Revolution SDK

Hio2lf Programming Manual

Version 1.03

The content of this document is highly confidential and should be handled accordingly.

RVL-06-0116-001-B
Released: October 17, 2008

Table of Contents

Figures

Tables

Code

Revision History

Version	Revision Date	Description
1.03	2008/08/29	• Changed "Hio2If for the Wii" to "Hio2If for the PC" in Chapter 5.
1.02	2008/05/29	• Added a description of Hio2If to Chapter 1 Introduction. • (Revised the document title in the header of the Japanese document only.)
1.01	2008/05/09	• Revised terminology. • (Changed the format of the Japanese document only.)
1.00	2006/03/09	Initial version.

1 Introduction

Hio2If is an interface that uses the HIO2 API to control communications between a Wii console (NDEV) and a PC. The Hio2If interface is not an independent library; it is a sample library that uses the HIO2 library. Its source code is provided, together with the `hio2demo` demo program.

This interface is implemented using C functions on the Wii console and C++ classes on the PC.

Hio2If uses a unique ID to manage each Expansion Interface (EXI) channel through a handle obtained using the HIO2 API. Hio2If hides the callback function specified for the HIO2 API as well as the handling required to form a connection and perform communication between the Wii console and the PC.

Hio2If notifies the application with an event if there is any change in the status of either of the connection partners. An event callback is specified when the application receives an Hio2If event.

2 Communication Method

Hio2lf uses mailboxes as the communication method to send notifications to connection partners.

2.1 Protocol Commands

Table 2-1 lists the protocol commands that manage communication between the Wii console and the PC.

Definition file: `$(REVOLUTION_SDK_ROOT)/build/demos/hio2demo/HioIf/include/Hio2If.h`

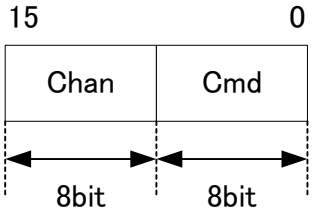
Table 2-1 Protocol Commands

Command	Description	Communication Direction
`HIO2IF_CMD_OPEN_RDONLY`	Open request (read only)	Wii console → PC
`HIO2IF_CMD_OPEN_WRONLY`	Open request (write only)	Wii console → PC
`HIO2IF_CMD_OPEN_RDWR`	Open request (read/write)	Wii console → PC
`HIO2IF_CMD_OPEN_RESULT`	Received open request	PC → Wii console
`HIO2IF_CMD_SEND`	Data send request	Wii console ↔ PC
`HIO2IF_CMD_SEND_RESULT`	Data receive end	Wii console ↔ PC
`HIO2IF_CMD_CLOSE`	Close request	Wii console ↔ PC

2.2 Protocol Command Format

Protocol commands are defined by the format shown in Figure 2-1.

Figure 2-1 Protocol Command Format



Chan — When the communication direction is from the Wii console to the PC, `chan` denotes the device type (`HIO2DeviceType`). When the communication direction is from the PC to the Wii console, `chan` denotes the pseudo-USB device number generated by Hio2lf.

Cmd — Protocol command (see Table 2-1)

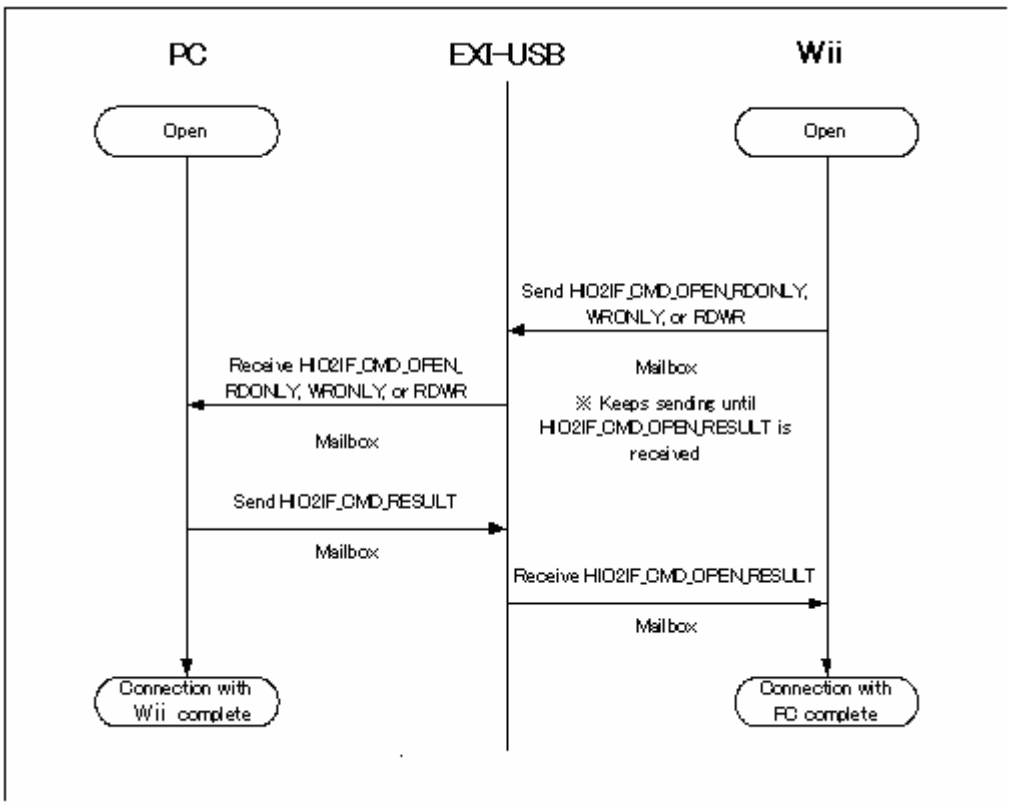
2.3 Protocols

2.3.1 Connections

The send processing on the Wii console as part of the execution of the `HIO2IF_CMD_OPEN_RDONLY`, `HIO2IF_CMD_OPEN_WRONLY`, and `HIO2IF_CMD_OPEN_RDWR` protocol commands is carried out by the functions `HIO2IFOpen` (see section 4.2.4 HIO2IFOpen) and `HIO2IFSync` (see section 4.2.11 HIO2IFSync).

Although executing the `HIO2IFOpen` function on the Wii console or the `Open` method on the PC opens an Expansion Interface (EXI) channel, data cannot be sent or received until the connection with the other communication partner is complete.

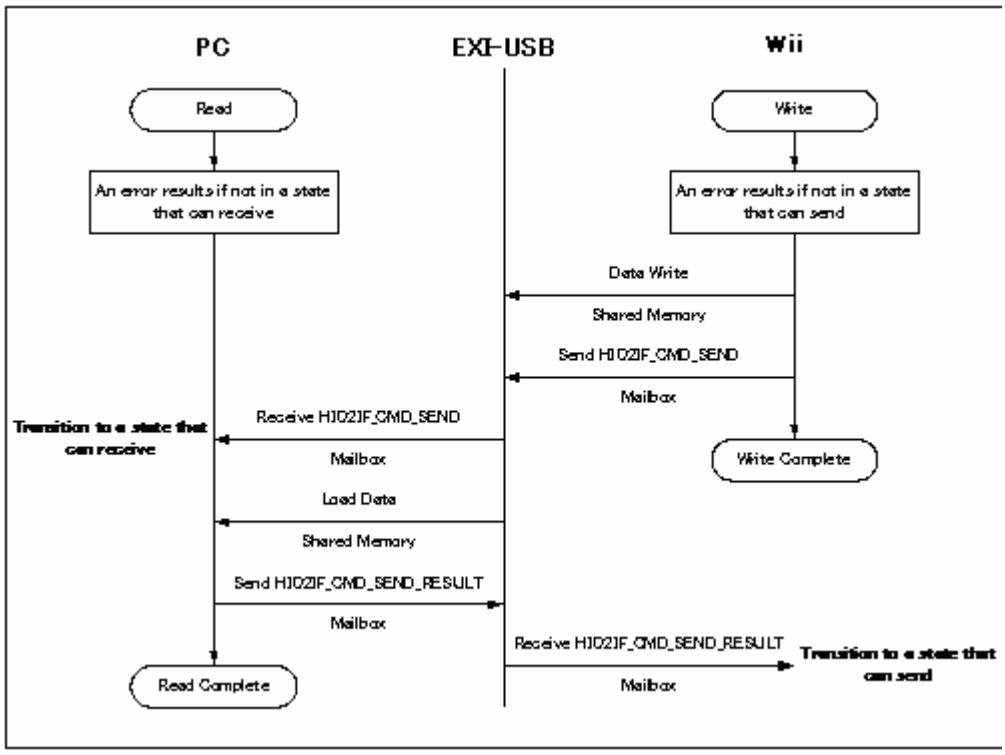
Figure 2-2 Protocol Connections



2.3.2 Sending and Receiving Data

Figure 2-3 depicts the protocol used when the Wii console sends data to the PC. The reverse flow occurs when the PC sends data to the Wii console.

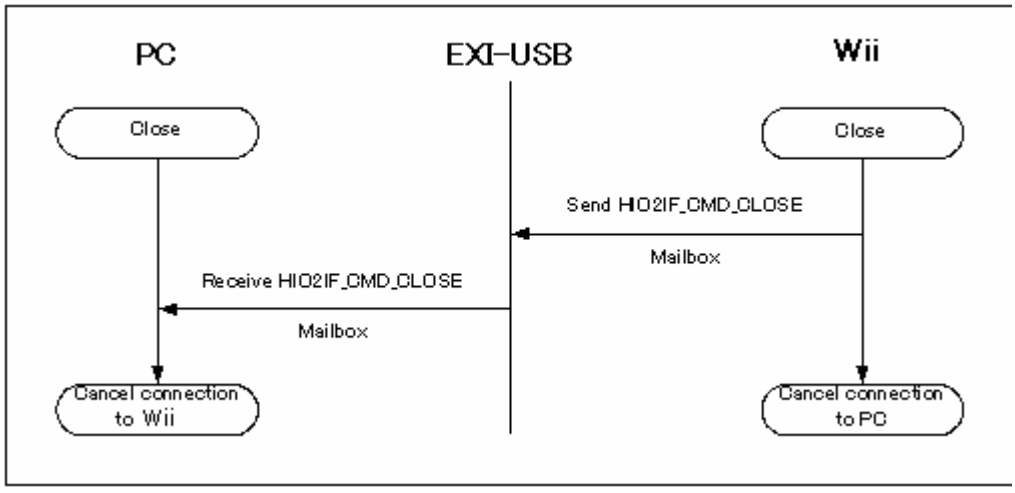
Figure 2-3 Data Send and Receive Protocol



2.3.3 Canceling Connections

Figure 2-4 shows the protocol processing when canceling connections from the Wii console. The reverse results when canceling connections from the PC.

Figure 2-4 Cancel Connection Protocol



3 Events

3.1 Event Types

Table 3-1 lists the events that can be generated by the Hio2If interface.

Definition file: `$(REVOLUTION_SDK_ROOT)/build/demos/hio2demo/HioIf/include/Hio2If.h`

Table 3-1 Event Types

Event	Description	Remarks
`HIO2IF_EVENT_CONNECT`	Completes connection	
`HIO2IF_EVENT_DISCONNECT`	Notification of disconnection from connection partner	
`HIO2IF_EVENT_RECEIVED`	Data received by connection partner	
`HIO2IF_EVENT_SEND_POSSIBLE`	Transition to data sendable status	Connected party has completed receiving data
`HIO2IF_EVENT_READ_ASYNC_DONE`	Asynchronous read complete	
`HIO2IF_EVENT_WRITE_ASYNC_DONE`	Asynchronous write complete	
`HIO2IF_EVENT_INTERRUPT`	Generates an interrupt when hot-plug occurs	Wii console only

3.2 Event Callback Function

Definition file: `$(REVOLUTION_SDK_ROOT)/build/demos/hio2demo/HioIf/include/Hio2If.h`

Syntax

```
typedef void      (* HIO2IF_EVENT_CALLBACK)(HIO2IF_ID id, HIO2IF_EVENT event);
```

Arguments

`id` ID obtained using the `HIO2IFOpen` function on the Wii console or the `Open` method on the PC.

`event` Event type (`HIO2IF_EVENT_XXXX`)

4 Hio2lf Interface for the Wii Console

This chapter lists all of the functions in the Hio2lf interface for the Wii console. All of these functions start with the prefix `HIO2IF`.

4.1 List of Functions

Declaration file: `$(ROOT)/build/demos/hiodemo/HioIf/include/Hio2If.h`

4.1.1 Functions for Accessing EXI (Expansion Interface) Channel Information

`HIO2IFGetDeviceCount`

`HIO2IFGetDevice`

4.1.2 Functions for Controlling the HIO2 API

`HIO2IFInit`

`HIO2IFOpen`

`HIO2IFRead`

`HIO2IFReadFree`

`HIO2IFWrite`

`HIO2IFWriteFree`

`HIO2IFReadStatus`

`HIO2IFClose`

`HIO2IFSync`

`HIO2IFExit`

4.1.3 Functions for Accessing States

`HIO2IFIsConnected`

`HIO2IFIsReceived`

`HIO2IFIsSendPossible`

`HIO2IFGetDeviceType`

`HIO2IFGetPcChan`

4.1.4 Functions for Getting Error Information

`HIO2IFGetLastError`

`HIO2IFGetErrorMessage`

4.2 Function Specifications

This section provides a summary specification of all of the functions in the Hio2If interface.

4.2.1 HIO2IFGetDeviceCount

Syntax

```
S32       HIO2IFGetDeviceCount( void );
```

Return Values

Number of EXI devices

Description

Gets the number of EXI devices detected by the `HIO2IFInit` function.

4.2.2 HIO2IFGetDevice

Syntax

```
HIO2DeviceType    HIO2IFGetDevice( s32 index );
```

Arguments

`index`	EXI device ID number

Return Values

EXI device type

Description

Gets the number of the EXI device type detected by the `HIO2IFInit` function.

4.2.3 HIO2IFInit

Syntax

```
HIO2IF_RESULT     HIO2IFInit( void );
```

Return Values

`HIO2IF_RESULT_SUCCESS`	Success
`HIO2IF_RESULT_FATAL`	Fatal error (error in the HIO2 API)

Description

Carries out `Hio2If` initialization.

Notes

The `HIO2IFInit` function must be called before any other functions.

Functions Used

HIO2EnumDevices, HIO2Init

4.2.4 HIO2IFOpen

Syntax

```
HIO2IF_RESULT    HIO2IFOpen( HIO2DeviceType type, HIO2IF_MODE mode,
                        HIO2IF_EVENT_CALLBACK callback, HIO2IF_ID* id );
```

Arguments

type	EXI device type	
mode	Open mode	
	HIO2IF_MODE_RDONLY	Read only
	HIO2IF_MODE_WRONLY	Write only
	HIO2IF_MODE_RDWR	Read/write
callback	Pointer to the event callback function	
id	Pointer to a buffer holding an Hio2If-specific ID	

Return Values

HIO2IF_RESULT_SUCCESS	Success
HIO2IF_RESULT_ERROR	Error
HIO2IF_RESULT_FATAL	Fatal error (error in the HIO2 API)

Description

Opens the specified EXI device type and gets its Hio2If-specific ID. An Open request is sent to the PC after the device is opened.

Notes

Be sure to specify the EXI device type obtained from the HIO2IFGetDevice function in the type argument.

Functions Used

HIO2Open, HIO2WriteMailbox

4.2.5 HIO2IFRead

Syntax

```
HIO2IF_RESULT  HIO2IFRead( HIO2IF_ID id, u32 addr, void* buffer, s32 size,
                           BOOL async );
```

Arguments

`id`	ID obtained from the `HIO2IFOpen` function
`addr`	Read address
`buffer`	Pointer to the buffer storing the data to be read
`size`	Size of the data to be read
`async`	Specifies the read method
	TRUE Asynchronous
	FALSE Synchronous

Return Values

`HIO2IF_RESULT_SUCCESS`	Success
`HIO2IF_RESULT_ERROR`	Error
`HIO2IF_RESULT_FATAL`	Fatal error (error in the HIO2 API)

Description

Reads data according to the `Hio2If` protocol. The method used to the read data is specified in the `async` argument.

Functions Used

`HIO2Read, HIO2WriteMailbox, HIO2ReadAsync`

4.2.6 HIO2IFReadFree

Syntax

```
HIO2IF_RESULT   HIO2IFReadFree( HIO2IF_ID id, u32 addr, void* buffer,
                                s32 size, BOOL async );
```

Arguments

`id`	ID obtained from the `HIO2IFOpen` function
`addr`	Read address
`buffer`	Pointer to the buffer storing the data to be read
`size`	Size of the data to be read
`async`	Specifies the read method
	`TRUE` Asynchronous
	`FALSE` Synchronous

Return Values

HIO2IF_RESULT_SUCCESS	Success
HIO2IF_RESULT_ERROR	Error
HIO2IF_RESULT_FATAL	Fatal error (error in the HIO2 API)

Description

Reads data ignoring the Hio2If protocol. The method used to read the data is specified in the async argument.

Notes

Data read using the HIO2ReadFree function must be interpreted appropriately by the application.

Functions Used

HIO2Read or HIO2ReadAsync

4.2.7 HIO2IFWrite

Syntax

```
HIO2IF_RESULT   HIO2IFWrite( HIO2IF_ID id, u32 addr, void* buffer,
                      s32 size, BOOL async );
```

Arguments

id	ID obtained from the HIO2IFOpen() function
addr	Write address
buffer	Pointer to the buffer storing the data to be written
size	Size of the data to be written
async	Specifies the write method
	TRUE Asynchronous
	FALSE Synchronous

Return Values

HIO2IF_RESULT_SUCCESS	Success
HIO2IF_RESULT_ERROR	Error
HIO2IF_RESULT_FATAL	Fatal error (error in the HIO2 API)

Description

Writes data according to the Hio2If protocol. The method used to write data is specified with the async argument.

Functions Used

HIO2Write, HIO2WriteMailbox, or HIO2WriteAsync

4.2.8 HIO2IFWriteFree

Syntax

```
HIO2IF_RESULT    HIO2IFWriteFree( HIO2IF_ID id, u32 addr, void* buffer,
                       s32 size, BOOL async );
```

Arguments

`id`	ID obtained from the `HIO2IFOpen()` function
`addr`	Write address
`buffer`	Pointer to the buffer storing the data to be written
`size`	Size of the data to be written
`async`	Specifies the write method
	`TRUE` Asynchronous
	`FALSE` Synchronous

Return Values

`HIO2IF_RESULT_SUCCESS`	Success
`HIO2IF_RESULT_ERROR`	Error
`HIO2IF_RESULT_FATAL`	Fatal error (error in the HIO2 API)

Description

Writes data ignoring the `Hio2If` protocol. The method used to write data is specified with the `async` argument.

Notes

Data written using the `HIO2WriteFree` function must be interpreted appropriately by the application.

Functions Used

`HIO2Write` or `HIO2WriteAsync`

4.2.9 HIO2IFReadStatus

Syntax

```
HIO2IF_RESULT   HIO2IFReadStatus( HIO2IF_ID id, u32* status );
```

Arguments

`id`	ID obtained from the `HIO2IFOpen` function
`status`	Pointer to the buffer storing the status

Return Values

`HIO2IF_RESULT_SUCCESS`	Success
`HIO2IF_RESULT_ERROR`	Error
`HIO2IF_RESULT_FATAL`	Fatal error (error in the HIO2 API)

Description

Gets the status and stores it in the `status` argument.

Functions Used

`HIO2ReadStatus`

4.2.10 HIO2IFClose

Syntax

```
HIO2IF_RESULT   HIO2IFClose( HIO2IF_ID id );
```

Arguments

`id`	ID obtained from the `HIO2IFOpen` function

Return Values

`HIO2IF_RESULT_SUCCESS`	Success
`HIO2IF_RESULT_ERROR`	Error
`HIO2IF_RESULT_FATAL`	Fatal error (error in the HIO2 API)

Description

Sends a close request to the connection partner. The `id` is closed after the close request is sent.

Functions Used

`HIO2Close`

4.2.11 HIO2IFSync

Syntax

```
void            HIO2IFSync( void );
```

Description

An open request is sent if the EXI channel opened using the `HIO2IFOpen` function exists and there is no connection with the PC.

Notes

Be sure to call the `HI2IFSync` function in the main loop during idle time.

Functions Used

`HIO2WriteMailbox`

4.2.12 HIO2IFExit

Syntax

```
void            HIO2IFExit( void );
```

Description

Carries out `Hio2If` end processing. Calls the `HIO2IFClose` function if an open ID exists.

Functions Used

`HIO2Exit`

4.2.13 HIO2IFIsConnected

Syntax

```
BOOL            HIO2IFIsConnected( HIO2IF_ID id );
```

Arguments

`id`	ID obtained from the `HIO2IFOpen` function

Return Values

`TRUE`	Connected
`FALSE`	Not connected

Description

Checks the status of the connection.

4.2.14 HIO2IFIsReceived

Syntax

```
BOOL            HIO2IFIsReceived( HIO2IF_ID id );
```

Arguments

id	ID obtained from the `HIO2IFOpen` function

Return Values

`TRUE`	Received data present
`FALSE`	No received data present

Description

Checks the data received status from the connection partner.

4.2.15 HIO2IFIsSendPossible

Syntax

```
BOOL            HIO2IFIsSendPossible( HIO2IF_ID id );
```

Arguments

`id`	ID obtained from the `HIO2IFOpen` function

Return Values

`TRUE`	Sendable
`FALSE`	Not sendable

Description

Checks the send status of the connection partner.

4.2.16 HIO2IFGetDeviceType

Syntax

```
HIO2DeviceType  GetDeviceType( HIO2IF_ID id );
```

Arguments

`id`	ID obtained from the `HIO2IFOpen` function

Return Values

EXI device type	Success
`HIO2_DEVICE_INVALID`	Error

Description

Gets the EXI device type assigned to the `id` argument.

4.2.17 HIO2IFGetPcChan

Syntax

```
s32             GetPcChan( HIO2IF_ID id );
```

Arguments

`id`	ID obtained from the `HIO2IFOpen` function

Return Values

Pseudo-USB device number	Success
`-1`	Error

Description

Gets the pseudo-USB device number of the PC connected with the ID given in the `id` argument.

4.2.18 HIO2IFGetLastError

Syntax

```
HIO2IF_ERROR    HIO2IFGetLastError( void );
```

Return Values

`HIO2IF_ERROR`	Error number

Description

Gets the last error that occurred during `Hio2If` processing.

4.2.19 HIO2IFGetErrorMessage

Syntax

```
const char*     HIO2IFGetErrorMessage( void );
```

Return Values

Pointer to the error message text

Description

Gets the error message corresponding to `HIO2IFGetLastError`.

5 Hio2If for the PC

`Hio2If` for the PC uses the C++ class `CHio2If`.

5.1 Class Types

Definition files: `$(ROOT)/build/demos/hiodemo/vc++/HioIf/include/Hio2IfHost.h`
`$(ROOT)/build/demos/hiodemo/vc++/HioIf/src/Hio2IfHost.cpp`

Code 5-1 Class Definition of CHio2If

```
// Host I/O interface for PC
class CHio2If
{
public:
  // Methods for accessing device path information
  int           GetDeviceCount();
  HIO2DevicePath       GetDevicePath(int nIndex);

  // Methods for controlling HIO2 APIs
  HIO2IF_RESULT Init();
  HIO2IF_RESULT Open(HIO2DevicePath pathName, HIO2IF_EVENT_CALLBACK
callback, HIO2IF_ID& id);
  HIO2IF_RESULT Read(HIO2IF_ID id, u32 addr, void* buffer, s32 size, BOOL
async);
  HIO2IF_RESULT ReadFree(HIO2IF_ID id, u32 addr, void* buffer, s32 size,
BOOL async);
  HIO2IF_RESULT Write(HIO2IF_ID id, u32 addr, void* buffer, s32 size, BOOL
async);
  HIO2IF_RESULT WriteFree(HIO2IF_ID id, u32 addr, void* buffer, s32 size,
BOOL async);
  HIO2IF_RESULT ReadStatus(HIO2IF_ID id, u32* status);
  HIO2IF_RESULT Close(HIO2IF_ID id);
  void           Exit();

  // Methods for accessing states
  BOOL          IsConnected(HIO2IF_ID id);
  BOOL          IsReceived(HIO2IF_ID id);
  BOOL          IsSendPossible(HIO2IF_ID id);
  HIO2IF_MODE   GetOpenMode(HIO2IF_ID id);
  int           GetPcChan(HIO2IF_ID id);
  HIO2DeviceType       GetDeviceType(HIO2IF_ID id);
```

```
  // Methods for multi-threaded applications
  void          EnterCriticalSection();
  void          LeaveCriticalSection();

  // Methods for getting error information
  HIO2IF_ERROR  GetLastError();
  LPCSTR        GetMessage();

  //!!!!!!!!!! Methods that must not be used with applications !!!!!!!!!!
  Omitted
};
```

5.2 Managing USB Devices for EXI-USB

When using `Hio2If` for the Wii console, a pseudo-USB device number is created from the device path in order to manage an EXI-USB (Expansion Interface-Universal Serial Bus) device. For details, see the `Open` method defined in `Hio2IfHost.cpp`.

5.3 Control Items

Methods defined in `Hio2IfHost.h` after the comment "Methods that must not be used with applications" are used internally by `Hio2If`. The reason that these methods are defined as public methods is just so that internal management data used inside the HIO2 API callback function can be set.

Note: If these methods are used by applications, `Hio2If` will malfunction. Their use is prohibited.

5.4 Interface Specifications

This section provides a summary description of all of the interfaces in the `CHio2If` class.

5.4.1 GetDeviceCount

Syntax

```
int             GetDeviceCount();
```

Return Values

Number of EXI-USB devices

Description

Gets the number of EXI-USB devices detected during execution of the `Init` method.

5.4.2 GetDevicePath

Syntax

```
HIO2DevicePath  GetDevicePath(int nIndex);
```

Arguments

`nIndex`	Number of the EXI-USB device detected

Return Values

Device path

Description

Gets the device path of the EXI-USB device detected during execution of the `Init` method.

5.4.3 Init

Syntax

```
HIO2IF_RESULT   Init();
```

Return Values

`HIO2IF_RESULT_SUCCESS`	Success
`HIO2IF_RESULT_FATAL`	Fatal error (error in the HIO2 API)

Description

Initializes `Hio2If`. The Init method is used to manage the device paths for all detected EXI-USB devices. This information is managed as internal variable data since there are cases where more than one EXI-USB device is connected to the PC.

Notes

The `Init` method must be called before any other method.

Functions Used

`HIO2EnumDevices, HIO2Init`

5.4.4 Open

Syntax

```
HIO2IF_RESULT   Open(int pathName, HIO2IF_EVENT_CALLBACK callback,
                     HIO2IF_ID& id);
```

Arguments

`pathName`	Device path of the EXI-USB device
`callback`	Pointer to the event callback function
`id`	Buffer for setting the interface-specific ID

Return Values

HIO2IF_RESULT_SUCCESS	Success
HIO2IF_RESULT_ERROR	Error
HIO2IF_RESULT_FATAL	Fatal error (error in the HIO2 API)

Description

Opens the specified device path and gets the interface-specific ID. Note that an Open mode (see Table 5-1) cannot be specified. Instead, the Open mode is automatically set by detecting the Open mode of the Wii console when a connection with the Wii console has been established.

Table 5-1 Open Modes on the Wii Console and PC

Wii Console	PC
HIO2IF_MODE_RDONLY	HIO2IF_MODE_WRONLY
HIO2IF_MODE_WRONLY	HIO2IF_MODE_RDONLY
HIO2IF_MODE_RDWR	HIO2IF_MODE_RDWR

Notes

Be sure to specify the device path obtained with the `GetDevicePath` method in the `pathname` argument.

Functions Used

`HIO2Open`

5.4.5 Read

Syntax

```
HIO2IF_RESULT    Read(HIO2IF_ID id, u32 addr, void* buffer, s32 size,
                      BOOL async);
```

Description

This method has the same functionality as the `HIO2IFRead` function on the Wii console. For details, see section 4.2.5 HIO2IFRead.

5.4.6 ReadFree

Syntax

```
HIO2IF_RESULT    ReadFree(HIO2IF_ID id, u32 addr, void* buffer, s32 size,
                 BOOL async);
```

Description

This method has the same functionality as the `HIO2IFReadFree` function on the Wii console. For details, see section 4.2.6 HIO2IFReadFree.

5.4.7 Write

Syntax

```
HIO2IF_RESULT    Write(HIO2IF_ID id, u32 addr, void* buffer, s32 size,
                       BOOL async);
```

Description

This method has the same functionality as the `HIO2IFWrite` function on the Wii console. For details, see section 4.2.7 HIO2IFWrite.

5.4.8 WriteFree

Syntax

```
HIO2IF_RESULT    WriteFree(HIO2IF_ID id, u32 addr, void* buffer, s32 size,
                       BOOL async);
```

Description

This method has the same functionality as the `HIO2IFWriteFree` function on the Wii console. For details, see section 4.2.8 HIO2IFWriteFree.

5.4.9 ReadStatus

Syntax

```
HIO2IF_RESULT    ReadStatus(HIO2IF_ID id, u32* status);
```

Description

This method has the same functionality as the `HIO2IFReadStatus` function on the Wii console. For details, see section 4.2.9 HIO2IFReadStatus.

5.4.10 Close

Syntax

```
HIO2IF_RESULT    Close(HIO2IF_ID id);
```

Description

This method has the same functionality as the `HIO2IFClose` function on the Wii console. For details, see section 4.2.10 HIO2IFClose.

5.4.11 Exit

Syntax

```
void             Exit(void);
```

Description

This method has the same functionality as the `HIO2IFExit` function on the Wii console. For details, see section 4.2.12 HIO2IFExit.

5.4.12 IsConnected

Syntax

```
BOOL            IsConnected(HIO2IF_ID id);
```

Description

This method has the same functionality as the `HIO2IFIsConnected` function on the Wii console. For details, see section 4.2.13 HIO2IFIsConnected.

5.4.13 IsReceived

Syntax

```
BOOL            IsReceived(HIO2IF_ID id);
```

Description

This method has the same functionality as the `HIO2IFIsReceived` function on the Wii console. For details, see section 4.2.14 HIO2IFIsReceived.

5.4.14 IsSendPossible

Syntax

```
BOOL            IsSendPossible(HIO2IF_ID id);
```

Description

This method has the same functionality as the `HIO2IFIsSendPossible` function on the Wii console. For details, see section 4.2.15 HIO2IFIsSendPossible.

5.4.15 GetOpenMode

Syntax

```
HIO2IF_MODE     GetOpenMode(HIO2IF_ID id);
```

Arguments

`id`	ID obtained from the `Open` method

Return Values

`HIO2IF_MODE_RDONLY`	Read Only mode
`HIO2IF_MODE_WRONLY`	Write Only mode
`HIO2IF_MODE_RDWR`	Read/Write mode
`HIO2IF_MODE_NONE`	Not connected to the Wii console

Description

Gets the Open mode set for the ID specified in the `id` argument.

5.4.16 GetPcChan

Syntax

```
int             GetPcChan(HIO2IF_ID id);
```

Arguments

`id`	ID obtained from the `Open` method

Return Values

Number of the pseudo EXI-USB device	Success
`HIO2IF_INVALID_ID`	Error

Description

Gets the number of the pseudo EXI-USB device assigned the ID specified in the `id` argument.

5.4.17 GetDeviceType

Syntax

```
HIO2DeviceType  GetDeviceType(HIO2IF_ID id);
```

Arguments

`id`	ID obtained from the `Open` method

Return Values

EXI device type	Success
`HIO2IF_DEVICE_INVALID`	Error

Description

Gets the EXI device type of the Wii console connected using the ID specified in the `id` argument.

5.4.18 EnterCriticalSection

Syntax

```
void            EnterCriticalSection();
```

Description

See section 5.4.19 LeaveCriticalSection.

5.4.19 LeaveCriticalSection

Syntax

```
void            LeaveCriticalSection();
```

Description

The methods `EnterCriticalSection` and `LeaveCriticalSection` have been prepared for multi-threaded applications. They are used to get and release the critical section object managed by

Hio2If. These methods are used to inhibit protocol inconsistencies arising due to the use of Hio2IF by the Microsoft Windows message handler and other software while performing protocol processing inside Hio2If.

Be sure to perform processing as shown in Code 5-2 when using Hio2If within the Windows message handler and other software.

Code 5-2 Thread-Safe Invocation of Hio2If Functions or Methods

```
Type MessageHandler()
{
  // Get the ownership rights to the critical section object for Hio2If
        pHIO2IF->EnterCriticalSection();

        // Interface processing
        pHIO2IF->Close(id);

  // Release ownership of the critical section object for Hio2If
        pHIO2IF->LeaveCriticalSection();
}
```

5.4.20 GetLastError

Syntax

```
HIO2IF_ERROR    GetLastError();
```

Return Values

Error code

Description

Gets the last error to occur while using Hio2If. For details on error codes, see Chapter 6 Error Codes.

5.4.21 GetMessage

Syntax

```
LPCSTR          GetMessage();
```

Return Values

LPCSTR	Pointer to the error message

Description

Gets the error message corresponding to GetLastError.

6 Error Codes

Table 6-1 lists the error codes that can be returned by `Hio2If`.

Definition file: `$(REVOLUTION_SDK_ROOT)/build/demos/hio2demo/HioIf/include/Hio2If.h`

Table 6-1 Error Codes

Error Code	Description	Remarks
`HIO2IF_ERROR_NONE`	No error	
`HIO2IF_ERROR_CHAN_NOT_FIND`	Specified EXI channel not detected	Wii console only
`HIO2IF_ERROR_CHAN_ALREADY_OPENED`	Specified EXI channel or USB device is open	
`HIO2IF_ERROR_NOT_CONNECT`	Not connected	
`HIO2IF_ERROR_WRITE_ONLY`	Open as write-only	Cannot be read
`HIO2IF_ERROR_READ_ONLY`	Open as read-only	Cannot be written
`HIO2IF_ERROR_NOT_RECV_DATA`	Data not received from connection partner	
`HIO2IF_ERROR_CANNOT_SEND_DATA`	Data cannot be sent	Connection partner is receiving data
`HIO2IF_ERROR_BUSY`	Processing impossible because data is being sent or received	
`HIO2IF_ERROR_INVALID_ID`	Invalid id specified	
`HIO2IF_FATAL_ENUMDEVICES`	Error occurred while enumerating Host I/O API devices	
`HIO2IF_FATAL_INIT`	Error occurred during Host I/O API initialization	
`HIO2IF_FATAL_OPEN`	Error occurred while opening Host I/O API	
`HIO2IF_FATAL_CLOSE`	Error occurred while closing Host I/O API	
`HIO2IF_FATAL_READ`	Error occurred while reading Host I/O API	
`HIO2IF_FATAL_WRITE`	Error occurred while writing Host I/O API	
`HIO2IF_FATAL_READSTATUS`	Error occurred while reading Host I/O API status	
`HIO2IF_FATAL_LOAD_DLL`	Failed to load DLL	PC only

If `HIO2IF_RESULT_ERROR` is returned from a function or method as the value of `HIO2IF_RESULT`, one of the non-fatal error codes in Table 6-1 (those that begin with the `HIO2IF_ERROR_` prefix) is set as the cause of the error.

If `HIO2IF_RESULT_FATAL` is returned from a function or method as the value of `HIO2IF_RESULT`, one of the fatal error codes in Table 6-1 (those that begin with the `HIO2IF_FATAL_` prefix) is set as the cause of the error.