication with OVirtualRobotComm ...................... 47
   4.2.1 OCommandVectorData ................................................. 47
   OCommandInfo .............................................................. 48
   OCommandData ............................................................ 49
   OJointCommandValue2 .................................................... 49
   OJointCommandValue3 .................................................... 50
   OLEDCommandValue2 ................................................... 50

4.2.2 OSensorFrameVectorData ............................................ 51
   OSensorFrameInfo ........................................................ 52
   OSensorFrameData ....................................................... 53
   OAcceleration ............................................................ 53
   OAngularVelocity ....................................................... 54
   OTemperature ............................................................ 54
   OForce ................................................................. 55
   OPressure .............................................................. 55
   OLength ................................................................. 56
   OSwitchStatus .......................................................... 56
   OJointValue ............................................................ 57

4.2.3 OFbkImageVectorData ............................................... 58
   OFbkImageInfo .......................................................... 58
   OFbkImage ............................................................. 59

4.3 Communication with OVirtualRobotAudioComm ............. 62
   4.3.1 OSoundVectorData .................................................. 62
   OSoundInfo ............................................................. 62
Chapter 5 OPEN-R API ................................................................. 66

OPENR::OpenPrimitive() ......................................................... 66
OPENR::ClosePrimitive() ......................................................... 66
OPENR::ControlPrimitive() ...................................................... 67
OPENR::NewCommandVectorData() .......................................... 68
OPENR::DeleteCommandVectorData() .................................... 68
OPENR::NewSoundVectorData() ............................................. 68
OPENR::DeleteSoundVectorData() ....................................... 69
OPENR::NewCdtVectorData() ................................................. 69
OPENR::DeleteCdtVectorData() ........................................... 69
OPENR::SetCdtVectorData() .................................................. 70
OPENR::EnableJointGain() ..................................................... 70
OPENR::DisableJointGain() ................................................... 70
OPENR::SetJointGain() ......................................................... 71
OPENR::RegisterDefaultJointGain() ....................................... 71
OPENR::SetDefaultJointGain() ............................................. 72
OPENR::GetJointValue() ....................................................... 72
OPENR::GetSensorValue() ...................................................... 72
OPENR::NewSyncKey() .......................................................... 73
OPENR::CancelSyncKey() ....................................................... 73
OPENR::DivideSyncKey() ....................................................... 73
OPENR::SetMotorPower() ..................................................... 74
OPENR::Shutdown() ............................................................. 74
OPENR::GetBootCondition() .................................................. 75
OPENR::GetPowerStatus() .................................................... 76
OPENR::ObservePowerStatus() ............................................. 79
OPENR::UnobservePowerStatus() ......................................... 80
OPENR::FindDesignData() ..................................................... 80
OPENR::DeleteDesignData() .................................................. 81
OPENR::GetRobotDesign() .................................................... 81
OPENR::GetMemoryStickStatus() .......................................... 81
OPENR::Fatal() ................................................................. 82
OPENR::SetTime() ............................................................. 82
OPENR::GetTime() ............................................................. 82
OPENR::SetTimeDifference() .............................................. 83
OPENR::GetTimeDifference() .............................................. 83
OPENR::SetVolumeSwitch() .................................................. 83
OPENR::GetVolumeSwitch() ..............................................................84
OPENR::GetJointGain().................................................................84
OPENR::GetDefaultJointGain().......................................................85

Chapter6 wireless LAN API ..........................................................86

ERA201D1_GetMACAddress()..........................................................86
ERA201D1_GetEtherStatistics()......................................................86
ERA201D1_GetWLANSettings().......................................................86
ERA201D1_GetWLANStatistics() ....................................................87
Chapter 1 Base Class
1.1 Class OObject

Descriptions
OObject is the base class of an object. oentryINIT, oentrySTART, oentrySTOP, and oentryDESTROY (these are entries) of the object respectively correspond to Init(), Start(), Stop() and Destroy().

When a message is notified to oentryINIT, oentrySTART, oentrySTOP, and oentryDESTROY, Init(), Start(), Stop() and Destroy() are called. Init(), Start(), Stop() and Destroy() call DoInit(), DoStart(), DoStop(), and DoDestroy() respectively.

In the derived class of OObject, you write the procedures unique to each object in DoInit(), DoStart(), DoStop(), and DoDestroy(). OObject has myOID_ as a protected member, and can be used in the derived class. myOID_ is initialized by OObject::OObject().

Header file
#include <OPENR/OObject.h>

Library
LD_LIBRARIES = ${DIR_LIB}/libOPENR.a

Class

class OObject {

public:
  OObject();
  virtual ~OObject();

  void Init (const OSystemEvent& event);
  void Start (const OSystemEvent& event);
  void Stop (const OSystemEvent& event);
  void Destroy (const OSystemEvent& event);

  virtual OStatus DoInit (const OSystemEvent& event);
  virtual OStatus DoStart (const OSystemEvent& event);
  virtual OStatus DoStop (const OSystemEvent& event);
  virtual OStatus DoDestroy (const OSystemEvent& event);

protected:
  OID myOID_;
  OStatus RegisterServiceEntry(const OServiceEntry& entry, const char* name);

};
The following are member functions.

**Init()**

**Syntax**

```cpp
void Init(const OSystemEvent& event)
```

**Description**

This is called from OObjectManager when an object is initialized. OObjectManager passes event to an object during the initialization. Init() calls DoInit() and notifies the returned value of DoInit() to OObjectManager.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Event information of Init</td>
</tr>
</tbody>
</table>

**Returned value**

None

**Start()**

**Syntax**

```cpp
void Start(const OSystemEvent& event)
```

**Description**

This is called from OObjectManager when an object starts. The OObjectManager passes event to an object during the start. Start() calls DoStart() and notifies the returned value of DoStart() to OObjectManager.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Event information of Start</td>
</tr>
</tbody>
</table>

**Returned value**

None

**Stop()**

**Syntax**

```cpp
void Stop(const OSystemEvent& event)
```

**Description**

This is called from OObjectManager when an object is stopped. The OObjectManager passes event to an object during the stop. Stop() calls DoStop() and notifies the returned value of DoStop() to OObjectManager.

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Event information of Stop</td>
</tr>
</tbody>
</table>

**Returned value**

None
**Destroy()**

**Syntax**

```cpp
void Destroy(const OSystemEvent& event)
```

**Description**

This is called from OObjectManager when an object is destroyed. OObjectManager passes event to an object during the destroy. Destroy() calls DoDestroy() and notifies the returned value of DoDestroy() to OObject Manager.

**Parameters**

- `event`: Event information of Destroy

**Returned value**

None

**DoInit()**

**Syntax**

```cpp
OStatus DoInit(const OSystemEvent& event)
```

**Description**

This is called from Init(). You write your method by overriding it in a derived class. Event is the same as the one passed in Init(). A return value of DoInit() is notified to OObjectManager in Init().

**Parameters**

- `event`: Event information of Init

**Returned value**

<table>
<thead>
<tr>
<th>Success</th>
<th>oSUCCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>other</td>
<td>In the case of a failure, a parameter other than oSUCCESS is returned. A return value can be set freely with DoInit(), which you override.</td>
</tr>
</tbody>
</table>

**DoStart()**

**Syntax**

```cpp
OStatus DoStart(const OSystemEvent& event)
```

**Description**

This is called from Start(). You write your method by overriding it in a derived class. Event is the same as the one passed in Start(). A return value of DoStart() is notified to OObjectManager in Start().

**Parameters**

- `event`: Event information of Start

**Returned value**

<table>
<thead>
<tr>
<th>Success</th>
<th>oSUCCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>other</td>
<td>In the case of a failure, a parameter other than oSUCCESS is returned. A return value can be set freely with DoStart(), which you override.</td>
</tr>
</tbody>
</table>
**DoStop()**

**Syntax**

```c
OStatus DoStop(const OSystemEvent& event)
```

**Description**

This is called from Stop(). You write your method by overriding it in a derived class. Event is the same as the one passed in Stop(). A return value of DoStop() is notified to OObjectManager in Stop().

**Parameters**

- `event`  
  Event information of Stop

**Returned value**

- `oSUCCESS`  
  Success
- `other`  
  In the case of a failure, a parameter other than `oSUCCESS` is returned. A return value can be set freely with DoStop(), which you override.

**DoDestroy()**

**Syntax**

```c
OStatus DoDestroy(const OSystemEvent& event)
```

**Description**

This is called from Destroy(). You write your method by overriding it in a derived class. Event is the same as the one passed in Destroy(). A return value of DoDestroy() is notified to OObjectManager in Destroy().

**Parameters**

- `event`  
  Event information of Destroy

**Returned value**

- `oSUCCESS`  
  Success
- `other`  
  In the case of a failure, a parameter other than `oSUCCESS` is returned. A return value can be set freely with DoDestroy(), which you override.

**RegisterServiceEntry()**

**Syntax**

```c
OStatus RegisterServiceEntry(out const OServiceEntry& entry, const char* name)
```

**Description**

This registers a service entry.

**Parameters**

- `entry`  
  Service entry
- `name`  
  Service name

**Returned value**

- `oSUCCESS`  
  Success
- `oALREADY_EXIST`  
  A service entry of the same name is already registered.
- `oFAIL`  
  Failure
Chapter 2 Inter-object communication
2.1 OSubject class

The following are member functions.

- **OSubject()**
  - **Syntax**
    
    OSubject(void)

  - **Description**
    
    Constructor

  - **Parameters**
    
    None

  - **Returned value**
    
    None

- **~OSubject()**
  - **Syntax**
    
    ~OSubject()

  - **Description**
    
    Destructor

  - **Parameters**
    
    None

  - **Returned value**
    
    None

- **SetReadyEntry()**
  - **Syntax**
    
    OStatus SetReadyEntry(const OServiceEntry& entry)

  - **Description**
    
    This sets entry for a subject to receive ASSERT-READY or DEASSERT-READY messages. This setting should be done in DoInit().

  - **Parameters**
    
    entry  Entry for receiving ASSERT-READY or DEASSERT-READY messages

  - **Returned value**
    
    oSUCCESS  success

- **GetID()**
  - **Syntax**
    
    const SubjectID& GetID(void) const

  - **Description**
    
    This gets the SubjectID of a subject. The SubjectID is a unique value among subjects.

  - **Parameters**
    
    None

  - **Returned value**
    
    subject ID
SetBufferSize()

**Syntax**

OStatus SetBufferSize(size_t size)

**Description**

This sets the maximum buffer size (number of entries) prepared in the subject for each observer. This setting should be done in DoInit().

**Parameters**

size   The maximum buffer size (number of entries) for each observer

**Returned value**

oSUCCESS  success
others  failure

GetBufferSize()

**Syntax**

size_t GetBufferSize(void) const

**Description**

This returns the buffer size (number of entries) that was set in DoInit().

**Parameters**

None

**Returned value**

Current buffer size (number of entries)

SetNotifyUnitSize()

**Syntax**

OStatus SetNotifyUnitSize(size_t size)

**Description**

This sets the number of SetData() calls to make the minimum unit of transmission data. For example, some data may be composed of a header part and a body part, with each part requiring SetData(), followed by the execution of NotifyObservers(). In this case, the setting value (size) is 2. The call of this function is used when the buffer size prepared by subject is calculated. Setting this value, if any, should be done in DoInit(). When no setting is done, the default value is 1. In this case, SetData() and NotifyObserver() are called once respectively for each transmission.

**Parameters**

size   The number of SetData() calls to makes the minimum unit of transmission data.

**Returned value**

oSUCCESS  success
others  failure
GetNotifyUnitSize()

Syntax
size_t  GetNotifyUnitSize(void) const

Description
This returns the number of SetData() calls to make the minimum unit of transmission data.

Parameters
None

Returned value
The number of SetData() calls necessary for one transmission.

SetData()

Syntax
OStatus  SetData(const void* buf, size_t size)

Description
In this function, the data region specified by ‘buf’ and ‘size’ are copied to a shared memory segment. Then, the information of the shared memory segment is set to the transmission buffers for all the observers. Because the specified region is copied to a shared memory segment, you can overwrite the source region after calling this function. If a buffer overflow occurs, the oldest entry waiting for transmission is overwritten by the current information. Use RemainBuffer() to check for buffer overflow beforehand.

Parameters
buf   The pointer to the region where the data is located.
size  The size of data in bytes.

Returned value
oSUCCESS  success
others  failure

SetData()

Syntax
OStatus  SetData(const ObserverInfo& info, const void* buf, size_t size)

Description
In this function, the data region specified by ‘buf’ and ‘size’ are copied to a shared memory segment. Then, the information of the shared memory segment is set to the transmission buffer for the observer specified by ‘info’. Because this function can omit the call to FindObserver(), this function is more efficient than SetData(const ObserverID&, const void*, size_t). Because the specified region is copied to a shared memory segment, you can overwrite the source region after calling this function. If a buffer overflow occurs, the oldest entry waiting for transmission is overwritten by the current information. Use RemainBuffer() to check for the buffer overflow beforehand.

Parameters
info  The observer information. For example, the ObserverInfo type can be obtained by accessing the data that ObserverConstIterator points to, which is obtained by calling OSubject::begin().
buf   The pointer to the region where the data is located.
size  The size of data in bytes.

Returned value
oSUCCESS  success
others  failure
SetData()

Syntax
OStatus  SetData(const ObserverID& id, const void* buf, size_t size)

Description
This function is the same as SetData(*FindObserver(id), buf, size). That is, the data region specified by ‘buf’ and ‘size’ are copied to a shared memory segment. Then, the information of the shared memory segment is set to the transmission buffer for the observer specified by ‘id’. Because the specified region is copied to a shared memory segment, you can overwrite the source region after calling this function. If a buffer overflow occurs, the oldest entry waiting for transmission is overwritten by the current information. Use RemainBuffer() to check for the buffer overflow beforehand.

Parameters
id    The observer ID. In case the ‘id’ is invalid for the present subject, the result or effect of this function is undefined.
buf    The pointer to the region where the data is located.
size  The size of data in bytes.

Returned value
oSUCCESS       success
others  failure

SetData()

Syntax
OStatus  SetData(RCRegion* region)

Description
This sets the information of the shared memory segment specified by ‘region’, to the transmission buffers for all observers. If a buffer overflow occurs, the oldest entry waiting for transmission is overwritten. To check for the buffer overflow beforehand, use RemainBuffer(). RCRegion::AddReference() is called in this function to increment the reference counter for the specified region. So, the region must not be overwritten until it becomes available again. Use RCRegion::NumberOfReference() to check if it is available or not.

Parameters
region  The pointer to the shared memory segment with a reference counter.

Returned value
oSUCCESS       success
others  failure
**SetData()**

**Syntax**

```c
OStatus SetData(const ObserverInfo& info, RCRegion* region)
```

**Description**

This is the same as `SetData(*FindObserver(id), region)`. That is, this function sets the information of the shared memory segment specified by `region`, to the transmission buffer for the observer specified by `info`. If a buffer overflow occurs, the oldest entry waiting for transmission is overwritten. To check for buffer overflow beforehand, use `RemainBuffer()`. In this function, `RCRegion::AddReference()` is called to increment the reference counter for the specified region. So, the region must not be overwritten until it becomes available again. Use `RCRegion::NumberOfReference()` to check if it is available or not.

**Parameters**

- `info`: The observer information. For example, the `ObserverInfo` type can be obtained by accessing the data that `ObserverConstIterator` points to, which is obtained by calling `OSubject::begin()`.
- `region`: The pointer to the shared memory segment with a reference counter.

**Returned value**

- `oSUCCESS`: success
- `others`: failure

---

**SetData()**

**Syntax**

```c
OStatus SetData(const ObserverID& id, RCRegion* region)
```

**Description**

This works the same as `SetData(*FindObserver(id), region)`. That is, this sets the information of the shared memory segment specified by argument `region`, to the transmission buffer for the observer specified by `id`. In case of a buffer overflow, the oldest entry for transmission is overwritten. In order to know the buffer overflow beforehand, use `RemainBuffer()`. In this function, `RCRegion::AddReference()` is called to increment the reference counter for the specified region. So, the region must not be overwritten until it becomes available again. Use `RCRegion::NumberOfReference()` to see if it is available or not.

**Parameters**

- `id`: The observer ID. In case the `id` is invalid for the present subject, the result or effect of this function is undefined.
- `region`: The pointer to the shared memory segment with reference counter.

**Returned value**

- `oSUCCESS`: success
- `others`: failure
SetData()

Syntax
OStatus SetData(OShmPtrBase& p)

Description
This sets the information of the shared memory segment specified by ‘p’ to the
transmission buffers for all observers. If a buffer overflow occurs, the oldest entry
waiting for transmission is overwritten. To check for buffer overflow beforehand,
use RemainBuffer().

Parameters
p   The pointer to the shared memory segment with a reference counter

Returned value
oSUCCESS  success
others     failure

SetData()

Syntax
OStatus SetData(const ObserverInfo& info, const OShmPtrBase& p)

Description
This sets the information of the shared memory segment specified by ‘p’ to the
transmission buffer for the observer specified by ‘info’. Because this function omits
the call to FindObserver(), this function is more efficient than SetData(const
ObserverID&, RCRegion* region). If a buffer overflow occurs, the oldest entry
waiting for transmission is overwritten To check for overflow beforehand, use
RemainBuffer().

Parameters
info  The observer information. For example, the ObserverInfo type can
      be obtained by accessing the data that ObserverConstIterator
      points to, which is obtained by calling OSubject::begin().
p   The pointer to the shared memory segment with a reference counter.

Returned value
oSUCCESS  success
others     failure

SetData()

Syntax
OStatus SetData(const ObserverID& id, const OShmPtrBase& p)

Description
This sets the information of the shared memory segment specified by ‘p’ to the
transmission buffer for the observer specified by ‘id’. If a buffer overflow occurs,
the oldest entry waiting for transmission is overwritten. To check for buffer
overflow beforehand, use RemainBuffer().
This function is the same as SetData(*FindObserver(id), p).

Parameters
id   The observer ID. In case the 'id' is invalid for the present subject, the result
      or effect of the function is undefined.
p   The pointer to the shared memory segment with a reference counter.

Returned value
oSUCCESS  success
others     failure
NotifyObserver()

Syntax
OStatus NotifyObserver(const ObserverInfo& observer)

Description
This sends the data in the transmission buffer to the specified observer. If the observer is in the ASSERT-READY state, the data is immediately sent. If the observer is in the DEASSERT-READY state, the data is deleted. If the observer is not in the ASSERT-READY or DEASSERT-READY state, the data is kept in the buffer and is sent soon after the observer’s state becomes ASSERT-READY.

Parameters
observer The observer information. For example, the ObserverInfo type can be obtained by accessing the data that ObserverConstIterator points to, which is obtained by calling OSubject::begin().

Returned value
oSUCCESS success
others failure

NotifyObserver()

Syntax
OStatus NotifyObserver(const ObserverID& id)

Description
This sends the data in the transmission buffer to the specified observer. If the observer is in the ASSERT-READY state, the data is immediately sent. If the observer is in the DEASSERT-READY state, the data is deleted. If the observer is not in the ASSERT-READY or DEASSERT-READY state, the data is kept in the buffer and is sent soon after the observer’s state becomes ASSERT-READY. Because this function is the same as NotifyObserver(*FindObserver(id)), the function has the overhead of FindObserver().

Parameters
id observer ID

Returned value
oSUCCESS success
others failure

NotifyObservers()

Syntax
OStatus NotifyObservers(void)

Description
This sends the data in the transmission buffers to all of the observers. This performs the followings for each observer. If an observer is in the ASSERT-READY state, the data is immediately sent. If an observer is in the DEASSERT-READY state, the data is deleted. If an observer is not in the ASSERT-READY or DEASSERT-READY state, the data is kept in the buffer and is sent soon after the observer’s state becomes ASSERT-READY.

Parameters
None

Returned value
oSUCCESS success
others failure
**RemainBuffer()**

**Syntax**

```cpp
size_t RemainBuffer(const ObserverInfo& observer) const
```

**Description**

This returns the remaining number of transmission buffer entries for the specified observer. If `setData()` is called more than the number of times obtained by the returned value, the data in the buffer is deleted in oldest-first manner.

**Parameters**

- `observer` The observer information. For example, the `ObserverInfo` type can be obtained by accessing the data that `ObserverConstIterator` points to, which is obtained by calling `OSubject::begin()`.

**Returned value**

Remaining number of buffer elements

---

**RemainBuffer()**

**Syntax**

```cpp
size_t RemainBuffer(const ObserverID& id) const
```

**Description**

This returns the remaining number of transmission buffer elements for the specified observer. If `setData()` is called more than the number of times obtained by the returned value, the data in the buffer is deleted in oldest-first manner. This function is the same as `RemainBuffer(*FindObserver(id))`.

**Parameters**

- `id` observer ID

**Returned value**

Remaining number of buffer elements. 0 if observer ID is invalid.

---

**RemainBuffer()**

**Syntax**

```cpp
size_t RemainBuffer(void) const
```

**Description**

This returns the remaining number of transmission buffer elements for observers. The number is the minimum value among the observers. If `setData()` is called more than the number of times obtained by the returned value, the data in the buffer is deleted in oldest-first manner.

**Parameters**

None

**Returned value**

Remaining number of buffer elements
ClearBuffer()

Syntax
OStatus ClearBuffer(void)

Description
This clears the transmission buffers for all observers.

Parameters
None

Returned value
oSUCCESS success
others failure

ClearBuffer()

Syntax
OStatus ClearBuffer(ObserverInfo& info)

Description
This clears the transmission buffer for the specified observer.

Parameters
info Observer information

Returned value
oSUCCESS success
others failure

ClearBuffer()

Syntax
OStatus ClearBuffer(ObserverID& id)

Description
This clears the transmission buffer for the specified observer. This function is the same as ClearBuffer(*FindObserver(id)).

Parameters
id ObserverID

Returned value
oSUCCESS success
others failure

NumberOfObservers()

Syntax
int NumberOfObservers(void) const

Description
This returns the number of observers connecting to the present subject.

Parameters
None

Returned value
The number of observers connecting to the present subject
begin()

Syntax
ObserverConstIterator begin(void) const

Description
This returns the iterator that points to the first observer in the list of observers that connect to the present subject.

Parameters
None

Returned value
The iterator that points to the first observer

end()

Syntax
ObserverConstIterator end(void) const

Description
This returns the invalid iterator that points to the location after the last observer in the list of observers that connect to the present subject.

Parameters
None

Returned value
The invalid iterator that points to the location after the last observer

FindObserver()

Syntax
ObserverConstIterator FindObserver(const ObserverID& id) const

Description
This returns the iterator that points to the observer specified by id. If the observer with id is not found, an invalid iterator is returned.

Parameters
None

Returned value
The iterator that points to the specified observer
IsAllReady()

Syntax
int IsAllReady(void) const

Description
This checks if all the observers are in the ASSERT-READY or DEASSERT-READY state.

Parameters
None

Returned value
Non-zero All the observers are in either the ASSERT-READY or DEASSERT-READY state, and at least one of observers is in the ASSERT-READY state. If NotifyObservers() is executed under this state, a message is immediately sent to the observers that require the message.
Zero At least one observer is in neither the ASSERT-READY nor DEASSERT-READY state, or all observers are in the DEASSERT-READY state.

IsAnyReady()

Syntax
int IsAnyReady(void) const

Description
This checks if any observers are in the ASSERT-READY state.

Parameters
None

Returned value
Non-zero At least one observer is in the ASSERT-READY state.
Zero No observers are in the ASSERT-READY state.

IsReady()

Syntax
int IsReady(const ObserverInfo& info) const

Description
This sees if the specified observer is in an ASSERT-READY state.

Parameters
info The observer information. For example, type ObserverInfo can be obtained by accessing the data that type ObserverConstIterator points to, which is obtained by calling OSubject::begin().

Returned value
Non-zero The specified observer is in the ASSERT-READY state.
Zero The specified observer is not in the ASSERT-READY state.
IsReady()

Syntax
int IsReady(const ObserverID& id) const

Description
This checks if the specified observer is in the ASSERT-READY state. This function is the same as IsReady(*FindObserver(id)).

Parameters
id ObserverID

Returned value
Non-zero The specified observer is in the ASSERT-READY state.
Zero The specified observer is not in the ASSERT-READY state, or ObserverID is invalid.

ReadyStatus()

Syntax
int ReadyStatus(const ObserverInfo& info) const

Description
This returns the state of the specified observer.

Parameters
info The observer information. For example, the ObserverInfo type can be obtained by accessing the data that ObserverConstIterator points to, which is obtained by calling OSubject::begin().

Returned value
A positive value The subject received an ASSERT-READY message from the specified observer. (ASSERT-READY state)
Zero Because the specified observer has not sent a message yet, the state is unknown.
A negative value The subject received a DEASSERT-READY message from the specified observer. (DEASSERT-READY state)

ReadyStatus()

Syntax
int ReadyStatus(const ObserverID& id) const

Description
This returns the status of the specified observer. This function is the same as ReadyStatus(*FindObserver(id)).

Parameters
id observer ID

Returned value
A positive value The subject received an ASSERT-READY message from the specified observer. (ASSERT-READY state)
Zero Because the specified observer has not sent a message yet, the state is unknown. Or, observer ID is invalid.
A negative value The subject received a DEASSERT-READY message from the specified observer. (DEASSERT-READY state)
ControlHandler()

Syntax

void ControlHandler(const OControlMessage& msg, OStatus status=oSUCCESS)

Description

This sets up a subject in accordance with the received OControlMessage. This is called during the connection phase of objects.

Parameters

msg OControlMessage received from an observer.
status A user defined state. Specify oSUCCESS for a default value. In case it is not oSUCCESS, this connection will be refused. For example, in case the initialization and resource allocation in a user defined hook method has failed, specify oFAIL.

Returned value

None

ReadyHandler()

Syntax

void ReadyHandler(const OReadyMessage& msg)

Description

This receives the OReadyMessage and responds to it.

Parameters

msg OReadyMessage received from an observer.

Returned value

None
2.2 OReadyEvent class

The following are member functions.

SbjIndex()

Syntax
int SbjIndex(void) const

Description
This returns the index of the subject that receives OReadyEvent.

Parameters
None

Returned value
Index of a subject

SenderID()

Syntax
const ObserverID& SenderID(void) const

Description
This returns the observer ID of the observer that has sent OReadyEvent.

Parameters
None

Returned value
Observer ID

IsAssert()

Syntax
bool IsAssert(void) const

Description
This checks if OReadyMessage is an ASSERT-READY message.

Parameters
None

Returned value
true An ASSERT-READY message
false Other

IsDeassert()

Syntax
bool IsDeassert(void) const

Description
This checks if OReadyMessage is a DEASSERT-READY message.

Parameters
None

Returned value
true A DEASSERT-READY message
false Other
2.3 OObserver class

The following are member functions.

OObserver()
Syntax
OObserver(void)

Description
Constructor

Parameters
None

Returned value
None

~OObserver()
Syntax
~OObserver()

Description
Destructor

Parameters
None

Returned value
None

SetNotifyEntry()
Syntax
OStatus  SetNotifyEntry(const OServiceEntry& entry)

Description
This sets the entry for the observer to receive NOTIFY messages. This setting should be done in DoInit().

Parameters
entry  An entry for receiving NOTIFY

Returned value
oSUCCESS  success
others  failure

GetID()
Syntax
const ObserverID&  GetID(void) const

Description
This returns the ObserverID of an observer. Each observer has a unique ObserverID.

Parameters
None

Returned value
A unique value for each observer
SetBufCtrlParam()

Syntax
void SetBufCtrlParam(size_t skip, size_t min, size_t max)

Description
This sets the necessary control parameters of the buffers that the subject holds for observers. This setting should be done in DoInit().

Parameters
skip This specifies the data-skip (a sampling interval) to reduce the amount of receiving data. The default value is zero, which means no sub-sampling.

min This specifies the minimum amount of data units when a subject sends the NOTIFY message to an observer. The default value is one. If you adequately set this parameter, you can reduce the frequency of data-receiving without data loss.

max This specifies the maximum transmission buffer size (units) that a subject should hold until an observer's state becomes ASSET-READY. This parameter must be greater than or equal to 'min'. The default value is one. Only the last transmission data unit is held in the buffer when the value is one.

Returned value
None

SetSkip()

Syntax
void SetSkip(size_t skip)

Description
This sets the necessary control parameter of the buffers that the subject holds for observers. This setting should be done in DoInit(). This function is available to keep compatibility with previous software. This function is the same as SetBufCtrlParam(skip, 1, 1).

Parameters
skip This specifies the data-skip (the sampling interval) to reduce the amount of receiving data. The default value is zero, which means no sub-sampling.

Returned value
None

AssertReady()

Syntax
OSStatus AssertReady(void)

Description
This sends an ASSERT-READY message to all connecting subjects.

Parameters
None

Returned value
oSUCCESS success
others failure
**AssertReady()**

**Syntax**

OStatus  AssertReady(const SubjectID& id)

**Description**

This sends an ASSERT-READY message to only the specified subject.

**Parameters**

id  The ID of a subject that receives messages.

**Returned value**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>success</td>
</tr>
<tr>
<td>others</td>
<td>failure</td>
</tr>
</tbody>
</table>

**AssertReady()**

**Syntax**

OStatus  AssertReady(const SubjectInfo& info)

**Description**

This sends an ASSERT-READY message to only the specified subject.

**Parameters**

info  The ID information of a subject that receives messages.

**Returned value**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>success</td>
</tr>
<tr>
<td>others</td>
<td>failure</td>
</tr>
</tbody>
</table>

**DeassertReady()**

**Syntax**

OStatus  DeassertReady(void)

**Description**

This sends a DEASSERT-READY message to all connecting subjects.

**Parameters**

None

**Returned value**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>success</td>
</tr>
<tr>
<td>others</td>
<td>failure</td>
</tr>
</tbody>
</table>

**DeassertReady()**

**Syntax**

OStatus  DeassertReady(const SubjectID& id)

**Description**

This sends a DEASSERT-READY message to only the specified subject.

**Parameters**

id  The ID of a subject that receives messages.

**Returned value**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>success</td>
</tr>
<tr>
<td>others</td>
<td>failure</td>
</tr>
</tbody>
</table>
DeassertReady()

Syntax
OStatus  DeassertReady(const SubjectInfo& info)

Description
This sends a DEASSERT-READY message to only the specified subject.

Parameters
info  The ID information of a subject that receives messages.

Returned value
oSUCCESS  success
others      failure

NumberOfSubjects()

Syntax
int  NumberOfSubjects(void) const

Description
This returns the number of subjects connecting to the present observer.

Parameters
None

Returned value
The number of subjects connecting to the present observer

begin()

Syntax
SubjectConstIterator  begin(void) const

Description
This returns the iterator that points to the first subject in the subject list that connects to the present observer.

Parameters
None

Returned value
The iterator that points to the first subject

end()

Syntax
SubjectConstIterator  end(void) const;

Description
This returns the invalid iterator that points to the location after the last subject in the subject list that connects to the present observer.

Parameters
None

Returned value
The invalid iterator that points to the location after the last subject
ConnectHandler()

Syntax
void ConnectHandler(const OConnectMessage& msg, OStatus status=oSUCCESS)

Description
This sets an observer in accordance with the received OConnectMessage. This is called during the connection phase of an object.

Parameters
msg An OConnectMessage that was notified by OServiceManager.
status This indicates the status of the function for any user-defined initialization/resource allocation. The default value is oSUCCESS, and in case it is not oSUCCESS, connection will be refused.

Returned value
None

NotifyHandler()

Syntax
void NotifyHandler(const ONotifyMessage& msg, ONotifyEvent* pEvent)

Description
This sets and initializes ONotifyEvent in accordance with the received ONotifyMessage. This function is automatically called in stub.cc.

Parameters
msg ONotifyMessage received from a subject.
pEvent The pointer to an ONotifyEvent data corresponding to the received ONotifyMessage.

Returned value
None
2.4 ONotifyEvent class

The following are member functions.

**ObsIndex()**

**Syntax**

```cpp
int ObsIndex(void) const
```

**Description**

This returns the index of the observer that receives ONotifyEvent.

**Parameters**

None

**Returned value**

The index of the observer that receives ONotifyEvent

**SenderID()**

**Syntax**

```cpp
const SubjectInfo& SenderID(void) const
```

**Description**

This returns the ID information of the subject that sent ONotifyEvent.

**Parameters**

None

**Returned value**

The ID information of the subject that sent ONotifyEvent

**NumOfData()**

**Syntax**

```cpp
int NumOfData(void) const
```

**Description**

This returns the number of the received data elements.

**Parameters**

None

**Returned value**

Number of the received data elements

**NumOfNotify()**

**Syntax**

```cpp
int NumOfNotify(void) const
```

**Description**

This returns the number of times that ONotifyEvent() was executed for the data that has been sent.

**Parameters**

None

**Returned value**

The number of times that a subject executed ONotifyEvent().
**Data()**

**Syntax**

```cpp
const void* Data(int i) const
```

**Description**

This returns the i-th data element address of the received data. This pointer becomes invalid soon after sending an ASSERT-READY or DEASSERT-READY message to a subject.

**Parameters**

- `i`: The index of the data element you want to process.

**Returned value**

The i-th data element address

---

**Data()**

**Syntax**

```cpp
const void** Data(void) const
```

**Description**

This returns a pointer to an array of the pointers to the received data.

**Parameters**

None

**Returned value**

A pointer to an array of pointers

---

**RCData()**

**Syntax**

```cpp
RCRegion* RCData(int i) const
```

**Description**

This returns the pointer to the shared memory segment, with reference counter, which corresponds to the i-th data element of the received data.

**Parameters**

- `i`: The index of the data you want to process.

**Returned value**

The pointer to the shared memory segment, with reference counter, which corresponds to the i-th data element
2.5 RCRegion class

This class has a pointer to the shared memory segment and controls the reference counter for the memory segment. The following are member functions. You cannot instantiate this class on the local stack.

**RCRegion()**

**Syntax**

RCRegion(void)

**Description**

This is constructor. It constructs the instance pointing to NULL.

**Parameters**

None

**Returned value**

None

**RCRegion()**

**Syntax**

RCRegion(size_t size)

**Description**

This reserves a shared memory segment with the specified size, and constructs an instance pointing to this memory segment.

**Parameters**

size The size of the allocating shared memory (units are in bytes)

**Returned value**

None

**RCRegion()**

**Syntax**

RCRegion(MemoryRegionID memID, size_t offset, void* baseAddr=NULL, size_t size=0)

**Description**

This constructs an instance pointing to the specified memory segment. Because no memory allocation is executed here, reserve the corresponding memory segment beforehand with the other means.

**Parameters**

memID The shared memory ID where the data is located.
offset The offset of baseAddr from the base address of the shared memory segment specified by memID.
baseAddr The base address of data (a starting address)
size Data size in bytes

**Returned value**

None
~RCRegion()

Syntax
~RCRegion()

Description
It is not allowable to call this function directly. RCRegion() should be placed on the heap, not on the local stack. 'Delete region' is also prohibited, because it is possible that this segment is being referred to by others. Instead of calling the destructor, you must call RCRegion::RemoveReference().

Parameters
None

Returned value
None

AddReference()

Syntax
void AddReference(void)

Description
This increments the reference counter of the shared memory segment.

Parameters
None

Returned value
None

RemoveReference()

Syntax
void RemoveReference(void)

Description
This decrements the reference counter of the shared memory segment. If all references to this region are removed, it automatically destructs itself. If it is the owner of that segment, the shared memory segment is deleted.

Parameters
None

Returned value
None
**NumberOfReference()**

**Syntax**
```
int NumberOfReference(void) const
```

**Description**
This returns the number of the reference counter.
If the returned value is 1, the segment is referred to by itself, and the owner of the segment can overwrite the segment.
If the returned value is more than 1, use the segment only for reading.
If the returned value is 0, do not access the segment since it is broken.

**Parameters**
None

**Returned value**
Number of reference counter

**Base()**

**Syntax**
```
char* Base(void) const
```

**Description**
This returns the base address of data in the shared memory segment.

**Parameters**
None

**Returned value**
The base address of data in the shared memory segment

**Size()**

**Syntax**
```
size_t Size(void) const
```

**Description**
This returns the size of data in the shared memory segment.

**Parameters**
None

**Returned value**
The size (in bytes) of data on the shared memory segment.

**MemID()**

**Syntax**
```
MemoryRegionID MemID(void) const
```

**Description**
This returns the ID of the shared memory segment.

**Parameters**
None

**Returned value**
The ID of the shared memory segment
Offset()

Syntax
size_t  Offset(void) const

Description
This returns the offset of the data segment. The offset is the number of bytes from
the base address obtained by the shared memory ID to the starting address of data.

Parameters
None

Returned value
The offset of the data segment

SetSize()

Syntax
void  SetSize(size_t size)

Description
This sets the value returned by RCRegion::Size() to ‘size’. This function is used so
the user can apply optimization in original memory allocation routines.

Parameters
size  The same value as the one returned by RCRegion::Size().

Returned value
None

ReserveSharedMemory()

Syntax
OStatus  ReserveSharedMemory(size_t size)

Description
This function is a static member function of class RCRegion. This function is used
to avoid a memory allocation at an unexpected time during a runtime. This function
guarantees that at least 'size' bytes of shared memory can be used for
libObjectComm library. In case enough shared memory segments do not exist when
this function is called, the necessary memory segment will be allocated. The
allocated memory segment is used when SetData(ptr, size) is executed. When
SetData(region) is used, it is not necessary to call this function. The reason is that
the SetData(region) function can freely control the generation time of class
RCRegion.

Parameters
size  The size of the memory segment to be reserved, for future
     SetData(ptr, size) calls.

Returned value
oSUCCESS  success
     others  failure
2.6 OShmPtrBase class

This is the base class that indicates the shared memory segment. This class is a capsule class of RRegion and does auto reference counting. The following are member functions.

OShmPtrBase()

Syntax
OShmPtrBase(void)

Description
This constructs an invalid OShmPtrBase.

Parameters
None

Returned value
None

OShmPtrBase()

Syntax
OShmPtrBase(const OShmPtrBase& p)

Description
This constructs OShmPtrBase that refers to the same region as the specified OShmPtrBase refers to.

Parameters
p  OShmPtrBase to be copied

Returned value
None

OShmPtrBase()

Syntax
OShmPtrBase(RCRegion* region)

Description
This constructs OShmPtrBase that refers to the specified region.

Parameters
region  The shared memory segment with a reference counter

Returned value
None

~OShmPtrBase()

Syntax
~OShmPtrBase()

Description
This destructs OShmPtrBase and decrements the reference counter.

Parameters
None

Returned value
None
operator=()

Syntax
OShmPtrBase&  operator=(const OShmPtrBase& p)

Description
This changes reference to the same segment as the specified OShmPtrBase refers to.

Parameters
p  OShmPtrBase to be copied

Returned value
*this

Deallocate()

Syntax
void  Deallocate(void)

Description
This decrements the reference counter and makes OShmPtrBase invalid.

Parameters
None

Returned value
None

Base()

Syntax
char*  Base(void) const

Description
This returns the base address of data in a shared memory segment.

Parameters
None

Returned value
The base address of data in a shared memory segment

Size()

Syntax
size_t  Size(void) const

Description
This returns the size of data in a shared memory segment.

Parameters
None

Returned value
The size of data in a shared memory segment
MemID()  

Syntax  
MemoryRegionID MemID(void) const

Description  
This returns the ID of a shared memory segment.

Parameters  
None

Returned value  
ID of a shared memory segment

Offset()  

Syntax  
size_t Offset(void) const

Description  
This returns the offset to the data segment. The offset is the number of bytes from the base address obtained by the corresponding shared memory ID to the starting address of data.

Parameters  
None

Returned value  
The offset to the data segment

RCRPtr()  

Syntax  
RCRegion* RCRPtr(void) const

Description  
This returns the pointer to a corresponding RCRegion.

Parameters  
None

Returned value  
The pointer to a corresponding RCRegion
2.7 OShmPtr class

This is a pointer to a shared memory segment. This is a template class that is different from the OShmPtrBase. The following are member functions.

**OShmPtr()**

**Syntax**

OShmPtr(void)

**Description**

This constructs an invalid instance of OShmPtr<T> type.

**Parameters**

None

**Returned value**

None

**OShmPtr()**

**Syntax**

OShmPtr(const OShmPtrBase& p)

**Description**

This constructs an instance of OShmPtr<T> type that refers to the region that the specified OShmPtrBase refers to.

**Parameters**

p OShmPtrBase to be copied

**Returned value**

None

**OShmPtr()**

**Syntax**

OShmPtr(RCRegion* region)

**Description**

This constructs an instance of OShmPtr<T> type that refers to the specified region.

**Parameters**

region The pointer to the shared memory segment with reference counter

**Returned value**

None
OShmPtr()

Syntax
OShmPtr(size_t n)

Description
This reserves a shared memory segment with the size of sizeof(T)*n, and constructs an array of OShmPtr<T> with n elements. This function internally calls Allocate(n). A constructor for type T is not called.

Parameters
n An array of OShmPtr<T> with n elements

Returned value
None

~OShmPtr()

Syntax
~OShmPtr()

Description
This destructs the OShmPtr<T> and decrements a reference counter.

Parameters
None

Returned value
None

operator=(

Syntax
OShmPtr<T>& operator=(const OShmPtrBase& p)

Description
This changes reference to the same region as the specified OShmPtrBase refers to.

Parameters
p OShmPtrBase to be copied

Returned value
*this

Allocate()

Syntax
void Allocate(int n)

Description
This reserves a shared memory segment with the size of sizeof(T)*n, and allocates an array of type T with n elements. The reference counter controls this newly constructed shared memory segment. A constructor for type T is not called.

Parameters
n The number of elements of an array of type T

Returned value
None
**NumOfElement()**

**Syntax**

```c
size_t  NumOfElement(void) const
```

**Description**

This returns the maximum number of elements in the array.

**Parameters**

None

**Returned value**

The number of elements in the array

**operator*()**

**Syntax**

```c
const T&  operator*(void) const
```

**Description**

This returns the reference to the first element in the array.

**Parameters**

None

**Returned value**

The reference to the first element in the array

**operator*()**

**Syntax**

```c
OShmPtr<T>::Proxy operator*(void)
```

**Description**

This returns the first element in the array. If someone tries to overwrite this element while someone else is still referring to it, the contents of the segment are copied to a newly reserved segment, and the newly reserved segment is overwritten.

**Parameters**

None

**Returned value**

The first element in the array

**operator[]()**

**Syntax**

```c
const T&  operator[](int i) const
```

**Description**

This returns the reference to the i-th element in the array.

**Parameters**

i  The index of the element in the array

**Returned value**

The reference to the i-th element in the array
**operator[]()**

**Syntax**
OShmPtr<T>::Proxy operator[](int index)

**Description**
This returns the i-th element in the array. If someone tries to overwrite this element while someone else is still referring to it, the contents of the segment are copied to a newly reserved segment, and the newly reserved segment is overwritten.

**Parameters**
i The index of the element in array

**Returned value**
The i-th element in the array

**operator->()**

**Syntax**
const T* operator->(void) const

**Description**
This returns the pointer to the first element in the array.

**Parameters**
None

**Returned value**
The pointer to the first element in the array
Chapter 3 Service
3.1 OVirtualRobotComm

Service
OVirtualRobotComm.Effector.OCommandVectorData.O
OVirtualRobotComm.Sensor.OSensorFrameVectorData.S

Description of Service
OVirtualRobotComm.Effector.OCommandVectorData.O
This is a service that receives joint and LED commands. The receiving data structure is OCommandVectorData. You can reserve a shared memory for OCommandVectorData with OPEN-R::NewCommandVectorData(). After the output of the received OCommandVectorData is completed, a READY EVENT is sent.

OVirtualRobotComm.Sensor.OSensorFrameVectorData.S
This is a service to send all of the sensor data available in a robot. The sending data structure is OSensorFrameVectorData. Four frames of data (32ms) is sent by one transmission.

This is a service to send the image data captured through the camera. The sending data structure is OFbkImageVectorData. Three sheets of YCrCb and a sheet of CDT are included in the image data.

3.2 OVirtualRobotAudioComm

Service
OVirtualRobotAudioComm.Speaker.OSoundVectorData.O
OVirtualRobotAudioComm.Mic.OSoundVectorData.S

Description of Service
OVirtualRobotAudioComm.Mic.OSoundVectorData.S
This is a service to send sound data from a microphone. Data is sent every 32ms. The sound data has the following format: PCM data, 16kHz and 16bit stereo.

OVirtualRobotAudioComm.Speaker.OSoundVectorData.O
This is a service to receive sound data. The receiving data structure is OSoundVectorData. You can reserve a shared memory for OSoundVectorData with OPENR::NewSoundVectorData(). After the output of the received data is finished, a READY EVENT is sent.
Chapter 4 Data Format

4.1 Common header

ODataVectorInfo

Description

ODataVectorInfo is a common header for OCommandVectorData, OSensorFrameVectorData, OFbkImageVectorData, OSoundVectorData, and OCdtVectorData. It contains the number of data elements, the size of the information block about elements and the information about a shared memory.

Structure

```c
struct ODataVectorInfo {
  MemoryRegionID memRegionID;
  void* physAddr;
  size_t offset;
  size_t totalSize;
  ODataType type;
  size_t infoOffset;
  size_t infoSize;
  size_t maxNumData;
  size_t numData;
  OVRSyncKey syncKey;
  longword wait;
  size_t optOffset;
  size_t optSize;
  longword padding[3];
  byte optional[odataOPTIONAL_MAX];
};
```

Header file

```c
#include <OPENR/ODataFormats.h>
```

Members

- **memRegionID**
  - This is the ID of a shared memory segment that holds data.

- **physAddr**
  - In OFbkImageVectorData and OSoundVectorData, this is set to the physical address of a shared memory. In other cases, this is set to 0.

- **offset**
  - This is the size of a shared memory that holds data

- **totalSize**
  - Data type and data structure corresponding to each type.

- **type**
  - A synchronous key

- **wait**
  - Delays commands and the output of sound, for the number of frames (in units of 8msec) specified by "wait".

- **optOffset**
  - The offset of the effective data in an optional area

- **optSize**
  - The size of the effective data in an optional area

- **padding[3]**
  - Padding to adjust the total number of bytes.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Data structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCommandVectorData</td>
<td>odataCOMMAND_VECTOR</td>
</tr>
<tr>
<td>OSensorFrameVectorData</td>
<td>odataSENSOR_FRAME_VECTOR</td>
</tr>
<tr>
<td>OFbkImageVectorData</td>
<td>odataFBKIMAGE_VECTOR</td>
</tr>
<tr>
<td>OSoundVectorData</td>
<td>odataSOUND_VECTOR</td>
</tr>
<tr>
<td>OCdtVectorData</td>
<td>odataCDT_VECTOR</td>
</tr>
</tbody>
</table>
```

- **infoOffset**
  - This is an offset (192 bytes) from the starting address of data to the array of the information block elements.

- **maxNumData**
  - The maximum number of elements that can be held in data

- **numData**
  - The number of elements in a valid data

- **syncKey**
  - A synchronous key

- **wait**
  - Delays commands and the output of sound, for the number of frames (in units of 8msec) specified by "wait".

- **optOffset**
  - The offset of the effective data in an optional area

- **optSize**
  - The size of the effective data in an optional area

- **padding[3]**
  - Padding to adjust the total number of bytes.
optional[odataOPTIONAL_MAX]

It is used for the delivery of the information between the object that receives OSensorFrameVectorData and the object that sends OCommandVectorData, OSoundVectorData. The data in optional[] (whose range is specified with optOffset and optSize) is updated, and the data is copied to optional[] of OSensorFrameVectorData.
4.2 Communication with OVirtualRobotComm

The following 3 types of data are used for communication with OVirtualRobotComm.

- OCommandVectorData  Command data
- OSensorFrameVectorData  Sensor data
- OFbkImageVectorData  Image data

The data is created in a shared memory. Each data has a common header (ODataVectorInfo), followed by an array containing an information block about each element, and an array of the main body of data.

4.2.1 OCommandVectorData

Description
This is a data structure that holds joint and LED commands. It consists of vectorInfo, followed by an array of OCommandInfo with a size of vector.Info.maxNumData, and an array of OCommandData. The type of each command is specified with the type of OCommandInfo. It is possible to keep different kinds of commands in one OCommandVectorData.

Structure

```c
struct OCommandVectorData {
    ODataVectorInfo   vectorInfo;
    OCommandInfo     info[1];

    void SetNumData(size_t ndata){vectorInfo.numData = ndata;}
    OCommandInfo* GetInfo(int index) {return &info[index];}
    OCommandData* GetData(int index) {
        return (OCommandData*)((byte*)&vectorInfo + info[index].dataOffset);
    }
};
```

Header file

```c
#include <OPENR/ODataFormats.h>
```
**OCommandInfo**

**Description**
This contains the type of element of OCommandVectorData, OPrimitiveID, the number of command frames, and an offset to commands.

**Structure**

```c
struct OCommandInfo {
    ODataType type;
    OPrimitiveID primitiveID;
    longword frameNumber;
    size_t numFrames;
    size_t frameSize;
    size_t dataOffset;
    size_t dataSize;
    longword padding[1];
}
```

**Header file**

```c
#include <OPENR/ODataFormats.h>
```

**Members**

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>This is the command type.</td>
</tr>
<tr>
<td></td>
<td>odataJOINT_COMMAND2</td>
</tr>
<tr>
<td></td>
<td>odataLED_COMMAND2</td>
</tr>
<tr>
<td>primitiveID</td>
<td>The ID of the CPC Primitive to be given a command.</td>
</tr>
<tr>
<td>frameNumber</td>
<td>The frame sequence number when the first frame is processed by the command will be stored here.</td>
</tr>
<tr>
<td>numFrames</td>
<td>This is the number of valid frames of command data that OCommandData keeps. Only numFrames frames out of ocommandMAX_FRAMES(=16) are processed.</td>
</tr>
<tr>
<td>frameSize</td>
<td>This is the size (8 bytes) of command data in one frame that OCommandData keeps.</td>
</tr>
<tr>
<td>dataOffset</td>
<td>This is an offset to OCommandData corresponding to OCommandInfo. This is an offset from the starting address of OCommandVectorData.</td>
</tr>
<tr>
<td>dataSize</td>
<td>This is the data size (128 bytes) of OCommandData corresponding to OCommandInfo.</td>
</tr>
<tr>
<td>padding[1]</td>
<td>Padding to adjust the total number of bytes.</td>
</tr>
</tbody>
</table>
OCommandData

**Description**
This is the main part of command data. OCommandValue is a generic data structure for one frame. In case of a joint command, OCommandData is cast to OJointCommandValue2. In case of an ear plunger, OCommandData is cast to OCameraCommandValue3. In case of an LED command, OCommandData is cast to OLEDCommandValue.

**Structure**
```
struct OCommandData {
    OCommandValue  value[ocommandMAX_FRAMES];
};
```

**Header file**
```
#include <OPENR/ODataFormats.h>
```

**Members**
- value[ocommandMAX_FRAMES]
  This is command data. OCommandData can hold data for a maximum of ocommandMAX_FRAMES (=16) frames. The number of valid frames is specified by numFrames of OCommandInfo.

OJointCommandValue2

**Description**
This is a joint command data for one frame.

**Structure**
```
struct OJointCommandValue2 {
    slongword  value;
    slongword  padding;
};
```

**Header file**
```
#include <OPENR/ODataFormats.h>
```

**Members**
- value
  This is a value to be set to a joint. The unit is micro radians (10⁻⁶ rad). In the case of 180 deg, the value would be 3141592.
- padding
  Padding to adjust the total number of bytes.
**OJointCommandValue3**

**Description**
The plunger movement in the ears.

**Structure**

```c
struct OJointCommandValue3 {
    OJointValue3   value;
    word           reserved;
    word           padding;
};
```

**Header file**

```
#include <OPENR/ODataFormats.h>
```

**Members**

- **value**
  It is a value to be set to a plunger. value can be ojoint3_STATE0 or ojoint3_STATE1.
- **reserved**
  This is reserved.
- **padding**
  Padding to adjust the total number of bytes.

**OLEDCommandValue2**

**Description**
This is a command data controlling an LED. The control of an LED is specified by ON/OFF and its duration. The minimum time to control the ON/OFF of an LED is 8 msec.

**Structure**

```c
struct OLEDCommandValue2 {
    OLEDValue  led;
    word       period;
    word       reserved;
};
```

**Header file**

```
#include <OPENR/ODataFormats.h>
```

**Members**

- **led**
  This specifies ON/OFF of an LED. led can be oledON or oledOFF.
- **period**
  This specifies how long an LED will remain in either state. The unit of time is 8ms..
- **reserved**
  This is reserved.
4.2.2 OSensorFrameVectorData

Description
This is a data structure in which data of each sensor, such as a joint sensor, an acceleration sensor, or a switch sensor, are kept. It consists of vectorInfo, followed by an array of OSensorFrameInfo with the number of vectorInfo.maxNumData elements and an array of OSensorFrameData. The type of each sensor data is specified by type in OSensorFrameInfo. One OSensorFrameVectorData can contain different kinds of sensor data.

Structure

```c
struct OSensorFrameVectorData {
    ODataVectorInfo   vectorInfo;
    OSensorFrameInfo  info[1];

    void SetNumData(size_t ndata){vectorInfo.numData = ndata; }
    OSensorFrameInfo* GetInfo(int index){return &info[index];}
    OSensorFrameData* GetData(int index) {
        return (OSensorFrameData*) ((byte*)&vectorInfo+info[index].dataOffset);
    }
};
```

Header file
#include <OPENR/ODataFormats.h>
OSensorFrameInfo

Description
This contains the type of element of OSensorFrameVectorData, OPrimitiveID, the number of frames in sensor data and the offset to sensor data.

Structure

```c
struct OSensorFrameInfo {
    ODataType    type;
    OPrimitiveID primitiveID;
    longword     frameNumber;
    size_t       numFrames;
    size_t       frameSize;
    size_t       dataOffset;
    size_t       dataSize;
    longword     padding[1];
}
```

Header file

```c
#include <OPENR/ODataFormats.h>
```

Members

- **type**: This is the type of sensor data. All the types are defined in ODataFormats.h.
- **primitiveID**: This is the ID number of a CPC Primitive that obtains sensor data.
- **frameNumber**: This is the frame sequence number when the first data of a corresponding OSensorFrameData is obtained.
- **numFrames**: This is the number of valid frames of sensor data that OSensorFrameData keeps.
- **frameSize**: This is the size (16 bytes) of a sensor data for one frame, which OSensorFrameData keeps.
- **dataOffset**: This is the offset to OSensorFrameData corresponding to OSensorFrameInfo. This offset is from the starting address of OSensorFrameVectorData.
- **dataSize**: This is a data size (256 bytes) of OSensorFrameData corresponding to OSensorFrameInfo.
- **padding[1]**: Padding to adjust the total number of bytes.
OSensorFrameData

Description
This is the main part of sensor data. OSensorValue is a generic data structure for one frame. It is used by casting to the various types of sensor data. For example, in case of a joint data, OSensorFrameData is cast to OJointValue. In case of an acceleration sensor, OSensorFrameData is cast to OAcceleration.

Structure

```
struct OSensorFrameData {
    OSensorValue frame[osensorframeMAX_FRAMES];
};
```

Header file
```
#include <OPENR/ODataFormats.h>
```

Members
- frame[osensorframeMAX_FRAMES]
  This is sensor data. OSensorFrameData can have data for the maximum number of osensorframeMAX_Frames (=16) frames. The number of valid frames is specified by numFrames in OSensorFrameinfo.

OAcceleration

Description
This is acceleration data. The units are in $10^{-6}$m/sec$^2$.

Structure

```
struct OAcceleration {
    slongword value;
    word signal;
    word padding[5];
};
```

Header file
```
#include <OPENR/ODataFormats.h>
```

Members
- value
  This value is converted from a signal value, by using a calibration table, obtained from an acceleration sensor. The units are in $10^{-6}$m/sec$^2$.
- signal
  This is an A/D signal value obtained from an acceleration sensor.
- padding[5]
  Padding to adjust the total number of bytes.
OAngularVelocity

Description
This is angular velocity data. The units are in $10^{-6}$rad/s.

Structure

```c
struct OAngularVelocity {
    slongword   value;
    word        signal;
    word        padding[5];
};
```

Header file
#include <OPENR/ODataFormats.h>

Members

value This is a value converted from a signal value, by using a calibration table, obtained from an angular velocity sensor. The units are in $10^{-6}$ rad/s.

signal This is an A/D signal value that was obtained from the angular velocity sensor.

padding[5] Padding to adjust the total number of bytes.

OTemperature

Description
This is temperature data. The units are in $10^{-6}$°C.

Structure

```c
struct OTemperature {
    slongword   value;
    word        signal;
    word        padding[5];
};
```

Header file
#include <OPENR/ODataFormats.h>

Members

value This is a value converted from a signal value, by using a calibration table, obtained from a temperature sensor. The units are in $10^{-6}$ °C.

signal This is an A/D signal value that was obtained from a temperature sensor.

padding[5] Padding to adjust the total number of bytes.
**OForte**

**Description**
This is force data. The units are in $10^6$ N.

**Structure**

```c
struct OForce {
    slongword   value;
    word        signal;
    word        padding[5];
};
```

**Header file**

```c
#include <OPENR/ODataFormats.h>
```

**Members**

- **value**
  This is a value converted from a signal value, by using a calibration table, obtained from a sensor. The units are in $10^6$ N.
- **signal**
  This is an A/D signal value that was obtained from a sensor.
- **padding[5]**
  Padding to adjust the total number of bytes.

**OPressure**

**Description**
This is pressure data. The units are in $10^6$ Pa (N/m²).

**Structure**

```c
struct OPressure {
    slongword   value;
    word        signal;
    word        padding[5];
};
```

**Header file**

```c
#include <OPENR/ODataFormats.h>
```

**Members**

- **value**
  This is a value converted from a signal value, by using a calibration table, obtained from a pressure sensor. The units are in $10^6$ Pa.
- **signal**
  This is an A/D signal value that was obtained from a pressure sensor.
- **padding[5]**
  Padding to adjust the total number of bytes.
### OLength

**Description**
This is length data. The units are in \(10^6\) m.

**Structure**

```c
struct OLength {
    slongword value;
    word signal;
    word padding[5];
};
```

**Header file**

```c
#include <OPENR/ODataFormats.h>
```

**Members**

- **value**: This is a value converted from a signal value, by using a calibration table, obtained from a sensor. The units are in \(10^6\) m.
- **signal**: This is an A/D signal value that was obtained from a sensor.
- **padding[5]**: Padding to adjust the total number of bytes.

### OSwitchStatus

**Description**
This is the status of a switch.

**Structure**

```c
struct OSwitchStatus {
    OSwitchValue value;
    word signal;
    word padding[5];
};
```

**Header file**

```c
#include <OPENR/ODataFormats.h>
```

**Members**

- **value**: This is the status of a switch, converted from an A/D signal value obtained from a switch. It is either oswitchON or oswitchOFF.
- **signal**: This is an A/D signal value obtained from a switch.
- **padding[5]**: Padding to adjust the total number of bytes.
OJointValue

Description
This is joint data. The units are in \(10^{-6}\) rad for a revolute joint.

Structure

```c
struct OJointValue {
    slongword   value;
    word        signal;
    sword       pwmDuty;
    slongword   refValue;
    word        refSignal;
    word        padding[1];
};
```

Header file

```c
#include <OPENR/ODataFormats.h>
```

Members

- **value**: The feedback signal of a joint is converted into “value” by using a calibration table. The units are in \(10^{-6}\) rad for a revolute joint.
- **signal**: This is the feedback signal of a joint.
- **pwmDuty**: This is the PWM signal value.
- **refValue**: This is the indicated value when a sensor data is obtained. The units are in micro radians.
- **refSignal**: This is a 10-bit value after a calibration conversion.
- **padding[1]**: Padding to adjust the total number of bytes.
4.2.3 OFbkImageVectorData

**Description**
This is image data.

**Structure**

```c
struct OFbkImageVectorData {
    ODataVectorInfo    vectorInfo;
    OFbkImageInfo      info[1];

    void SetPrimitiveID(ORemotePrimitiveID primitiveID) {
        for (int i = 0; i < vectorInfo.numData; i++)
            info[i].primitiveID = primitiveID;
    }

    OFbkImageInfo* GetInfo(int index) { return &info[index]; }
    byte* GetData(int index) { return ((byte*)&vectorInfo + info[index].dataOffset); }
};
```

**Header file**
```
#include <OPENR/ODataFormats.h>
```

**OFbkImageInfo**

**Description**
This is the image information. This is the data structure that holds a YCrCb image and a CDT image.

**Structure**

```c
struct OFbkImageInfo {
    ODataType              type;
    ORemotePrimitiveID     primitiveID;
    longword               frameNumber;
    size_t                 dataOffset;
    size_t                 dataSize;
    size_t                 width;
    size_t                 height;
    size_t                 padding[1];
};
```

**Header file**
```
#include <OPENR/ODataFormats.h>
```

**Members**

- **type**
  This is the data type. `odataFBK_YCrCb` or `odataFBK_CDT` can be used.

- **primitiveID**
  This is the primitiveID of the FbkImageSensor that captured the image data.

- **frameNumber**
  This is the frame sequence number when the image was obtained.

- **dataOffset**
  This is an offset from the starting address of the shared memory to the image data.

- **dataSize**
  This is the size of the image data.

- **width**
  This is the number of pixel columns of the image data.

- **height**
  This is the number of pixel rows of the image data.

- **padding[1]**
  Padding to adjust the total number of bytes.
OFbkImage

Function
This class accesses the Y, Cr, Cb, and CDT images in OFbkImageVectorData.

Header file
#include<OPENR/OFbkImage.h>

Library
libOPENR.a

Syntax
OFbkImage(OFbkImageInfo* info, byte* data, OFbkImageBand band)

Description
This is the constructor for OFbkImage. You specify the pointer, obtained by OFbkImageVectorData::GetInfo(), for info, and also specify the pointer, obtained by OFbkImageVectorData::GetData(), for data.

When the arguments of OFbkImageVectorData::GetInfo() and OFbkImageVectorData::GetData() are either ofbkimageLAYER_H, ofbkimageLAYER_M, ofbkimageLAYER_L, you must specify one of the following: ofbkimageBAND_Y, ofbkimageBAND_Cr, ofbkimageBAND_Cb for band. When the argument is ofbkimageLAYER_C, specify ofbkimageBAND_CDT.

Parameters
info Pointer to OFbkImageInfo
data Pointer to image data
band The band of image data

IsValid()

Syntax
bool IsValid()

Description
This checks if OFbkImage is valid or not. False is returned when the constructor was called with invalid parameters.

Parameters
none

Returned value
ture valid
false invalid

Pointer()

Syntax
byte* Pointer()

Description
This returns the pointer to an image data.

Parameters
none

Returned value
The pointer to an image data
Width()

**Syntax**
```
int Width()
```

**Description**
This returns the width of an image.

**Parameters**
none

**Returned value**
The width of an image

---

Height()

**Syntax**
```
int Height()
```

**Description**
This returns the height of an image.

**Parameters**
none

**Returned value**
The height of an image

---

Skip()

**Syntax**
```
int Skip()
```

**Description**
This returns the number of bytes to skip when a pointer is moved to the next line of an image.

**Parameters**
none

**Returned value**
The number of bytes to skip when a pointer is moved to the next line of an image.

---

Pixel()

**Syntax**
```
byte Pixel(int x, int y)
```

**Description**
This returns the pixel value of an image with coordinate (x, y). The (0,0) coordinate is the upper-left corner of the image.

**Parameters**
x x coordinate of an image
y y coordinate of an image

**Returned value**
The pixel value of an image with coordinate (x, y)
FieldCounter()

Syntax
word FieldCounter()

Description
A counter number is stored in the last line of an image in each layer. The counter number is incremented in each image. FieldCounter() returns this counter.

Parameters
none

Returned value
The counter number of an image

ColorFrequency()

Syntax
byte ColorFrequency(OCdtChannel chan)

Description
The color frequency information (pixel number/16), which was detected with a color detection scheme, is stored in the last line of an image in each layer. ColorFrequency() returns the color frequency.

Parameters
chan CDT channel

Returned value
The color frequency (pixel number/16), which was detected with a color detection scheme
4.3 Communication with OVirtualRobotAudioComm

The following is the data for communication with OVirtualRobotAudioComm.

OSoundVectorData  Sound data

The data is created in a shared memory segment. The contents of this data are placed in the following order: ODataVectorInfo as a common header, the array of the information block about each element, and the array of the data body.

4.3.1 OSoundVectorData

Description

This is the data structure that holds sound data. It consists of the vectorInfo, followed by an array of OSoundInfo with number of elements determined by vectorInfo.maxNumData, and the byte string of sound data.

Structure

```
struct OSoundVectorData {
    ODataVectorInfo    vectorInfo;
    OSoundInfo         info[1];

    void SetNumData(size_t ndata) {
        vectorInfo.numData = ndata;
    }
    OSoundInfo* GetInfo(int index) {return &info[index];}
    byte* GetData(int index) {
        return ((byte*)&vectorInfo + info[index].dataOffset);
    }
};
```

Header file

```
#include <OPENR/ODataFormats.h>
```

OSoundInfo

Description

This is the data structure that holds sound data information.

Structure

```
struct OSoundInfo {
    ODataType             type;
    OPrimitiveID          primitiveID;
    longword              frameNumber;
    size_t                frameSize;
    size_t                dataOffset;
    size_t                maxDataSize;
    size_t                dataSize;
    OSoundFormat          format;
    OSoundChannel         channel;
    word                  samplingRate;
    word                  bitsPerSample;
    size_t                actualDataSize;
    longword              padding[6];

    void Set(ODataType t, OPrimitiveID id, size_t dsize) {
        type        = t;
        primitiveID = id;
        dataSize    = dsize;
    }
};
```
**Header file**

```c
#include <OPENR/ODataFormats.h>
```

**Members**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>type</code></td>
<td>This is the data type. <code>odataSOUND</code> is used.</td>
</tr>
<tr>
<td><code>OPrimitiveID</code></td>
<td>This is the ID number of the CPC Primitive which inputs/outputs sound data. To output sound, <code>OPrimitiveID</code> of a speaker is used. To input sound, <code>OPrimitiveID</code> of a microphone is used.</td>
</tr>
<tr>
<td><code>frameNumber</code></td>
<td>For the output of sound, <code>frameNumber</code> is the frame sequence number when <code>OVirtualRobot</code> processes the first frame of sound. For input of sound, the frame sequence number when data was input is used.</td>
</tr>
<tr>
<td><code>frameSize</code></td>
<td>This is the size of 1 frame of sound data.</td>
</tr>
<tr>
<td><code>dataOffset</code></td>
<td>This is an offset to the byte string of sound data corresponding to <code>OSoundInfo</code>. This is an offset from the starting address of <code>OSoundVectorData</code>.</td>
</tr>
<tr>
<td><code>maxDataSize</code></td>
<td>This is the maximum size of the byte string of sound data corresponding to <code>OSoundInfo</code>.</td>
</tr>
<tr>
<td><code>dataSize</code></td>
<td>This is the size of the valid byte string of sound data.</td>
</tr>
<tr>
<td><code>format</code></td>
<td>This is the format of the sound data. Currently, only <code>osoundformatPCM</code> is supported.</td>
</tr>
<tr>
<td><code>channel</code></td>
<td>The number of channels in the sound data.</td>
</tr>
<tr>
<td><code>samplingRate</code></td>
<td>The sampling rate</td>
</tr>
<tr>
<td><code>bitsPerSample</code></td>
<td>This is the number of bits per one sample in the sound data.</td>
</tr>
<tr>
<td><code>actualDataSize</code></td>
<td>This is the size of the sound data transferred from a device.</td>
</tr>
<tr>
<td><code>padding [6]</code></td>
<td>Padding to adjust the total number of bytes.</td>
</tr>
</tbody>
</table>
4.4 Others

“Others” includes the following data.

**OCdtVectorData**

CDT table data

This data is created in a shared memory. Each data has a common header
ODataVectorInfo, followed by an array containing an information block about each
element, and an array of the main body of data.

4.4.1 OCdtVectorData

**Description**

This is a data structure that holds a color detection table. It can have a maximum of
ocdNUM_CHANNELS (=8) tables. The number of valid OCdtInfo is specified by
ODataVectorInfo::numData.

**Structure**

```c
struct OCdtVectorData{
    ODataVectorInfo vectorInfo;
    OCdtInfo info[ocdtNUM_CHANNELS];

    void SetNumData(size_t ndata) { vectorInfo.numData = ndata; }
    OCdtInfo* GetInfo(int index) { return &info[index]; }
};
```

**Header file**

#include <OPENR/ODataFormats.h>

**OCdtInfo**

**Description**

In the color detection table, Y (a luminance signal) is divided into 32 segments, and
Crmax, Crmin, Cbmax and Cbmin are specified for each segment of Y.
The values of Cr and Cb are offset binary ranging from 0x0 to 0xff.

**Structure**

```c
struct OCdtInfo {
    ODataType type;
    OPrimitiveID primitiveID;
    OCdtChannel channel;
    longword table[ocdtMAX_Y_SEGMENT];
    longword padding;

    void Init(OPrimitiveID prmID, OCdtChannel chan) {
        type = odataCDT;
        primitiveID = prmID;
        channel = chan;
        for (int i = 0; i < ocdtMAX_Y_SEGMENT; i++) table[i] = ocdtINIT;
    }

    void Set(int y_segment,
             byte cr_max, byte cr_min, byte cb_max, byte cb_min) {
        longword crMax = (longword)cr_max;
        longword crMin = (longword)cr_min;
        longword cbMax = (longword)cb_max;
        longword cbMin = (longword)cb_min;
        crMax = (crMax << 8) & ocdtCr_MAX_MASK;
        crMin = (crMin) & ocdtCr_MIN_MASK;
        cbMax = (cbMax << 24) & ocdtCb_MAX_MASK;
        cbMin = (cbMin) & ocdtCb_MIN_MASK;
    }
};
```
cbMin = (cbMin << 16) & ocdtCb_MIN_MASK;

table[y_segment] = crMax | crMin | cbMax | cbMin;

};

Header file
#include <OPENR/ODataFormats.h>

Members
  type  This is the data type. odataCDT is used.
  primitiveID The PrimitiveID of OFbkImageSensor that the CDT is set to.
  channel This is a channel of the CDT that a table is set to.
  padding Padding to adjust the total number of bytes.
Chapter 5 OPEN-R API

OPENR::OpenPrimitive()
Syntax
OStatus OPENR::OpenPrimitive(char* locator, OPrimitiveID* primitiveID)

Description
This opens a CPC Primitive and gets its OPrimitiveID. If it fails, oprimitiveID_UNDEF is returned to primitiveID.

Parameters
locator   CPC Primitive Locator
primitiveID  CPC Primitive ID

Returned value
oSUCCESS    Success
oNOT_FOUND  CPC Primitive corresponding to the locator does not exist.
 oOPEN_FAILURE Fails to open the CPC Primitive.
 oINVALID_ARG locator is a NULL pointer
 oFAIL       Failure

OPENR::ClosePrimitive()
Syntax
OStatus OPENR::ClosePrimitive(OPrimitiveID primitiveID)

Description
This closes a CPC Primitive.

Returned value
oSUCCESS    Success
oINVALID_PRIMITIVE_ID An invalid primitiveID
OPENR::ControlPrimitive()

Syntax
OStatus OPENR::ControlPrimitive(OPrimitiveID primitiveID,
OPrimitiveRequest request, void* param, size_t paramSize,
void* result, size_t resultSize)

Description
This sets parameters of the CPC Primitive. param, paramSize, result and resultSize
are specified by request. When it is not necessary to specify a parameter, specify 0.
The following are the kinds of requests.

oprmreqSPEAKER_MUTE_ON
oprmreqSPEAKER_MUTE_OFF
oprmreqMIC_UNI
oprmreqMIC_OMNI
oprmreqMIC_ALC_ON
oprmreqMIC_ALC_OFF
oprmreqCAM_SET_WHITE_BALANCE
oprmreqCAM_SET_GAIN
oprmreqCAM_SET_SHUTTER_SPEED
oprmreqSPEAKER_SET_SOUND_TYPE
oprmreqSPEAKER_GET_SOUND_TYPE

The following are samples of function calls.
/* Mute ON */
OPENR::ControlPrimitive(spekerID, oprmreqSPEAKER_MUTE_ON, 0, 0, 0, 0);

/* Mute OFF */
OPENR::ControlPrimitive(spekerID, oprmreqSPEAKER_MUTE_OFF, 0, 0, 0, 0);

/* UNI MIC */
OPENR::ControlPrimitive(micID, oprmreqMIC_UNI, 0, 0, 0, 0);

/* OMNI MIC */
OPENR::ControlPrimitive(micID, oprmreqMIC_OMNI, 0, 0, 0, 0);

/* ALC ON */
OPENR::ControlPrimitive(micID, oprmreqMIC_ALC_ON, 0, 0, 0, 0);

/* ALC OFF */
OPENR::ControlPrimitive(micID, oprmreqMIC_ALC_OFF, 0, 0, 0, 0);

/* Set white balance */
OPrimitiveControl_CameraParam wb(ocamparamWB_OUTDOOR_MODE);
OPENR::ControlPrimitive(prmID, oprmreqCAM_SET_WHITE_BALANCE,
&wb, sizeof(wb), 0, 0);

/* Camera gain */
OPrimitiveControl_CameraParam gain(ocamparamGAIN_MID);
OPENR::ControlPrimitive(prmID, oprmreqCAM_SET_GAIN,
&gain, sizeof(gain), 0, 0);

/* Shutter speed */
OPrimitiveControl_CameraParam shutter(ocamparamSHUTTER_FAST);
OPENR::ControlPrimitive(prmID, oprmreqCAM_SET_SHUTTER_SPEED,
&shutter, sizeof(shutter), 0, 0);

/* Set sound data type */
OPrimitiveControl_SpeakerSoundType soundType(ospksndMONO16K16B) ;
OPENR::ControlPrimitive(speakerID, oprmreqSPEAKER_SET_SOUND_TYPE,
&soundType, sizeof(soundType)) ;
/* Get sound data type */
OPrimitiveControl_SpeakerSoundType soundType;
OPENR : :ContorlPrimitive(speakerID, prmreqSPEAKER_GET_SOUND_TYPE,
&soundType, sizeof (soundType) );

Parameters
primitiveID OPrimitiveID
request Control request
param Parameter data
paramSize Size of parameter data
result Result data
resultSize Size of result data

Returned value
oSUCCESS Success
oINVALID_PRIMITIVE_ID An invalid primitiveID
oINVALID_ARG request and param are invalid.

OPENR::NewCommandVectorData()
Syntax
OStatus OPENR::NewCommandVectorData(size_t numCommands,
MemoryRegionID* memID, OCommandVectorData** baseAddr)
Description
This reserves shared memory for OCommandVectorData.
vectorInfo.numData is initialized to 0. Set the valid number of elements with
SetNumData().

Parameters
numCommands The number of elements in OCommandData
memID MemoryRegionID of the shared memory for
OCommandVectorData
baseAddr Pointer to OCommandVectorData

Returned value
oSUCCESS Success
oNO_MEMORY Fails to reserve shared memory

OPENR::DeleteCommandVectorData()
Syntax
OStatus OPENR::DeleteCommandVectorData(MemroryRegionID memID)
Description
This releases the shared memory for OCommandVectorData.

Parameters
memID MemoryRegionID of the shared memory for
OCommandVectorData

Returned value
oSUCCESS Success
oFAIL Failure

OPENR::NewSoundVectorData()
Syntax
OStatus NewSoundVectorData(size_t numSounds, size_t dataSize,
MemoryRegionID* memID, OSoundVectorData** baseAddr)
**Description**
This reserves shared memory for OSoundVectorData. vectorInfo.numData is initialized to 0. Set the valid number of elements with SetNumData().

**Parameters**
- `numSounds`: The number of elements in sound data
- `dataSize`: Size of each sound data
- `memID`: MemoryRegionID of the shared memory for OSoundVectorData
- `baseAddr`: Pointer to OSoundVectorData

**Returned value**
- oSUCCESS: Success
- oNO_MEMORY: Fails to reserve shared memory.

**OPENR::DeleteSoundVectorData()**

**Syntax**
OStatus DeleteSoundVectorData(MemoryRegionID memID)

**Description**
This releases the shared memory for OSoundVectorData.

**Parameters**
- `memID`: MemoryRegionID of the shared memory for OSoundVectorData

**Returned value**
- oSUCCESS: Success
- oINVALID_ARG: An invalid memID
- oFAIL: Failure

**OPENR::NewCdtVectorData()**

**Syntax**
OStatus NewCdtVectorData(MemoryRegionID* memID, OCdtVectorData** baseAddr)

**Description**
This reserves shared memory for OCdtVectorData. vectorInfo.numData is initialized to 0. Set the valid number of elements with SetNumData().

**Parameters**
- `memID`: MemoryRegionID of the shared memory for OCdtVectorData
- `baseAddr`: Pointer to OCdtVectorData

**Returned value**
- oSUCCESS: Success
- oNO_MEMORY: Fails to reserve shared memory.

**OPENR::DeleteCdtVectorData()**

**Syntax**
OStatus DeleteCdtVectorData(MemoryRegionID memID)

**Description**
This releases the shared memory for OCdtVectorData.

**Parameters**
- `memID`: MemoryRegionID of the shared memory for OCdtVectorData

**Returned value**
- oSUCCESS: Success
- oFAIL: Failure
OPENR::SetCdtVectorData()

Syntax
OStatus SetCdtVectorData(MemoryRegionID memID)

Description
This sets OCdtVectorData to FbkImageSensor.

Parameters
memID  MemoryRegionID of the shared memory for OCdtVectorData.

Returned value
oSUCCESS   Success
oINVALID_ARG  An invalid OCdtInfo::channel
oINVALID_PRIMITIVE_ID An invalid primitiveID
oINVALID_DATA_TYPE type is not odataCDT_VECTOR.
ofAIL    Failure, excluding the above

OPENR::EnableJointGain()

Syntax
OStatus EnableJointGain(OPrimitiveID primitiveID)

Description
This sets the gain of a joint to effective. When the gain of a joint is effective and
OPENR::SetJointGain() or OPENR::SetDefaultJointGain() is executed,
the PID gain is set to a servo device. When oprimitiveID_UNDEF is specified to
primitiveID, the gain of all joints opened by OPENR::OpenPrimitive() become
effective.

Parameters
primitiveID  OPrimitiveID of a Joint or oprimitiveID_UNDEF

Returned value
oSUCCESS    Success
oINVALID_PRIMITIVE_ID An invalid primitiveID
oALERT_JOINT_UNCONTROLLABLE Impossible to control due to the break of
a potentiometer.

OPENR::DisableJointGain()

Syntax
OStatus DisableJointGain(OPrimitiveID primitiveID)

Description
This sets the gain of a joint to 0 and ineffective. If oprimitiveID_UNDEF is specified
to primitiveID, it sets the gain of all joints opened by OPENR::OpenPrimitive() to 0
and ineffective.

Parameters
primitiveID  OPrimitiveID of a joint or oprimitiveID_UNDEF

Returned value
oSUCCESS   Success
oINVALID_PRIMITIVE_ID An invalid primitiveID
ofAIL    Failure
OPENR::SetJointGain()

Syntax
OStatus SetJointGain(OPrimitiveID primitiveID, word pg, word ig, word dg, word ps, word is, word ds)

Description
This sets the gain of a joint. When the gain of a joint is ineffective, no gain is set and oGAIN_DISABLED is returned. If primitiveID_UNDEF is specified to primitiveID, it sets the gain of all joints opened by OPENR::OpenPrimitive(). oSUCCESS is returned when setting of the gain has succeeded.

Parameters
primitiveID OprimitiveID of a joint or primitiveID_UNDEF
pg PGAIN coefficient
ig IGAIN coefficient
dg DGAIN coefficient
ps PSHIFT coefficient
is ISHIFT coefficient
ds DSHIFT coefficient

Returned value
oSUCCESS Success
oINVALID_PRIMITIVE_ID An invalid primitiveID
oGAIN_DISABLED The state of an ineffective gain
oALERT_JOINT_UNCONTROLLABLE Impossible to control due to the break of a potentiometer.
oFAIL Failure

OPENR::RegisterDefaultJointGain()

Syntax
OStatus RegisterDefaultJointGain(OPrimitiveID primitiveID, word pg, word ig, word dg, word ps, word is, word ds)

Description
This registers the default gain to a joint. If primitiveID_UNDEF is specified to primitiveID, it registers the default gain to all joints opened by OPENR::OpenPrimitive().

Parameters
primitiveID OprimitiveID of a joint or primitiveID_UNDEF
pg PGAIN coefficient
ig IGAIN coefficient
dg DGAIN coefficient
ps PSHIFT coefficient
is ISHIFT coefficient
ds DSHIFT coefficient

Returned value
oSUCCESS Success
oINVALID_PRIMITIVE_ID An invalid primitiveID
OPENR::SetDefaultJointGain()

Syntax
OStatus SetDefaultJointGain(OPrimitiveID primitiveID)

Description
This sets the registered default gain to a joint. When a gain is ineffective, no gain is
set and oGAIN_DISABLED is returned. If oprimitiveID_UNDEF is specified to
primitiveID, it sets the gain of all joints opened by OPENR::OpenPrimitive().
oSUCCESS is returned when the gain of a joint has successfully been set.

Parameters
primitiveID  OPrimitiveID of the joint or oprimitiveID_UNDEF

Returned value
oSUCCESS    Success
oINVALID_PRIMITIVE_ID An invalid primitiveID
oGAIN_DISABLED The gain of a joint is ineffective.
oALERT_JOINT_UNCONTROLLABLE Impossible to control due to the break
of a potentiometer.
oFAIL      Failure

OPENR::GetJointValue()

Syntax
OStatus GetJointValue(OPrimitiveID primitiveID, OJointValue* value)

Description
This gets the current value of a joint.

Parameters
primitiveID  OPrimitiveID of a joint
value        The current joint value

Returned value
oSUCCESS    Success
oINVALID_PRIMITIVE_ID An invalid primitiveID

OPENR::GetSensorValue()

Syntax
OStatus GetJointValue(OPrimitiveID primitiveID, OSensorValue* value)

Description
This gets the current value of a sensor.

Parameters
primitiveID  OPrimitiveID of a sensor
value        The current sensor value

Returned value
oSUCCESS    Success
oINVALID_PRIMITIVE_ID An invalid primitiveID
OPENR::NewSyncKey()
Syntax
OStatus OPENR::NewSyncKey(OVRSyncKey* syncKey)

Description
This is used to synchronize LED, sound, and motion so that they start at the same time. A synchronization key is issued with OPENR::NewSyncKey(), and the synchronization key is divided into the number of objects which you want to synchronize, by OPENR::DivideSyncKey(). The maximum number of synchronization keys is 8. When you have exceeded 8, an ovrsynckeyUNDEF is substituted for the synchronization key, and oNO_SYNC_KEY is returned.

Parameters
syncKey  Synchronization key

Returned value
oSUCCESS  Success
oNO_SYNC_KEY  The maximum number of synchronization keys (8) have been issued.

OPENR::CancelSyncKey()
Syntax
OStatus OPENR::CancelSyncKey(OVRSyncKey syncKey)

Description
This cancels a synchronization key.

Parameters
syncKey  Synchronization key

Returned value
oSUCCESS  Success
oINVALID_SYNC_KEY  An invalid synckey

OPENR::DivideSyncKey()
Syntax
OStatus OPENR::DivideSyncKey(OVRSyncKey syncKey,
    OVRSyncKey* key1, OVRSyncKey* key2)

Description
This divides a synchronization key

Parameters
syncKey  Synchronization key before division
key1, key2  Synchronization key after division

Returned value
oSUCCESS  Success
oFAIL  Failure
**OPENR::SetMotorPower()**

**Syntax**

```c
OStatus OPENR::SetMotorPower(OPower power)
```

**Description**

This controls the power to motors. `opowerOFF` or `opowerON` is specified to `power`.

**Parameters**

- `power`  opowerON or opowerOFF

**Returned value**

- oSUCCESS  Success
- oFAIL  Failure

---

**OPENR::Shutdown()**

**Syntax**

```c
OStatus OPENR::Shutdown(const OBootCondition& bootCondition)
```

**Description**

This sets the specified `bootCondition`, and then the shutdown procedure starts.

**Parameters**

- `bootCondition`  boot condition

**Returned value**

- oSUCCESS  Success
- oFAIL  Failure
- oNOT_FOUND  The system object does not exist.
**OPENR::GetBootCondition()**

**Syntax**

```cpp
OStatus OPENR::GetBootCondition(OBootCondition* bootCondition)
```

**Description**

This gets the boot condition.

```cpp
struct OBootCondition {
    word bitmap;
    time_t bootTime;
    longword bootTimeType;
    byte vibrationLevel;
};
```

The boot condition is saved to bitmap. bootTime, bootTimeType, and vibrationLevel are invalid.

Types of boot conditions

- `obcbBOOT_TIMER` = 0x0001
  Starts on scheduled time.
- `obcbVIBRATION_DETECTED` = 0x0002
  Starts with vibration.
- `obcbPAUSE_SW` = 0x0004
  Starts with the pause button.
- `obcbSTATION_CONNECTED` = 0x0008
  Starts when connected to the station.
- `obcbSTATION_DISCONNECTED` = 0x0010
  Starts when disconnected from the station.
- `obcbBATTERY_CAPACITY_FULL` = 0x0020
  Starts when a battery is fully charged.
- `obcbREQ_FROM_STATION` = 0x0040
  Reserved

**Parameters**

- `bootCondition` Boot condition

**Returned value**

- `oSUCCESS` Success
- `oFAIL` Failure
- `oNOT_FOUND` A system object does not exist.
OPENR::GetPowerStatus()

Syntax

OStatus OPENR::GetPowerStatus(OPowerStatus* powerStatus)

Description

This gets the hardware status, which is defined by the following structure.

```c
struct OPowerStatus {
    longword robotStatus;
    word batteryStatus;
    word remainingCapacity;
    word temperature;
    word fullyChargedCapacity;
    word voltage;
    sword current;
    sbyte timeDif;
    byte volume;
};
```

The following are the units for each member.

- **remainingCapacity**: The battery remaining capacity (%, 0 - 100%)
- **temperature**: The battery temperature (0.1Kelvin, 0 - 500.0Kelvin)
- **fullyChargedCapacity**: The battery capacity when it is fully charged (mAh)
- **voltage**: The battery voltage (mV, 0 - 65535mV)
- **current**: The battery current (mA, –32768 - 32767mA)
- **timeDif**: The time difference from UTC (Universal CoordinateTime)
- **volume**: Volume. One of 0, 1, 2, 3.
**robotStatus** Indicates general hardware status.

- **orsbPAUSE** = 0x00000001
  - Pause switch is on.
- **orsbMOTOR_POWER** = 0x00000002
  - Motor power is on.
- **orsbVIBRATION_DETECT** = 0x00000004
  - Vibration detected.
- **orsbEX_PORT_CONNECTED** = 0x00000008
  - Connected to an external connector. External connectors include connectors of the AC adaptor and the station.
- **orsbSTATION_CONNECTED** = 0x00000010
  - Connected to the station.
- **orsbEX_POWER_CONNECTED** = 0x00000020
  - Connected to an external power supply.
- **orsbBATTERY_CONNECTED** = 0x00000040
  - Battery is connected.
- **orsbBATTERY_CHARGING** = 0x00000080
  - Battery is charging.
- **orsbBATTERY_CAPACITY_FULL** = 0x00000100
  - Battery capacity full.
- **orsbBATTERY_CAPACITY_LOW** = 0x00000200
  - Battery capacity low.
- **orsbBATTERY_OVER_CURRENT** = 0x00000400
  - Battery current too high.
- **orsbBATTERY_OVER_TEMP_DISCHARGING** = 0x00000800
  - Battery temperature on discharging is too high.
- **orsbBATTERY_OVER_TEMP_CHARGING** = 0x00001000
  - Battery temperature on charging is too high.
- **orsbBATTERY_ERROR_OF_CHARGING** = 0x00002000
  - Error on battery charging.
- **orsbERROR_OF_PLUNGER** = 0x00004000
  - Error on plunger. Unable to lock battery.
- **orsbOPEN_R_POWER_GOOD** = 0x00008000
  - Power supplied to OPEN-R Bus system (3.3V).
- **orsbERROR_OF_FAN** = 0x00010000
  - Error on cooling fan.
- **orsbDATA_STREAM_FROM_STATION** = 0x00020000
  - The station has written data onto the datastream region.
- **orsbREGISTER_UPDATED_BY_STATION** = 0x00040000
  - The station has updated some of the register region.
- **orsbRTC_ERROR** = 0x00080000
  - Error on RTC (Real Time Clock).
- **orsbRTC_OVERFLOW** = 0x00100000
  - Overflow occurred in RTC. (Note 1)
- **orsbRTC_RESET** = 0x00200000
  - Indicates RTC has been reset. (Note 2)
- **orsbRTC_SET** = 0x00400000
  - Indicates time-setting to RTC has been performed. This flag will be cleared on the notification to the entry that is monitoring this flag.
- **orsbSPECIAL_MODE** = 0x00800000
  - Required to enter special mode.
- **orsbBMN_DEBUG_MODE** = 0x01000000
  - Indicates BMN microcontroller is in the debug mode.
- **orsbCHARGER_STATUS** = 0x02000000
  - Indicates the charging circuit in AIBO is on.
- **orsbPLUNGER** = 0x04000000
  - Indicates the plunger is locked.
- **orsbSUSPENDED** = 0x08000000
  - reserved
- **orsbSPECIAL_DATA_READ_REQ** = 0x10000000
  - reserved
Note 1
The time is represented by the number of seconds elapsed since 2000/1/1 0:00. The data length is 32-bits (signed). Therefore, if the value exceeds 0x7fffffff, the elapsed seconds will be negative and unable to represent the time properly. Starting from year 2000, it is possible to represent time until around year 2068. This flag will be cleared when the time is set, by using the LCD panel on AIBO, via a command by the CPU, or via the station.

Note 2
If it is not charged for a long period, the local power of the RTC will be exhausted and the time kept in the RTC will be lost. This flag will also be cleared when the time is set, using the methods described above.

batteryStatus  Indicates battery status.
obsbERROR_CODE_MASK  = 0x000F
  Error code returned by the battery.
obsbFULLY_DISCHARGED  = 0x0010
  Indicates the battery is fully discharged.
obsbFULLY_CHARGED  = 0x0020
  Indicates the battery is fully charged.
obsbDISCHARGING  = 0x0040
  Indicates the battery is discharging.
obsbINITIALIZED  = 0x0080
  Always one
obsbREMAINING_TIME_ALARM  = 0x0100
  Indicates the operable battery time is short.
obsbREMAINING_CAPACITY_ALARM  = 0x0200
  Indicates remaining capacity of the battery is low. This is different from obsbBATTERY_CAPACITY_LOW in robotStatus.
obsbRESERVED0  = 0x0400
  reserved
obsbTERMINATED_DISCHARGING_ALARM  = 0x0800
  Indicates discharging is terminated.
obsbOVER_TEMP_ALARM  = 0x1000
  Temperature is too high.
obsbRESERVED1  = 0x2000
  reserved
obsbTERMINATED_CHARGING_ALARM  = 0x4000
  Indicates that the battery charging is terminated.
obsbOVER_CHARGED_ALARM  = 0x8000
  Alarm for excessive charging

Parameters
powerStatus  This is the power status.

Returned value
oSUCCESS  Success
oFAIL  Failure
oNOT_FOUND  A system object does not exist.
OPENR::ObservePowerStatus()

Syntax

OStatus OPENR::ObservePowerStatus(const OPowerStatus & notifyStatus,
const OServiceEntry & entry)

Description

When a parameter specified by notifyStatus is changed, the specified ‘entry’ will be notified of the change. In NotifyStatus, fullyChargedCapacity, ‘voltage’, or ‘current’ cannot be monitored for their changes. For robotStatus and batteryStatus, a notification will occur when a specified bit is changed. For remainingCapacity, temperature, timeDif, and volume, the following symbolic constants are defined in OPower.h. Specifying opso*_NOTIFY_EVERY_CHANGE for a parameter indicates notification of changes of this parameter. Specifying opso*_NOT_NOTIFY for a parameter indicates not to notify when this parameter is changed. A value excluding the above two indicates notification when the parameter’s value becomes the specified value. The notified message structure is OPowerStatusMessage.

Symbolic constants defined in OPower.h

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsoTEMPERATURE_NOTIFY_EVERY_CHANGE</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>opsoTEMPERATURE_NOT_NOTIFY</td>
<td>0xFFFE</td>
</tr>
<tr>
<td>opsoREMAINING_CAPACITY_NOTIFY_EVERY_CHANGE</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>opsoREMAINING_NOT_NOTIFY</td>
<td>0xFFFE</td>
</tr>
<tr>
<td>opsoTIME_DIF_NOTIFY_EVERY_CHANGE</td>
<td>0xFF</td>
</tr>
<tr>
<td>opsoTIME_DIF_NOT_NOTIFY</td>
<td>0xFE</td>
</tr>
<tr>
<td>opsoVOLUME_NOTIFY_EVERY_CHANGE</td>
<td>0xFF</td>
</tr>
<tr>
<td>opsoVOLUME_NOT_NOTIFY</td>
<td>0xFE</td>
</tr>
</tbody>
</table>

Once ObservePowerStatus() is executed, the specified entry will be notified every time the power status matches the specified notifyStatus. This continues until OPENR::UnobservePowerStatus() is executed. For each bit of robotStatus and batteryStatus in notifyStatus, a notification will occur on both rising and falling edges. For remainingCapacity, temperature, timeDif, and volume, a notification will occur when each parameter’s value is changed, or it becomes the specified value. When a value is specified, a notification occurs when the parameter’s value becomes the specified value. However, a notification will not occur if the parameter’s value is changed from the specified value, nor if the parameter’s value is unchanged.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifyStatus</td>
<td>OPowerStatus structure which specifies parameters to be monitored for changes.</td>
</tr>
<tr>
<td>entry</td>
<td>Entry that is notified of a change.</td>
</tr>
</tbody>
</table>

Returned value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oSUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>oFAIL</td>
<td>Failure</td>
</tr>
<tr>
<td>oNOT_FOUND</td>
<td>A system object does not exist.</td>
</tr>
</tbody>
</table>
OPENR::UnobservePowerStatus()

Syntax
OStatus OPENR::UnobservePowerStatus(const OServiceEntry& entry)

Description
This cancels a monitoring request in OPENR::ObservePowerStatus().

Parameters
entry This is the entry to cancel the monitoring requests.

Returned value
oSUCCESS Success
oFAIL Failure
oNOT_FOUND A system object does not exist.
oINVALID_ARG An invalid entry

OPENR::FindDesignData()

Syntax
OStatus OPENR::FindDesignData(const char* keyword, ODesignDataID* dataID, byte** data, size_t* size)

Description
This retrieves a file corresponding to the keyword in a design database. If it is found, the design data file is copied to shared memory, and the starting address and ODesignDataID are returned. If you specify the reserved keyword “SYS_CPUINFO” to a parameter, you can obtain the operating frequency of the CPU, as the starting address of OCPUInfo is returned. Even if the keyword “SYS_CPUINFO” is not registered to DESIGNDB.CFG, this keyword works.

```
struct OCPUInfo{
    longword sclk;  // system clock
    longword pclk;  // pipeline clock
    lognword processID  // processor ID
    byte reserved[244]
}
```

Parameters
keyword This is the key that retrieves a design database.
dataID The design data ID
data The starting address in design data
size Size of design data in bytes

Returned value
oSUCCESS Success
oNOTFOUND The keyword or design data body does not exist.
oDESIGNDATA_SIZE_ZERO The file size for design data is 0.
oNO_MEMORY Insufficient memory
oFAIL Failure
OPENR::DeleteDesignData()

Syntax
OStatus OPENR::DeleteDesignData(ODesignDataID dataID)

Description
This releases the memory for design data.

Parameters
dataID   Design data ID

Returned value
oSUCCESS  Success
oINVALID_ARG  An invalid dataID
oFAIL   Failure

OPENR::GetRobotDesign()

Syntax
OStatus OPENR::GetRobotDesign(char* robotDesign)

Description
This gets the ‘robot design’.

Parameters
robotDesign   ‘Robot design’ string (ex. “ERS-210”)

Returned value
oSUCCESS  Success
oFAIL   Failure

OPENR::GetMemoryStickStatus()

Syntax
OStatus OPENR::GetMemoryStickStatus(OMemoryStickStatus* status)

Description
This checks the status of the AIBO Programming Memory Stick
omemorystickNOT_EXIST
No AIBO Programming Memory Stick exists.
omemorystickWRITE_PROTECTED
The write protection switch is ON.
omemorystickWRITABLE
The write protection switch is OFF.

Parameters
status   The status of the AIBO Programming Memory Stick

Returned value
oSUCCESS  Success
oFAIL   Failure
OPENR::Fatal()

Syntax
OStatus OPENR::Fatal(OFatal fatal)

Description
This sounds a warning sound with the buzzer in the BMN microcontroller, and turns off power. Specify the kind of warning sound with ‘fatal’.

Parameters
fatal The kind of warning sound.

- ofatalUNDEF “Toccata and fugue”: sound
- ofatalMEMORY_STICK AIBO Programming Memory Stick destruction error sound
- ofatalPAUSE_SW No sound

Returned value
oSUCCESS Success

OPENR::SetTime()

Syntax
OStatus OPENR::SetTime(const OTime& time)

Description
This sets the time specified by ‘time’ to the time of the RTC. If the time difference is set in ‘time’ as a value from –12 to +12 that is different from the current time difference, the time difference is also set to the BMN microcontroller.

Parameters
time The structure of time and a time difference

Returned value
oSUCCESS Success
oFAIL Failure
oNOT_FOUND A system object does not exist.

OPENR::GetTime()

Syntax
OStatus OPENR::GetTime(OTime* time)

Description
This gets the time and the time difference.

Parameters
time The structure of time and time difference

Returned value
oSUCCESS Success
oFAIL Failure
oNOT_FOUND A system object does not exist.
OPENR::SetTimeDifference()

Syntax
OStatus OPENR:: SetTimeDifference(sbyte timeDifference)

Description
This sets the time difference.

Parameters
timeDifference  Time difference

Returned value
oSUCCESS  Success
oFAIL  Failure
oNOT_FOUND  A system object does not exist.

OPENR::GetTimeDifference()

Syntax
OStatus OPENR:: GetTimeDifference(sbyte* timeDifference)

Description
This gets the time difference.

Parameters
timeDifference  Time difference

Returned value
oSUCCESS  Success
oFAIL  Failure
oNOT_FOUND  A system object does not exist.

OPENR::SetVolumeSwitch()

Syntax
OStatus SetVolumeSwitch( OV olumeSwitch volSW)

Description
This sets the level of the volume switch.

Parameters
volSW  The level of the volume switch
ovolumeSW0
ovolumeSW1
ovolumeSW2
ovolumeSW3

Returned value
oSUCCESS  Success
oFAIL  Failure
OPENR::GetVolumeSwitch()

Syntax
OStatus GetVolumeSwitch(OVolumeSwitch* volSW)

Description
This gets the level of the volume switch.

Parameters
volSW The level of the volume switch

ovolumeSW0
ovolumeSW1
ovolumeSW2
ovolumeSW3

Returned value
oSUCCESS Success
oFAIL Failure

OPENR::GetJointGain()

Syntax
OStatus GetJointGain(OPrimitiveID primitiveID, 
word* pg, word* ig, word* dg, 
word* ps, word* is, word* ds)

Description
This retrieves the gain value. If the result value is not oSUCCESS, the parameters pg, ig, dg, ps, is, and ds return 0.

Parameters
primitiveID Joint’s OPrimitiveID
pg PGAIN
ig IGAIN
dg DGAIN
ps PSHIFT
is ISHIFT
ds DSHIFT

Returned value
oSUCCESS Success
oINVALID_PRIMITIVE_ID invalid OPrimitiveID
oGAIN_DISABLED no effect of gain
oALERT_JOINT_UNCONTROLLABLE loss of control for disconnection of potentiometer
oFAIL Failure
OPENR::GetDefaultJointGain()

Syntax

OStatus GetDefaultJointGain(OPrimitiveID primitiveID,
    word* pg, word* ig, word* dg,
    word* ps, word* is, word* ds)

Description

This retrieves the default gain value. If the result value is not oSUCCESS, the parameters pg, ig, dg, ps, is, and ds return 0.

Parameters

<table>
<thead>
<tr>
<th>primitiveID</th>
<th>Joint's OPrimitiveID</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg</td>
<td>PGAIN</td>
</tr>
<tr>
<td>ig</td>
<td>IGAIN</td>
</tr>
<tr>
<td>dg</td>
<td>DGAIN</td>
</tr>
<tr>
<td>ps</td>
<td>PSHIFT</td>
</tr>
<tr>
<td>is</td>
<td>ISHIFT</td>
</tr>
<tr>
<td>ds</td>
<td>DSHIFT</td>
</tr>
</tbody>
</table>

Returned value

| oSUCCESS | Success |
| oFAIL    | Failure |
Chapter6 wireless LAN API

As for the details for the obtained data, refer to the header file of each data type or the sample program.

**ERA201D1_GetMACAddress()**

**Syntax**

```c
EtherStatus ERA201D1_GetMACAddress(EtherDriverGetMACAddressMsg* msg)
```

**Description**

This gets the MAC address.

**Parameters**

- `msg`  MAC address

**Returned value**

- `ETHER_OK` Success
- `ETHER_INVALID_PORT` No WLAN card exists.
- `ETHER_UNSUPPORTED` WLANDRV.BIN doesn't exist.

**ERA201D1_GetEtherStatistics()**

**Syntax**

```c
EtherStatus ERA201D1_GetEtherStatistics(EtherDriverGetStatisticsMsg* msg)
```

**Description**

This gets statistics of the network interface.

**Parameters**

- `msg`  statistics of the network interface

**Returned value**

- `ETHER_OK` Success
- `ETHER_INVALID_PORT` No WLAN card exists.
- `ETHER_UNSUPPORTED` WLANDRV.BIN doesn't exist.

**ERA201D1_GetWLANSettings()**

**Syntax**

```c
EtherStatus ERA201D1_GetWLANSettings(EtherDriverGetWLANSettingsMsg* msg)
```

**Description**

This gets settings of the wireless network.

**Parameters**

- `msg`  settings of the wireless network

**Returned value**

- `ETHER_OK` Success
- `ETHER_INVALID_PORT` No WLAN card exists.
- `ETHER_UNSUPPORTED` WLANDRV.BIN doesn't exist.
ERA201D1_GetWLANStatistics()

Syntax
EtherStatus ERA201D1_GetWLANStatistics
    (EtherDriverGetWLANStatisticsMsg* msg)

Description
This gets statistics for the wireless network.

Parameters
msg     statistics for the wireless network

Returned value
ETHER_OK                   Success
ETHER_INVALID_PORT        No WLAN card exists.
ETHER_UNSUPPORTED        WLANDRV.BIN doesn't exist.