WiiConnect24 Programming Manual

Version 1.0.1

The contents in this document are highly confidential and should be handled accordingly.

© 2007-2008 Nintendo
CONFIDENTIAL

RVL-06-0271-001-B
Released: January 28, 2008

Confidential

These coded instructions, statements, and computer programs contain proprietary information of Nintendo of America Inc. and/or Nintendo Company Ltd. and are protected by Federal copyright law. They may not be disclosed to third parties or copied or duplicated in any form, in whole or in part, without the prior written consent of Nintendo.

Table of Contents

Code

Tables

Figures

© 2007-2008 Nintendo
CONFIDENTIAL

Revision History

Version	Revision Date	Description
1.0.1	2007/12/06	Added a note concerning specification changes to slot illumination in Wii Menu Version 3.0 (section 3.2). Added supplementary information on the existence of restrictions in the Programming Guidelines (sections 4.1.2, 4.2.2). Changed to confirm that the download box was successfully mounted (section 4.3.3). Deleted the explanation stating that still images cannot be attached to messages for the Wii Message Board (sections 5.1.1, 5.3.2). Added an explanation of how to delay displaying to the Wii Message Board (section 5.1.4).
1.0.0	2007/07/24	Initial Version.

1 Introduction

This document describes the procedures used to create applications that support WiiConnect24.

1.1 Requirements

The following SDKs are required to develop applications that support WiiConnect24.

- Revolution SDK

  Library that includes the basic features.

- Revolution SDK Extensions

  Extended library that includes communication features, abbreviated as RevoEX. The Revolution SDK is required.

1.2 RevoEX (Revolution SDK Extensions)

With RevoEX, wireless/wired network features using the Internet, features for data communication with the Nintendo DS, and similar features can be used.

1.2.1 Preparing the Development Environment

Refer to the "Quick Start" section in the RevoEX README to prepare the development environment.

Also, prepare the required SDK version and development equipment to apply the most recent patch, and update the NDEV firmware as well.

1.2.2 Using the Wii Menu

After preparing the development environment, it is not required that you also use the Wii Menu. However, in the following cases, version 2.0 or later of the system menu must be installed.

- Debugging after entering network connection settings similar to a production environment
- Registering and using data that has a Mii attached in the address book
- Performing display check to the Wii Message Board

Note that the Wii Menu update feature cannot be used in the development environment.

© 2007-2008 Nintendo
CONFIDENTIAL

1.2.3 Configuration Procedures to use the Communication Features

To use communication features with Wii, the four-step procedure below must be followed.

1. Network connection settings
2. Network connection test
3. Agreement to the User's License (EULA)
4. WiiConnect24 related settings

See the *Network Development Environment* document included in the RevoEX package for details on configuration methods and tools for configuration.

1.2.4 Support Codes and Errors

After completing the procedures above, a five-digit support code can be seen, but this code indicates the characteristics of the communication environment and is not necessarily an indication of an error. You can ignore it.

See the *Network Development Environment Document* for details on the error correspondence table or error generation methods. Although descriptions of Wii network related errors and WiiConnect24 errors are provided, see *Nintendo Wi-Fi Connection Error Simulation Manual* for Nintendo Wi-Fi Connection errors.

1.2.5 Cautions for Configuration Items that Limit Use of Communication Features

Note that even if an environment using the communication features is arranged and the settings for network communications have been made appropriately, there are still some configuration items that can limit the communication features.

- EULA (User's Agreement)

  If not agreed to, some communication features cannot be used.

- WiiConnect24

  If OFF, WiiConnect24 cannot be used.

- Parental Control

  If "Use" is selected, some communication features cannot be used.

  In WiiConnect24 the restriction can be turned ON or OFF separately, when parental control is used.

2 NWC24 Library

The NWC24 library provides an environment to send and receive messages and perform downloads using the WiiConnect24 features.

2.1 Opening and Closing the NWC24 Library

To use the APIs for sending messages, managing download tasks, or accessing the Wii console's friend roster, the NWC24 library must be opened.

If some of the APIs are used when the library is not opened, an error is generated. While the library is opened, the automatic send/receive feature is temporarily stopped to maintain consistency. Because the sending and receiving of messages cannot be performed if the library is left open, be sure to close the library after using the message API or when closing the application.

The following APIs are used to open or close the NWC24 library.

Code 2-1 APIs to Open/Close the NWC24 Library

```
NWC24Err NWC24OpenLib( void* workMemory );
NWC24Err NWC24CloseLib( void );
```

The following is some NWC24 library sample code.

Code 2-2 Example for Opening/Closing the NWC24 Library

```
/*---------------------------------------------------------------------------*/
MEMHeapHandle   HeapHndl;
MEMAllocator    Allocator;

/*---------------------------------------------------------------------------*
   Main
 *---------------------------------------------------------------------------*/
int main(void)
{
   NWC24Err   err;
   s32        result;
   void*      arenaLo;
   void*      arenaHi;
   char*      libWorkMem;

   // Memory allocator initialization.
   arenaLo = OSGetMEM1ArenaLo();
   arenaHi = OSGetMEM1ArenaHi();
   HeapHndl = MEMCreateExpHeap(arenaLo, (u32)arenaHi - (u32)arenaLo);
   OSSetMEM1ArenaLo(arenaHi);
```

```
    MEMInitAllocatorForExpHeap(&Allocator, HeapHndl, 32);

    // NAND Library initialization.
    result = NANDInit();
    if ( result != NAND_RESULT_OK ) {
        OSHalt("NANDInit() failed.¥n");
    }

    // VF Library initialization.
    VFInit();

    // Allocate work memory.
    libWorkMem = MEMAllocFromAllocator(&Allocator, NWC24_WORK_MEM_SIZE);

    // Open the NWC24 library.
    err = NWC24OpenLib(libWorkMem);
    if ( err != NWC24_OK ) {
        OSReport("NWC24OpenLib(): Error %d¥n", err);
        OSHalt("Failed.¥n");
    }

    ...

    // Close the NWC24 library.
    err = NWC24CloseLib();
    if ( err != NWC24_OK ) {
        OSReport("NWC24CloseLib(): Error %d¥n", err);
        OSHalt("Failed.¥n");
    }

    // Release work memory.
    MEMFreeToAllocator(&Allocator, libWorkMem);

    OSHalt("¥nCompleted.¥n");
    return 0;
}
```

2.1.1 Opening the NWC24 Library

The NWC24 library requires memory to perform operations internally, so when opening the library, pass a pointer to a memory region in an argument. The required size for this memory region is defined with the `NWC24_WORK_MEM_SIZE` macro. The start address must be 32-byte aligned. The VF library must also be initialized.

When the return value of the `NWC24OpenLib` function is an error, as indicated in Table 2-1, the error message specified in the guidelines is displayed. Do not use WiiConnect24 features after this error message is displayed.

Even if the return value is an error, if the error is `NWC24_ERR_BUSY`, `NWC24_ERR_INPROGRESS`, or `NWC24_ERR_MUTEX`, the library is only off-limits temporarily, and normally the library can be opened without error by calling it again after some time passes. It is recommended that retries are spaced 1–2 seconds apart and are performed for about 10–15 seconds.

Table 2-1 Error Codes that Prohibit Subsequent Use of WiiConnect24

Error Type	Error Code (Return Value)
File Corrupted	`NWC24_ERR_FILE_OPEN`
	`NWC24_ERR_FILE_CLOSE`
	`NWC24_ERR_FILE_READ`
	`NWC24_ERR_FILE_WRITE`
	`NWC24_ERR_FILE_NOEXISTS`
	`NWC24_ERR_FILE_BROKEN`
	`NWC24_ERR_FILE_OTHER`
	`NWC24_ERR_BROKEN`
	`NWC24_ERR_NAND_CORRUPT`
	`NWC24_ERR_INTERNAL_VF`
System menu must be updated	`NWC24_ERR_OLD_SYSTEM`
Other fatal errors	`NWC24_ERR_FATAL`
	`NWC24_ERR_INTERNAL_IPC`

The restoration operation for corrupted files is performed with the system menu, so the application does not need to perform restoration processes.

2.1.2 Closing the NWC24 Library

After closing the NWC24 library, the memory region passed while opening can be released. The automatic send/receive feature, which was stopped while the library was in use, becomes operational.

3 Messages

The NWC24 message API provides an environment where messages that are exchanged between one Wii and another Wii or an external e-mail address are created, viewed, and managed.

3.1 Initializing the Message API

The NWC24 library must be opened to use the NWC24 message API. For help when opening and closing the NWC24 library, see section 2.1 Opening and Closing the NWC24 Library.

3.2 Creating Messages

The sequence for creating messages is described using an example from NWC24 library sample code, as shown in Code 3-1.

Code 3-1 Example for Creating Messages

```
/*---------------------------------------------------------------------------*
   Data for this test.
 *---------------------------------------------------------------------------*/
static char* TestSubject = "Test Message";
static char* TestMsgText =
   "Hello WiiConnect24 World!!¥x0d¥x0a"
   "This is a test mail.¥x0d¥x0a"
   "Thank you.¥x0d¥x0a";
static NWC24UserId TestIdTo = (u64)12345678;

/*---------------------------------------------------------------------------*
   Post a test message into the send box.
 *---------------------------------------------------------------------------*/
void PostTestMsg( void )
{
   NWC24Err      err;
   NWC24MsgObj   msgObj;

   // Initialize the message object as a message from a Wii to another Wii.
   err = NWC24InitMsgObj(&msgObj, NWC24_MSGTYPE_WII_APP);
   if ( err != NWC24_OK ) {
      OSReport("NWC24InitMsgObj(): error %d¥n", err);
      return;
   }
   // Destination (user ID specification)
   err = NWC24SetMsgToId(&msgObj, TestIdTo);
```

```
    if ( err != NWC24_OK ) {
        OSReport("NWC24SetMsgToId(): error %d¥n", err);
        return;
    }
    // Subject line
    err = NWC24SetMsgSubject(&msgObj, TestSubject, (u32)strlen(TestSubject));
    if ( err != NWC24_OK ) {
        OSReport("NWC24SetMsgSubject(): error %d¥n", err);
        return;
    }
    // Message text
    err = NWC24SetMsgText(&msgObj, TestMsgText, (u32)strlen(TestMsgText),
                      NWC24_US_ASCII, NWC24_ENC_7BIT);
    if ( err != NWC24_OK ) {
        OSReport("NWC24SetMsgText(): error %d¥n", err);
        return;
    }
    // Finalize message settings, and post message to the send box.
    err = NWC24CommitMsg(&msgObj);
    if ( err != NWC24_OK ) {
        OSReport("NWC24CommitMsg: error %d¥n", err);
        return;
    }
    OSReport("Posted a test message successfully.¥n");
    return;
}
```

To create a message, the prepared message object must first be initialized with the `NWC24InitMsgObj` function. At the time the message object is initialized, the type of message to create must be decided. In the code example above, `NWC24_MSGTYPE_WII_APP` is specified and a message from one Wii to another Wii is created. See the `NWC24` library function reference for the other types of messages.

To set an addressee in the created message object, use either the `NWC24SetMsgToId` or `NWC24SetMsgToAddr` function. In the code example above, the `NWC24SetMsgToId` function is used, and the Wii having Wii number 12345678 is specified as the addressee. The `NWC24SetMsgToAddr` message is used to specify a general e-mail address as the addressee, but `NWC24_MSGTYPE_PUBLIC` must be specified when the message object is initialized. Furthermore, the buffer used to specify the e-mail address must not be released until the message transmission completes.

To set a message subject line when creating a message to a Wii, use the `NWC24SetMsgSubject` function. The string passed to the subject must be characters that can be displayed in 7 bits (ASCII, ISO-2022-JP, or similar). Furthermore, the subject is not displayed when displaying on the Wii Message Board. For messages to general e-mail addresses, use the `NWC24SetMsgSubjectPublic` function instead because character code conversion and MIME encoding for the mail header is required. The string passed to the subject must be UTF-16BE, which is used for internal encoding. For either function, the buffer used to specify the subject must not be released until the message transmission completes.

To set the body text of the message when creating a message to Wii, use the `NWC24SetMsgText` function. The character set and MIME encoding method must be selected to match the body text character code. In Code 3-1, because the entire body text is input as ASCII characters that can be displayed in 7 bits, `NWC24_US_ASCII` is specified in the character set and `NWC24_ENC_7BIT` is specified in the MIME encoding method. For messages to general e-mail addresses, the `NWC24SetMsgTextPublic` function is used. The character set and MIME encoding method are not specified and the body text string is passed as UTF-16BE, with the hint setting specified for automatic encoding detection, along with substitute characters for characters that could not be encoded. With general mail clients, if a message is sent with different character encoding for the subject name and body, characters may display corrupted. To prevent this occurrence, use the `NWC24SetMsgSubjectAndTextPublic` function, which sets the subject name and body at the same time. Regardless of which function is used, the buffer and work area used must not be released until the message transmission completes.

Finally, the message creation is completed by writing the message to the send box with the `NWC24CommitMsg` function. However, the single exception is messages that have one's own Wii number set as the addressee. These are written to the receive box and not the send box after transmission completes. Buffers and so forth, that were used to create messages, can be released after this function completes.

The following functions, which were not used in Code 3-1, are also available.

With the `NWC24SetMsgAttached` function, up to two binary data files can be attached to a message. See Table 3-1 for types of data that can be attached.

Table 3-1 Types of Binary Data that can be Attached to Messages

Sender		Wii		PC (Reference)
Destination		Wii	PC	Wii
Data Type	Text Data	O	O	O (Note 1)
	Static Image Data	O	X	O (Note 1)
	Data dependent on an application	O	X	X
	Letterhead Data	O	X	X
	Mii Data	O	X	X

Note 1: A total of 400 kilobytes can be sent from a PC to a Wii. In addition, only one JPEG-formatted image file can be sent.

By adding a tag number to a message with the NWC24SetMsgTag function, the message that matches the sender's address, application ID, and tag number can be updated by being overwritten. This can be used to leave only one copy of a message in the receive box when that message is frequently updated and contains information that is only meaningful when it is the most recent (such as high scores).

Using the NWC24SetMsgLedPattern function, the slot illumination of the Wii console can be used in a specified pattern when a message arrives. The slot illumination is only lit for messages to the Wii Message Board. Beginning with Wii Menu 3, the slot can be also illuminated when an application sends a message directly to the Wii Message Board. However, the slot can only be illuminated when the message was sent via the network with Wii Menu 2. Also, if the Wii console settings or application restrict slot illumination, it will not illuminate even if a message is received by the Wii Message Board.

3.3 Obtaining the Message List

The sequence for obtaining the message list is described using an example from NWC24 library sample code (Code 3-2).

Code 3-2 Example for Obtaining Message Lists

```
/*---------------------------------------------------------------------------*
   Listing
 *---------------------------------------------------------------------------*/
void ListMessageBox( NWC24MsgBoxId mBoxId )
{
   u32          numMsgs;
   u32          bufSize;
   u32*         idListBuf;
   NWC24Err     err;
   Int          i;

   // Obtains the number of messages in the specified message box.
   err = NWC24GetNumMsgs(mBoxId, &numMsgs);
   if ( err != NWC24_OK ) {
      OSReport("NWC24GetNumMsgs(): error %d¥n", err);
      return;
   }
   if ( numMsgs == 0 ) {
      OSReport("(No message.)¥n");
      return;
   }

   // Allocates memory to store the message ID list.
   bufSize = OSRoundUp32B(numMsgs * sizeof(u32));
```

```
    idListBuf = (u32*)MEMAllocFromAllocator(&Allocator, bufSize);

    // Obtains the message ID list.
    err = NWC24GetMsgIdList(mBoxId, idListBuf, bufSize);
    if ( err != NWC24_OK ) {
        OSReport("NWC24GetNumMsgList(): error %d¥n", err);
        return;
    }
    for ( i = 0 ; i < numMsgs ; ++i ) {
        NWC24MsgObj    msgObj;
        NWC24MsgType   type;

        // Obtains the message objects.
        err = NWC24GetMsgObj(&msgObj, mBoxId, idListBuf[i]);
        if ( err != NWC24_OK ) {
            OSReport("NWC24GetMsgObj(): error %d¥n", err);
            return;
        }
        ...
    }
    MEMFreeToAllocator(&Allocator, idListBuf);
    return;
}
```

To check the number of messages stored in the send or receive box, use the `NWC24GetNumMsgs` function.

Obtain the message ID list with the `NWC24GetMsgIdList` function. When preparing the array to store the message ID list to pass in an argument, the number of messages (as obtained earlier) may be used.

To obtain the message objects, specify the message box type and the message ID and call the `NWC24GetMsgObj` function. The following section describes the method for viewing message contents.

3.4 Viewing Messages

To view messages, first message objects must be obtained, either by getting the message list as described above or searching for messages. For more information see section 3.6 Searching for Messages. This section describes how to obtain information stored in a message, such as its sender, using an example from NWC24 library sample code.

3.4.1 Obtaining the Sender

To obtain the sender information from the message object, use the `NWC24GetMsgFromId` function for messages between Wii consoles and the `NWC24ReadMsgFromAddr` function for messages from a general e-mail address.

Code 3-3 Example of Obtaining the Sender

```
/*---------------------------------------------------------------------------*
   Views 'from' address.
 *---------------------------------------------------------------------------*/
void ViewFrom( NWC24MsgObj* msgObj )
{
   NWC24Err      err;
   NWC24MsgType  type;

   // Obtains the message type.
   err = NWC24GetMsgType(msgObj, &type);
   if ( err != NWC24_OK ) {
      OSReport("NWC24GetMsgType(): error %d¥n", err);
      return;
   }
   switch (type) {
     // Message type used for exchange between Wii consoles
     case NWC24_MSGTYPE_WII_MENU_SHARED:
     case NWC24_MSGTYPE_WII_APP:
     case NWC24_MSGTYPE_WII_MENU:
     case NWC24_MSGTYPE_WII_APP_HIDDEN:
       {
          NWC24UserId    uid;

          // Obtains the sender by the user ID.
          err = NWC24GetMsgFromId(msgObj, &uid);
          if ( err != NWC24_OK ) {
             OSReport("NWC24GetMsgFromId(): error %d¥n", err);
             return;
          }
          ...
       }
       break;
     // Message type from a device other than a Wii
```

```
     case NWC24_MSGTYPE_PUBLIC:
       {
          char   addrStr[NWC24MSG_MAX_ADDRSTR];

          // Obtains the sender by the e-mail address.
          err = NWC24ReadMsgFromAddr(msgObj, addrStr, NWC24MSG_MAX_ADDRSTR);
          if ( err != NWC24_OK ) {
             OSReport("NWC24ReadMsgFromAddr(): error %d¥n", err);
             return;
          }
          ...
       }
       break;
     default:
       break;
   }
   return;
}
```

3.4.2 Obtaining the Recipient

To obtain the recipient information from the message object, first obtain the number of registered recipients with the `NWC24GetMsgNumTo` function. Then use `NWC24ReadMsgToId` for messages between Wii consoles or `NWC24ReadMsgToAddr` for messages from a general e-mail address.

Code 3-4 Example of Obtaining the Recipients

```
/*------------------------------------------------------------------------*
   Views 'to' addresses.
 *------------------------------------------------------------------------*/
void ViewTo( NWC24MsgObj* msgObj )
{
   NWC24Err      err;
   NWC24MsgType  type;
   u32           numTo;
   u32           i;

   // Obtains the message type.
   err = NWC24GetMsgType(msgObj, &type);
   if ( err != NWC24_OK ) {
      OSReport("NWC24GetMsgType(): error %d¥n", err);
      return;
   }

   // Obtains the number of recipients registered.
   err = NWC24GetMsgNumTo(msgObj, &numTo);
   if ( err != NWC24_OK ) {
      OSReport("NWC24GetMsgNumTo(): error %d¥n", err);
      return;
   }
```

```
   switch (type) {
     // Message type used for exchange between Wii consoles
     case NWC24_MSGTYPE_WII_MENU_SHARED:
     case NWC24_MSGTYPE_WII_APP:
     case NWC24_MSGTYPE_WII_MENU:
     case NWC24_MSGTYPE_WII_APP_HIDDEN:
       {
          NWC24UserId    uid;

          for ( i = 0 ; i < numTo ; ++i ) {
             // Obtains the recipients by the user IDs.
             err = NWC24ReadMsgToId(msgObj, i, &uid);
             if ( err != NWC24_OK ) {
                OSReport("NWC24ReadMsgToId(): error %d¥n", err);
                return;
             }
             ...
          }
       }
       break;
     // Message type from a device other than a Wii
     case NWC24_MSGTYPE_PUBLIC:
       {
          char    addrStr[NWC24MSG_MAX_ADDRSTR];

          for ( i = 0 ; i < numTo ; ++i ) {
             // Obtains the recipients by the e-mail addresses.
             err = NWC24ReadMsgToAddr(
                    msgObj, i, addrStr, NWC24MSG_MAX_ADDRSTR);
             if ( err != NWC24_OK ) {
                OSReport("NWC24ReadMsgToAddr(): error %d¥n", err);
                return;
             }
             ...
          }
       }
       break;
     default:
       break;
   }

   return;
}
```


© 2007-2008 Nintendo
CONFIDENTIAL

3.4.3 Obtaining the Subject

To obtain the subject information from the message object, first prepare a buffer to store the subject string that is the size obtained by the `NWC24GetMsgSubjectSize` function. Then, call the function to obtain the subject string with that buffer passed in an argument.

The code example (Code 3-5) uses the `NWC24ReadMsgSubject` function, which is primarily used for messages exchanged between Wii consoles. The `NWC24ReadMsgSubjectPublic` function automatically performs the internal conversion process based on the mail header and the encode information included in messages from general e-mail addresses. Ultimately, a string in UTF-16BE format, which is the character encoding for internal formatting, can be obtained. To use this function, a sufficient region must be allocated as a work area to be used for character conversion.

Code 3-5 Example of Obtaining the Subject

```
/*---------------------------------------------------------------------------*
   Views subject.
 *---------------------------------------------------------------------------*/
void ViewSubject( NWC24MsgObj* msgObj )
{
   NWC24Err   err;
   u32        bufSize;
   char*      buffer;
   u32        i;

   // Obtains the size of the buffer needed to store the subject.
   err = NWC24GetMsgSubjectSize(msgObj, &bufSize);
   if ( err != NWC24_OK ) {
      OSReport("NWC24GetMsgSubjectSize(): error %d¥n", err);
      return;
   }

   // Allocates a buffer to store the subject.
   buffer = (char*)MEMAllocFromAllocator(&Allocator, OSRoundUp32B(bufSize));

   // Obtains the subject.
   err = NWC24ReadMsgSubject(msgObj, buffer, bufSize);
   if ( err != NWC24_OK ) {
      OSReport("NWC24ReadMsgSubject(): error %d¥n", err);
      MEMFreeToAllocator(&Allocator, buffer);
      return;
   }

   ...

   // Release the buffer storing the subject.
   MEMFreeToAllocator(&Allocator, buffer);
   return;
}
```

3.4.4 Obtaining the Creation Time and Date

To obtain the send date and time information from the message object, use the `NWC24GetMsgDate` function. The information on the date and time when the message was created is stored in the `OSCalendarTime` structure passed in an argument.

Code 3-6 Example of Obtaining the Creation Date and Time

```
/*---------------------------------------------------------------------------*
   Views date.
 *---------------------------------------------------------------------------*/
static const char* MonStr[12] =
{
   "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
};

static const char* WdayStr[7] =
{
   "Sun", "Mon", "Tue", "Wed", "Thr", "Fri", "Sat"
};

void ViewDate( NWC24MsgObj* msgObj )
{
   NWC24Err       err;
   OSCalendarTime  cTime;

   // Obtains the message creation date and time.
   err = NWC24GetMsgDate(msgObj, &cTime);
   if ( err != NWC24_OK ) {
      OSReport("NWC24GetMsgDate(): error %d¥n", err);
      return;
   }
   OSReport("%s, %d %s %d ",
         WdayStr[cTime.wday], cTime.mday, MonStr[cTime.mon], cTime.year);
   OSReport("%02d:%02d -0000¥n", cTime.hour, cTime.min);
   return;
}
```

3.4.5 Obtaining the Body Text

To obtain the body text information from the message object, first prepare a buffer that is the size obtained by the `NWC24GetMsgTextSize` function to store the body text string. Then, call the function to obtain the body text with that buffer passed in an argument.

The code example (Code 3-7) uses the `NWC24ReadMsgText` function, which is primarily used for messages exchanged between Wii consoles. With the `NWC24ReadMsgTextEx` function, which performs the same operation, the character code information can be obtained in a string.

The internal conversion process based on the encode information included in messages from general e-mail addresses is performed automatically with the `NWC24ReadMsgTextPublic` function, and ultimately a string in UTF-16BE format (which is the character encoding for internal formatting) can be obtained. To use this function, a sufficient region must be allocated as a work area to be used for character conversion.

Code 3-7 Example of Obtaining the Body Text

```
/*---------------------------------------------------------------------------*
   Views body text.
 *---------------------------------------------------------------------------*/
void ViewBodyText( NWC24MsgObj* msgObj )
{
   NWC24Err      err;
   u32           bufSize;
   char*         buffer;
   u32           i;
   NWC24Charset  charset;
   NWC24Encoding encoding;

   // Obtains the size of the buffer needed to store the body text.
   err = NWC24GetMsgTextSize(msgObj, &bufSize);
   if ( err != NWC24_OK ) {
      OSReport("NWC24GetMsgTextSize(): error %d¥n", err);
      return;
   }

   // Allocates a buffer to store the body text.
   buffer = (char*)MEMAllocFromAllocator(&Allocator, OSRoundUp32B(bufSize));

   // Obtains the body text.
   err = NWC24ReadMsgText(msgObj, buffer, bufSize, &charset, &encoding);
   if ( err != NWC24_OK ) {
      OSReport("NWC24ReadMsgText(): error %d¥n", err);
```

```
        MEMFreeToAllocator(&Allocator, buffer);
        return;
    }
    OSReport("--------- [Charset=%08X  Encoding=%d] -----------¥n",
            charset, encoding);
    for ( i = 0 ; i < bufSize ; ++i ) {
        if ( buffer[i] != 0x0A ) {
            OSReport("%c", buffer[i]);
        }
    }

    // Release the buffer storing the body text.
    MEMFreeToAllocator(&Allocator, buffer);
    return;
}
```

3.4.6 Obtaining Attached Binary Data

To obtain the number of binary data files attached to the message, use the `NWC24GetMsgNumAttached` function. Prepare a buffer to store the binary data based on the size of the various binary data files obtained with the `NWC24GetMsgAttachedSize` function and then obtain the binary data files with the `NWC24ReadMsgAttached` function. To obtain the type of binary data, use the `NWC24GetMsgAttachedType` function.

Code 3-8 Example of Obtaining the Attached Binary Data

```
/*---------------------------------------------------------------------------*
    Views attachment binaries.
 *---------------------------------------------------------------------------*/
void ViewAttachment( NWC24MsgObj* msgObj )
{
    NWC24Err          err;
    u32               bufSize;
    char*             buffer;
    u32               i, j;
    u32               numAttach;
    NWC24MIMEType     fileType;

    // Obtains the number of binary data files attached.
    err = NWC24GetMsgNumAttached(msgObj, &numAttach);
    if ( err != NWC24_OK ) {
        OSReport("NWC24GetMsgNumAttached(): error %d¥n", err);
        return;
    }
    for ( j = 0 ; j < numAttach ; ++j ) {
        // Obtains the size of the attached binary data files.
        err = NWC24GetMsgAttachedSize(msgObj, j, &bufSize);
```

```
      if ( err != NWC24_OK ) {
         OSReport("NWC24GetMsgAttachedSize(): error %d¥n", err);
         return;
      }

      // Allocates a buffer to store the attached binary data.
      buffer = (char*)MEMAllocFromAllocator(
            &Allocator, OSRoundUp32B(bufSize));

      // Obtains the attached binary data.
      err = NWC24ReadMsgAttached(msgObj, j, buffer, bufSize);
      if ( err != NWC24_OK ) {
         OSReport("bufsize = %d¥n", bufSize);
         OSReport("NWC24ReadMsgAttached(): error %d¥n", err);
         MEMFreeToAllocator(&Allocator, buffer);
         return;
      }

      // Obtains the type of binary data attached.
      err = NWC24GetMsgAttachedType(msgObj, j, &fileType);
      if ( err != NWC24_OK ) {
         OSReport("NWC24GetMsgAttachedType(): error %d¥n", err);
         return;
      }
      OSReport("[Attached binary %d : Type=%08X]¥n", j, fileType);
      for ( i = 0 ; i < bufSize ; ++i ) {
         OSReport(" %02X", (u8)buffer[i]);
         if ( (i % 16) == 15 ) {
            OSReport("¥n");
         }
      }
      OSReport("¥n");

      // Releases the buffer storing the attached binary data.
      MEMFreeToAllocator(&Allocator, buffer);
   }
   return;
}
```

3.4.7 Obtaining Miscellaneous Information

The message object also includes other information, such as the application ID, group ID, and tag number. These can be obtained with the `NWC24GetMsgAppId`, `NWC24GetMsgGroupId`, and `WC24GetMsgTag` functions, respectively. See the *NWC24 API* for details about these information items.

3.5 Deleting Messages

To delete messages in the send or receive box, call the `NWC24DeleteMsg` function with the message box and the message ID of the message to delete passed in arguments. As a rule, only the application that created the message can delete it.

Use the function call in Code 3-9 to delete a message.

Code 3-9 API to Delete Messages

```
NWC24Err NWC24DeleteMsg( NWC24MsgBoxId mboxId, u32 msgId );
```

3.6 Searching for Messages

The NWC24 library provides a search feature where simple limiting conditions can be specified.

The following conditions can be specified as search conditions.

- Message Box Type (Send Box/Receive Box)
- Sender's Wii Number
- Application ID of the Application that Created the Message
- Whether Displayed in the Wii Message Board

The functions that are used are the `NWC24SetSearchCondMsgBox`, `NWC24SetSearchCondFromAddrId`, `NWC24SetSearchCondAppId`, and `NWC24SetSearchCondForMenu` functions.

When multiple conditions are specified at once, they will be treated as AND conditions and serve to narrow the search. To reset a condition, first clear the search conditions by calling the `NWC24InitSearchConds` function.

After setting the conditions, call the `NWC24SearchMsgs` function to store the message objects matching the conditions in an argument array. Other arguments passed include a variable to return the number stored and a variable to return the number of messages remaining. By not clearing the search conditions and calling this function until zero messages remain, all search results can be obtained.

Code 3-10 is NWC24 library sample code. While changing the search conditions, the content of the messages obtained as search results are displayed in an array of message objects containing MSGOBJ_ARRAY_SIZE number of objects.

Code 3-10 Example of Searching Messages

```
int main( void )
{
   NWC24Err   err;
   s32        result;
   void*      arenaLo         = NULL;
   void*      arenaHi         = NULL;
   char*      libWorkMem      = NULL;
   u8         iLoop;

   // Memory allocator initialization.
   arenaLo = OSGetMEM1ArenaLo();
   arenaHi = OSGetMEM1ArenaHi();
   s_HeapHndl = MEMCreateExpHeap( arenaLo, (u32)arenaHi - (u32)arenaLo );
   OSSetMEM1ArenaLo( arenaHi );
   MEMInitAllocatorForExpHeap( &s_Allocator, s_HeapHndl, 32 );

   // NAND Library initialization.
   result = NANDInit();
   if ( result != NAND_RESULT_OK ) {
      OSHalt( "NANDInit() failed.¥n" );
   }

   // VF Library initialization.
   VFInit();

   OSReport( "*****************************************************¥n" );
   OSReport( "   MessageBox Search demo¥n" );
   OSReport( "*****************************************************¥n" );

   // Allocate work memory.
   libWorkMem = MEMAllocFromAllocator( &s_Allocator, NWC24_WORK_MEM_SIZE );

   // Open the NWC24 library.
   err = NWC24OpenLib( libWorkMem );
   if ( err != NWC24_OK ) {
      OSReport( "NWC24OpenLib(): Error %d¥n", err );
      OSHalt( "Failed.¥n" );
```

```
}

// Obtains own user ID. (used for search conditions)
err = NWC24GetMyUserId( &s_uidMy );
if ( err != NWC24_OK ) {
   OSReport( "NWC24GetMyUserId(): Error %d¥n", err );
   OSHalt( "Failed.¥n" );
}

// Obtains current application ID. (used for search conditions)
s_appId = *(u32*)OSGetAppGamename();

// Post test message.
PostTestMsg();

// Search while changing search conditions, and display results.
for ( iLoop = 0; iLoop < 5; ++iLoop ) {
   u32           iObj;
   u32           numStored;
   u32           numRemain;
   NWC24MsgObj   msgObjArray[MSGOBJ_ARRAY_SIZE];

   // Change the search condition with the number of loops.
   switch ( iLoop ) {
      case 0: break;
      case 1: s_bSCondSendBox     = TRUE; break;
      case 2: s_bSCondAppId       = TRUE; break;
      case 3: s_bSCondFromAddrId  = TRUE; break;
      case 4: s_bSCondForMenu     = TRUE; break;
   }

   // Set the search conditions.
   (void)NWC24InitSearchConds();
   if ( s_bSCondSendBox )   (void)NWC24SetSearchCondMsgBox(
                                       NWC24_SEND_BOX );
   if ( s_bSCondAppId )     (void)NWC24SetSearchCondAppId( s_appId );
   if ( s_bSCondFromAddrId ) (void)NWC24SetSearchCondFromAddrId(
                                       s_uidMy );
   if ( s_bSCondForMenu )   (void)NWC24SetSearchCondForMenu();

   // Display the search conditions.
   ViewSearchConds();
```


© 2007-2008 Nintendo
CONFIDENTIAL

```
      do
      {
         // Search for messages.
         err = NWC24SearchMsgs( msgObjArray, MSGOBJ_ARRAY_SIZE,
                                &numStored, &numRemain );
         if ( err != NWC24_OK ) {
            OSReport( "NWC24SearchMsgs(): Error %d¥n", err );
            OSHalt( "Failed.¥n" );
         }

         OSReport("=====================================================¥n");
         OSReport(" [NWC24SearchMsgs():  Stored: %d  Remain: %d]¥n",
                  numStored, numRemain);
         OSReport("=====================================================¥n");

         // Displays the message contents.
         for ( iObj = 0 ; iObj < numStored ; ++iObj ) {
            ViewMessage( &msgObjArray[iObj] );
         }
      }
      while ( numRemain > 0 );
      // Repeats until all messages meeting the search conditions are displayed.
   }

   // Close the NWC24 library.
   err = NWC24CloseLib();
   if ( err != NWC24_OK ) {
      OSReport( "NWC24CloseLib(): Error %d¥n", err );
      OSHalt( "Failed.¥n" );
   }

   // Release work memory.
   MEMFreeToAllocator( &s_Allocator, libWorkMem );

   OSHalt( "¥nCompleted.¥n" );
   return 0;
}
```

4 Downloading

The Wii has a feature to automatically download contents while an application is executing or while in standby mode and store the contents in Wii console NAND memory. Using this feature, the application can obtain additional data without making the user wait.

4.1 Outline

The *NWC24 Download API* supplies features to automatically download data (contents) from a designated location on the network (URL) without the programmer needing to write complicated network communication processing code. It can also automatically verify signatures to guarantee that the downloaded data is not falsified.

This section summarizes the terms specific to the *NWC24 Download API*. Refer to *WiiConnect24 Overview* or *WiiConnect24 Programming Guidelines* for details on the restrictions related to the download feature.

4.1.1 Download Task

The download task combines into one, all the settings required to preschedule the download, such as the interval to confirm URL or content updates.

4.1.2 Download Task List

The download task list is a list of download tasks that are currently enabled. Download tasks first become enabled when they are registered in this download task list.

A maximum of two download tasks can be registered from a single application. The total number of tasks that can be registered by a normal application is 112. When the 113th task is registered, the oldest task (the task with the oldest registration time or the one with the oldest update time) is deleted. The upper limit for the data volume that can be downloaded at a single time is 200 kilobytes.

Restrictions on the registration count and data capacity depend on the *WiiConnect24 Programming Guidelines*. Refer to these guidelines for details.

4.1.3 Download Box

The download box is a save region that exists, one for each application, to store the downloaded contents. When first starting an application, it is created within the application's save region with any specific capacity that is within the save data capacity limitation range and must be in a state that can be used when download tasks are registered.

The download box is a VF library archive that allows referencing and updating from other applications, so the VF library API can be used to obtain contents.

4.1.4 Download Signature

WiiConnect24 can automatically verify digital signatures to guarantee that the downloaded data is not falsified. It employs RSA-SHA1 (2048-bit key length) for the signature algorithm.

Because falsified data can easily be sent to a Wii through impersonation, contents obtained via http must include a signature. Although signatures are normally obtained in a binary stream separate from the contents, with WiiConnect24 a method where the signature is included in the content file is used to simplify handling files. However, content obtained via https does not require signature verification because the communication route is secure and the legitimacy of the sender is certain. For this reason, content files are divided into two types, with and without signatures.

Details on issuing and embedding digital signatures will be provided when available.

4.2 Download Task Configuration Items

The download feature was designed with consideration to keeping the impact on the server, network, and currently running applications as small as possible. For this reason, detailed parameter settings are possible, but except for a few settings, the user does not need to worry about them.

This section describes the items that are set for download tasks.

4.2.1 URL

This specifies the location where the contents to be obtained exist.

Both http and https protocols are supported, but the following restrictions are in place to preserve security.

- http

  The WiiConnect24 signature detection feature must be enabled. An API can be used to disable the signature verification feature, but only for debug applications.

- https

  The connected server must have a certificate issued by the Nintendo CA (Certificate Authority).

4.2.2 Update Verification Interval

This specifies the interval to query the server where the update content exists. The time is specified in minutes, and the range is from a minimum value of 6 hours to a maximum value of 168 hours (one week).

The determination of this value is extremely important. Although the user can quickly obtain updated content when the update verification interval is set to be short, the load on the server and network is increased. So various factors must be considered such as the scale of the operation, content size, and update frequency.

Currently, there is no method available to later change the update verification interval specified by the application, so careful consideration is required when determining the value. If the server or network equipment turns out to be insufficient, the equipment will have to be augmented.

Please note that update verification is not always performed at the value set for the interval. The verification may be delayed due to a variety of causes, such as the Wii console being disconnected from power or restrictions by the application.

Refer to the WiiConnect24 Programming Guidelines, which includes restrictions on the update verification interval and the priority value described below.

4.2.3 Priority

When the timing for verifying updates overlaps, the task with the highest priority among the overlapping tasks is given precedence and the update verification for the other tasks is pushed back to the next time (about two minutes afterwards).

For download tasks with short update verification intervals, please specify a low priority so that other tasks have precedence. The values are specified in the range of 0–255, with a default of `NWC24_DL_PRIORITY_DEFAULT` (192). Smaller values have a higher priority.

4.2.4 Download Count

The download count is the number of times that update verification for content is performed. Each time the update verification process is performed, the count is decremented by one. When zero is reached, that download task is deleted from the download task list. No update verification process will then be performed for the deleted task until the application registers it again. The values are specified in the range of 1–100, with a default of `NWC24_DL_COUNT_DEFAULT` (1).

4.2.5 File Name

This specifies the file name when storing to the download box. Normally, the user does not need to specify this.

This specification can be used when one desires to preserve an earlier file in the download box. If a directory is prepared in advance, a location other than the application's root directory can be specified.

4.2.6 Server Side Update Interval

This specifies the interval by which the content specified in the URL is updated.

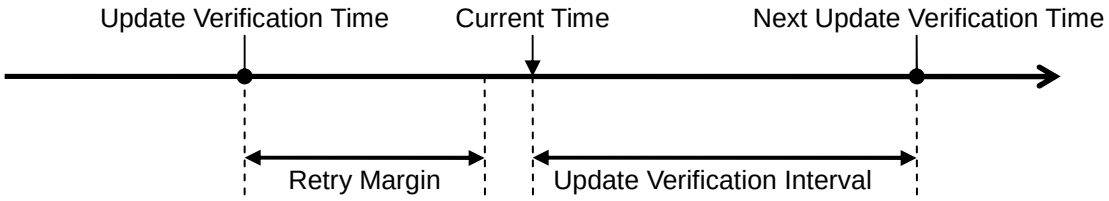
This setting is reserved for future expansion, but is not currently used.

4.2.7 Retry Margin

The download task update verification processes are not performed when the Wii power is not applied or when the application prohibits WiiConnect24 operation. For this reason, the time when the verification process is performed may be much later than the originally intended time. The amount of permissible time delay is specified in the retry margin. When a delay greater than allowed has passed

between the originally-intended update verification time and the time when it is determined a delay has occurred, the next task update verification process is performed.

Figure 4-1 When a Delay Surpassing the Retry Margin Has Occurred



It is easy for conflicts to arise with update verification processes when WiiConnect24 begins operation after being in an extended state where operation was not permitted. Afterwards, the download process may continue for a fixed period of time and negatively impact other network processes.

For tasks satisfying the following conditions, the retry margin value must be set to a smaller value so that other tasks will have precedence.

- Their update interval is short.
- Their priority is high.
- Their download content is large, and operations preferably would be avoided after starting.

Normally the user does not have to specify the retry margin, as the library automatically sets an appropriate value.

4.2.8 Flag

Various additional features for download tasks can be specified with flags.

To specify additional features, the following flags have been provided.

- NWC24_DL_FLAG_SEND_USERINFO

  Include Wii information in the file request (limited to use for debug application).

- NWC24_DL_FLAG_USE_MYPUBLICKEY

  Verify signatures by using public keys specified by the application.

- NWC24_DL_FLAG_RAW_CONTENT

  Disables the signature verification feature, and accepts unprocessed data as is (can only be set when using https).

- NWC24_DL_FLAG_USE_MYSECRETKEY

  Decode contents using the secret key configured by the application.

- NWC24_DL_FLAG_GROUP_WRITABL

  Allow task contents to be modified among applications with the same company code.

4.3 Using the Download API

The use of the *NWC24 Download API* is explained following the download task registration flow and using sample code.

4.3.1 Initializing the Download API

To use the Download API, the NWC24 library must be opened. For information on opening the NWC24 library. Close the library after use, see section 2.1 Opening and Closing the NWC24 Library.

4.3.2 Obtaining Tasks

If download tasks previously registered by the application remain in the download task list, those tasks are obtained.

Code 4-1 Example of Obtaining Download Tasks

```
NWC24DlTask mytask;
BOOL bAlreadyRegist;

NWC24Err err;

// Obtains download tasks created by the currently executing application.
err = NWC24GetDlTaskMine(&mytask);
if (err == NWC24_OK) {
   // When download tasks exist
   bAlreadyRegist = TRUE;
} else
if (err == NWC24_ERR_NOT_FOUND) {
   // When download tasks do not exist
   bAlreadyRegist = FALSE;
} else {
   OSReport("NWC24GetDlTaskMine(): Error %d¥n", err);
   OSHalt("Failed.¥n");
}
```

The `NWC24GetDlTaskMine` function obtains download tasks that were registered by an application having the same application ID as the application calling the function. To obtain tasks that were registered by an application different than the currently executing application, either call the `NWC24GetDlTaskByAppId` function or call the `NWC24GetDlTask` function using the task ID obtained with the `NWC24GetDlTaskIdByAppId` function as an argument.

4.3.3 Creating New Tasks

If no previously registered download tasks remain as download tasks, create new tasks.

The download box to save the downloaded content must be created when the application first starts.

Code 4-2 Example of Creating New Tasks

```
#define MY_DL_URL "http://foobar/dlcontents.bin"
#define MY_DLBOX_SIZE 256 * 1024

if (bAlreadyRegist == FALSE) {
   // Initialize the download task.
   if (NWC24InitDlTask(&mytask, NWC24_DLTYPE_OCTETSTREAM) != NWC24_OK) {
      OSReport("NWC24InitDlTask(): Error %d¥n", err);
      OSHalt("Failed.¥n");
   }

   // Set the URL of the download content.
   if (NWC24SetDlUrl(&mytask, MY_DL_URL) != NWC24_OK) {
      OSReport("NWC24SetDlUrl(): Error %d¥n", err);
      OSHalt("Failed.¥n");
   }

   // Set the update verification interval. (24 hour interval)
   if (NWC24SetDlInterval(&mytask, 1 * 24 * 60) != NWC24_OK) {
      OSReport("NWC24SetDlInterval(): Error %d¥n", err);
      OSHalt("Failed.¥n");
   }
}

char vfpath[64];
// Obtain the file name of the download box.
err = NWC24GetDlVfName(&mytask, vfpath, sizeof(vfpath));
if (err != NWC24_OK) {
   OSReport("NWC24GetDlVfName(): Error %d¥n", err);
   OSHalt("Failed.¥n");
}

// Re-create the download box if it failed to mount
if (VFMountDriveNANDFlashEx("C", vfpath) != VF_ERR_SUCCESS) {
   // Create the download box.
   if (NWC24CreateDlVf(&mytask, MY_DLBOX_SIZE) != NWC24_OK) {
      OSReport("NWC24CreateDlVf(): Error %d¥n", err);
      OSHalt("Failed.¥n");
   }
} else {
   VFUnmountDrive("C");
}
```

Normally, when creating a download task, it is sufficient to set only the URL where the download content exists and the update verification interval. See the *NWC24 API* for details about the other settings.

Since the download box might have been deleted by the Wii system menu or corrupted by an unexpected accident, confirm that mounting succeeded regardless of whether there are download tasks, and if mounting failed, re-create the download box with the `NWC24CreateDlVf` function.

4.3.4 Updating the Remaining Download Count

This sets the remaining download count.

Each time the update verification process for download content is performed, its download count is decremented. When the count reaches zero, that download task disappears from the download task list. For this reason, the application must restore the original value of any remaining download counts whenever it is started. However, this does not apply when execution only needs to occur once.

Code 4-3 Example of Setting the Remaining Download Count

```
#define MY_DLCOUNT 3

// Set the remaining download count.
if (NWC24SetDlCount(&mytask, MY_DLCOUNT) != NWC24_OK) {
   OSReport("NWC24SetDlCount(): Error %d¥n", err);
   OSHalt("Failed.¥n");
}
```

Code 4-3 sets the remaining download count to three. This process is executed regardless of whether the download task already existed or was new.

4.3.5 Registering Tasks

This registers download tasks to the download task list.

For tasks that are already registered, call the `NWC24UpdateDlTask` function. This function does not change the next update verification time but updates the remaining download count.

For tasks that are newly created, call the `NWC24AddDlTask` function.

Code 4-4 Example of Registering Tasks

```
if (bAlreadyRegist == TRUE) {
   // Update (re-register) a download task.
   if (NWC24UpdateDlTask(&mytask) != NWC24_OK) {
      OSReport("NWC24UpdateDlTask(): Error %d¥n", err);
      OSHalt("Failed.¥n");
   }
} else {
   // Newly register a download task.
   if (NWC24AddDlTask(&mytask) != NWC24_OK) {
      OSReport("NWC24AddDlTask(): Error %d¥n", err);
      OSHalt("Failed.¥n");
   }
}
```

4.3.6 Deleting Tasks

When download tasks are no longer necessary, for example, because the content has already been obtained, and it is unnecessary to verify updates, the download task can be deleted from the download task list by calling the `NWC24DeleteDlTask` function with the object of the task to be deleted set in an argument.

Use the API in Code 4-5 to delete a download task.

Code 4-5 API to Delete a Download Task

```
NWC24Err NWC24DeleteDlTask( NWC24DlTask taskPublic );
```

5 Relation Between the NWC24 API and the Wii Message Board

Wii has an application called the "Wii Message Board" as a standard feature of the Wii console system menu.

This is a system to exchange messages within the family or with the Wii of a friend and is implemented by using the WiiConnect24 system. Messages sent from applications can be posted to this Wii Message Board.

This chapter explains methods of doing this.

5.1 Posting Messages from Applications to the Wii Message Board

There are two major methods to post messages to the Wii Message Board using the NWC24 API.

- Posting locally to one's own Wii Message Board.

  Create and send a message with one's own Wii number in the destination address and `NWC24_MSGTYPE_WII_MENU` set as the message type.

  This feature can be used even if use of WiiConnect24 is not enabled in the WiiConnect24 setting screen.

- Send a message to a Wii Friend and post to the Wii Message Board of the other Wii console.

  Create and send a message with the Wii number of the Wii Friend (or multiple friends) in the destination address and `NWC24_MSGTYPE_WII_MENU` or `NWC24_MSGTYPE_WII_MENU_SHARED` as the message type.

  To use this feature, message delivery with WiiConnect24 must be enabled in the Wii console settings.

5.1.1 Message Format Requirements

The format of messages that are displayed on the Wii Message Board must fulfill the following conditions.

- The text to be displayed is specified in the body text, the UTF-16BE format is used as the character code, and UTF-16 formatted carriage returns (0x000A) are used as carriage return codes.
- The maximum number of characters that can be displayed is 3,000. Because it is treated as UTF-16, both single-byte and double-byte characters are two bytes.
- When posting to one's own Wii, specify the encoding passed to the `NWC24SetMsgText` function as 8 bit (`NWC24_ENC_8BIT`). To post to another Wii, specify base64 (`NWC24_ENC_BASE64`).
- Because the subject is not displayed on the Wii Message Board, create the message with the subject blank.

- By attaching data in the NWC24_APP_WII_MSGBOARD format as attached binary data, the background pattern used when displaying on the Wii Message Board can be specified as letterhead data. (For information regarding letterhead data, see section 5.3 Letterhead Template.)
- When letterhead data is not attached, the default letterhead is used.

5.1.2 Cautions when Posting Messages

Normally, specifications call for the sender's nickname registered in the Wii friend roster to be displayed in the sender's field of the message posted to the Wii Message Board. For this reason, even when messages sent by applications are posted to one's own Wii console locally, one's own address is displayed as the sender.

To prevent this situation, a mechanism to display a different name (game name or the name of a game character) in the sender's field of the Wii Message Board is provided. The different name that is displayed is called the alternate nickname.

The alternate nickname is set with the same format as the nickname registered with the Wii friend roster and is given precedence and displayed over the name set in the Wii friend roster. The alternate nickname is set using the NWC24SetMsgAltName function and must fulfill the following conditions.

- The character code must be specified using UTF-16BE format.
- It can be up to 35 characters, and the terminal character (0x0000) counts as one character.
- A single carriage return can be included. Use the UTF-16 carriage return (0x000A), which counts as one character.
- Up to 17 characters can be on one line (when there is no carriage return, the line wraps after the 17th character automatically, but display may be distorted depending on the character width).

Use the NWC24SetMsgMBNoReply function so that users cannot reply to messages posted to the Wii Message Board from an application. This function can hide the reply button when messages are displayed in the Wii Message Board.

5.1.3 Sending Application Data Simultaneously

When notifying the Wii Message Board that data to be used by the application has been sent in a message, two types of messages must be prepared: one for the application to use and one to post to the Wii Message Board.

Include body text for the notification and, if necessary, letterhead data in the message to be sent to the Wii Message Board. Attach the binary data without adding body text to the message for the application and then send the message.

© 2007-2008 Nintendo
CONFIDENTIAL

5.1.4 Detailed Control Features

A received message can be posted to any date on the Wii Message Board calendar. Specify any date between January 1, 2000 and December 31, 2035 with the `NWC24SetMsgMBRegDate` function. This function only specifies the date on the calendar to which to display the message. It does not cause the message to be displayed only when that date arrives.

To delay the display of a message, use the `NWC24SetMBDelay` function. This function delays the display of a message on the local Wii Message Board. The range of time that can be specified for the delay is from one hour to 240 hours (10 days), specified in hours. To delay display of a Wii Message Board message sent over the network, use the `NWC24SetMsgDesignatedTime` function to delay the message transmission itself. The `NWC24SetMBDelay` and `NWC24SetMsgDesignatedTime` functions can only be used when messages are created on a Wii with Wii Menu 3 installed.

5.2 Timing when the Wii Message Board Processes Messages

The Wii Message Board is implemented as part of the Wii console's system menu.

When the Wii Menu starts, the message box is searched and messages to the Wii Message Board are picked up and registered to the Wii Message Board. For this reason, messages that are posted locally or received externally while the application is running are not reflected in the Wii Message Board until returning to the Wii Menu the next time.

5.3 Letterhead Template

Normally, messages posted to the Wii Message Board from an application are displayed with the default design prepared for the Wii Message Board, but it can be displayed with a design consistent with the presentation design of the application. When letterhead data for the display of a designed letterhead is attached to the message posted to the Wii Message Board, it is drawn based on that letterhead data. This mechanism is called the letterhead template.

Although effects and controls other than rendering are possible when letterhead templates are used, only rendering is discussed here. For information on non-rendering topics, see the reference manual.

5.3.1 Composition of Letterhead Data

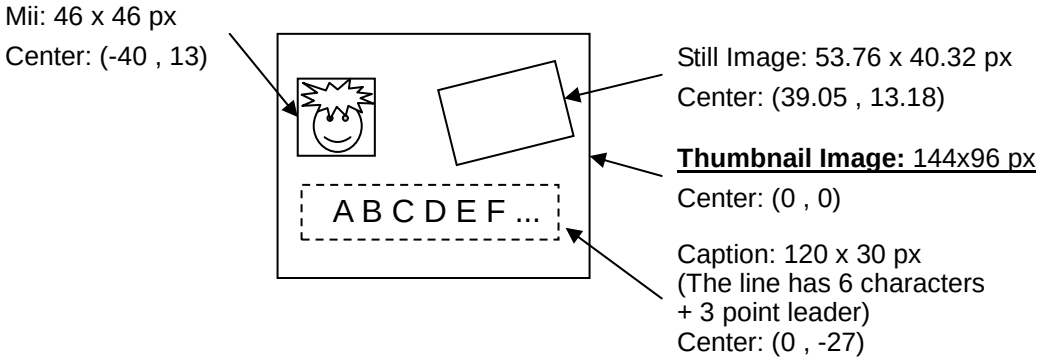
With the letterhead template, rendering is performed using a total of 10 images: one thumbnail image and nine images for enlarged display (three each of the header, body, and footer.) All the images are created in TPL format and with designated file names and folder structure.

The LZ77 compressed archive of these images is called the letterhead data, and this data is attached to the message as `NWC24_APP_WII_MSGBOARD` format data.

5.3.2 Thumbnail Image

The location of each part arranged in the thumbnail image to be displayed on the Wii Message Board is shown in Figure 5-1.

Figure 5-1 Thumbnail Image Part Distribution

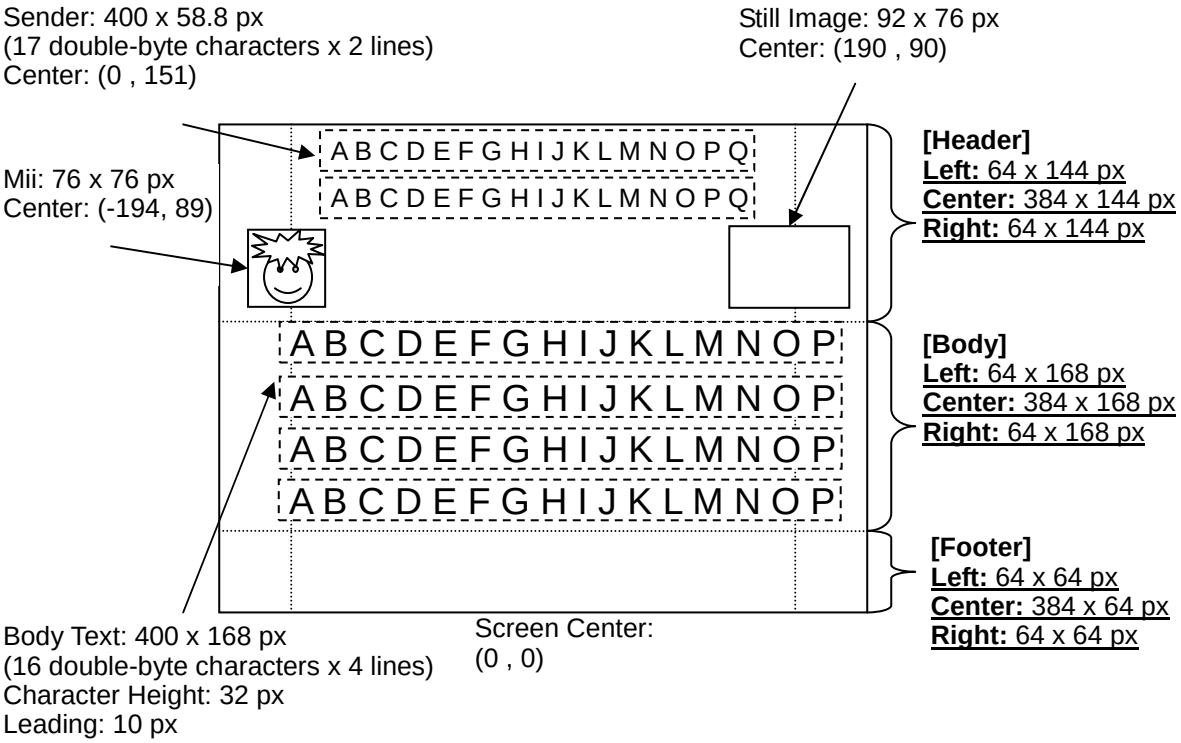


The caption portion displays the first six characters of the sender's name and a three point leader. (The nickname is used when the recipient is registered as a Wii Friend by the recipient. The alternate nickname has precedence when it is set with the `NWC24SetMsgAltName` function.)

5.3.3 Images for Enlarged Display

The location of each part of the image for enlarged display, which is displayed when a thumbnail image is selected on the Wii Message Board, is shown in Figure 5-2.

Figure 5-2 Part Distribution of the Image for Enlarged Display



The body text portion is displayed protruding 8 px to the left and right of the body center part. When the body text exceeds four lines, the body portion is repeated in four-line units.

5.3.4 Restrictions on Images Used in Letterhead Data

Each image used in the letterhead template must be of the same or smaller height and width as each part indicated in sections 5.3.2 Thumbnail Image and 5.3.3 Images for Enlarged Display. When the height/width size is smaller, the image is scaled and then displayed.

The supported file format for images is only the TPL (texture palette library) format. By using `TexConv.exe`, TGA format images can be converted to TPL format.

Please use a format supported by the layout library, other than `GX_TEX_RGBA8`, for the texel format. Specifically, the following formats can be used.

- GX_TF_I4
- GX_TF_I8
- GX_TF_IA4
- GX_TF_IA8
- GX_TF_RGB565
- GX_TF_RGB5A3
- GX_TF_CMPR
- GX_TF_C4
- GX_TF_C8

5.3.5 Creating Letterhead Data

First, create images for each part using `darch.exe` to create two archive files for the thumbnail image and the enlarged image. At this time, use the following folder structure and file names for the archive.

- Thumbnail Image

  img/my_LetterS_b.tpl

- Image for Enlarged Display

  img/my_Letter_a.tpl ... (Header Left)

  img/my_Letter_b.tpl ... (Header Center)

  img/my_Letter_c.tpl ... (Header Right)

  img/my_Letter_d.tpl ... (Body Left)

  img/my_Letter_e.tpl ... (Body Center)

  img/my_Letter_f.tpl ... (Body Right)

  img/my_Letter_g.tpl ... (Footer Left)

  img/my_Letter_h.tpl ... (Footer Center)

  img/my_Letter_i.tpl ... (Footer Right)

The thumbnail image should be archived as a file named `thumbnail.arc`, and the enlarged images for display should be archived as a file named `letter.arc`.

Next, use `ntcompress.exe` to perform LZ77 compression on these archive files. These must be 4-byte aligned.

Example:

```
$ ntcompress -l -A4 -o thumbnail_LZ.bin thumbnail.arc

convert thumbnail.arc(27808Byte) to thumbnail_LZ.bin(4580Byte)

$ ntcompress -l -A4 -o letter_LZ.bin letter.arc

convert letter.arc(369536Byte) to letter_LZ.bin(56816Byte)
```

The compressed archive files should be archived with the following folder structure and file names.

```
./thumbnail_LZ.bin

./letter_LZ.bin
```

The archive file obtained in the last step can be freely named. Although this archive file can be attached to a message as letterhead data, **be sure that the file size does not exceed the 120 kilobyte limit**.

5.3.6 Attaching Letterhead Data

The letterhead template is a Wii Message Board-specific feature. The letterhead template feature cannot be used from a PC or from non-Wii applications.

To use this feature, attach the created letterhead data as `NWC24_APP_WII_MSGBOARD` format binary data to a Wii message that has either `NWC24_MSGTYPE_WII_MENU` or `NWC24_MSGTYPE_WII_MENU_SHARED` specified as the message type.

6 Friend Roster

The Wii console provides a Wii friend roster to register the Wii number or e-mail address of partners that are used to send and receive messages with WiiConnect24.

The Wii friend roster can be managed with the Wii address book that is provided as a feature of the Wii Message Board and has the following characteristics.

- Can be used from applications.
- Can register up to 100 Wii numbers or e-mail addresses.
- The Wii console does not accept (as a rule) messages from non-registered addresses.

The NWC24 Friend Roster API provides an environment to access the Wii friend roster.

6.1 Using the Friend Roster API

The use of the NWC24 Friend Roster API is explained using sample code for obtaining friend information.

6.1.1 Initializing the Friend Roster API

To use the Friend Roster API, the NWC24 library must be opened. For information on opening and closing the NWC24 library, see section 2.1 Opening and Closing the NWC24 Library.

6.1.2 Obtaining Friend Information

The following sample code (Code 6-1) stores only the friend information of those with whom a friend relationship is established to the list prepared from the specified item number.

Code 6-1 Example of Obtaining Friend Information of Wii Friends with whom a Relation is Established

```
/*---------------------------------------------------------------------------*
  Name        :  GetEstablishedFriendInfo
  Description :  Gets for "list" the friend information of the Wii friend with
                 whom a relation is established from the "start" numbered item
                 up to a maximum of "listSize" items.
  Arguments   :  list      -  List where the obtained friend information is stored.
                 listSize  -  The size of the list.
                 start     -  Specifies from what item number to store information
                              to the list.
                 allocator –  Allocator used to obtain the friend information.
  Returns     :  Number of items stored in the list. If an error is generated,
                 returns -1.
 *---------------------------------------------------------------------------*/
```

```
int GetEstablishedFriendInfo(NWC24FriendInfo *list, int listSize,
                             int start, MEMAllocator* allocator)
{
    NWC24FriendInfo* info;
    int numReturns;
    int numEstInfos;
    u32 numInfos;
    NWC24Err err;
    int i, ierr;
    // The memory storing the friend information must be 32-byte aligned.
    info = (NWC24FriendInfo*) MEMAllocFromAllocator(
                                allocator, sizeof(NWC24FriendInfo));
    // Obtains the number of friend information items that can be registered to the
    // Wii Friend Roster.
    err = NWC24GetNumFriendInfos(&numInfos);
    if (err != NWC24_OK) {
        OSReport("NWC24GetNumFriendInfos(): Error %d¥n", err);
        MEMFreeToAllocator(allocator, info);
        return -1;
    }
    numReturns = 0;
    numEstInfos = 0;
    for (i = 0; i < numInfos; i++) {
        // Checks whether friend information is registered in the specified index.
        ierr = NWC24IsFriendInfoThere(i);
        // A returned negative value means an error was generated.
        if (ierr < 0) {
            OSReport("NWC24IsFriendInfoThere(): Error %d¥n", ierr);
            MEMFreeToAllocator(allocator, info);
            return -1;
        }
        // Friend information is not registered.
        if (ierr == 0) continue;
        // Friend information is registered.
        if (ierr == 1) {
            // Obtain friend information.
            memset(info, NULL, sizeof(NWC24FriendInfo));
            err = NWC24ReadFriendInfo(info, i);
            if (err != NWC24_OK) {
                OSReport("NWC24ReadFriendInfo(): Error %d¥n", err);
                MEMFreeToAllocator(allocator, info);
                return -1;
```

```
            }
            // Go to next if friend information does not have an established friend
            // relationship.
            if (info->attr.status != NWC24_FI_STAT_ESTABLISHED) continue;
            // Go to next if the friend information does not have a registered Wii
            // number or e-mail address.
            if ((info->attr.type != NWC24_FI_TYPE_WII)
             && (info->attr.type != NWC24_FI_TYPE_PUBLIC)) continue;
            // Skip until the “start”-numbered item is obtained.
            numEstInfos++;
            if (numEstInfos <= start) continue;
            // Copy friend information to the list.
            memcpy(&(list[numReturns]), info, sizeof(NWC24FriendInfo));
            numReturns++;
            // Finish obtaining if the list is full.
            if (numReturns >= listSize) break;
        }
    }
    MEMFreeToAllocator(allocator, info);
    return numReturns;
}
```

The friend information registered in the Wii friend roster does not necessarily exist from the 0th index. This sample searches for friend information from the beginning of the list each time.

Before actually obtaining friend information, confirm with the `NWC24IsFriendInfoThere` function whether friend information is registered in the specified index. If 0 is returned, no friend information is registered. If 1 is returned, information is registered. A negative value is returned when an error is generated.

Whether a friend relationship has been established can be confirmed with the value of the `NWC24FriendInfo.attr.status` member. As a rule, send messages only to Wii Friends with whom a friend relationship has been established.

Whether the address in the friend information is registered as a Wii number or registered as an e-mail address can be confirmed by the value of the `NWC24.attr.type` member. In particular, because messages to Wii Message Boards can only be sent to Wii Friends who have a registered Wii number, please confirm this member value when obtaining friend information to select a send destination.

6.1.3 Searching for Friend Information

By using the following functions, the index where a given friend's information is stored can be obtained from their Wii number or e-mail address. See the *NWC24 API* for details.

Code 6-2 API to Search for Friend Information

```
NWC24Err NWC24SearchFriendInfoById( NWC24UserId id, u32* index );
NWC24Err NWC24SearchFriendInfoByAddr(
                        const NWC24FriendAddr* addr, u32* index );
```

6.1.4 Number of Friend Information Items

Use the following functions to obtain the number of friend information items that can be registered in the friend roster (`NWC24GetNumFriendInfos` function), the number of friend information items registered (`NWC24GetNumRegFriendInfos` function), or the number of friend information items that are registered and have established friend relationships (`NWC24GetNumEstFriendInfos` function). See the *NWC24 API* for details.

Code 6-3 API to Obtain the Number of Friend Information Items

```
NWC24Err NWC24GetNumFriendInfos( u32* num );
NWC24Err NWC24GetNumRegFriendInfos( u32* num );
NWC24Err NWC24GetNumEstFriendInfos( u32* num );
```

6.2 Registering to the Friend Roster

Although the NWC24 Friend Roster API provides functions related to registering and deleting friend information to and from the Wii friend roster, please do not implement them in production applications unless there is a special reason.

6.3 Nickname Display

The nickname in the friend information (the `NWC24FriendInfo.attr.name` member) is input from the address book by the user, so the character set for display is the same character set that can be used for input to the address book.

See the forthcoming *Character Sets that can be Input to the Address Book*, planned to be added to the *NWC24 API*, for information regarding the character set that can be input to the address book.

7 Scheduler Operation

Wii consoles that have communication ON via WiiConnect24 have messages sent and received and data downloaded automatically in the background (as needed when connected to the network) even if an application is running. Although these processes occur infrequently (about once every several minutes) applications may be affected in the following ways.

- Loss of performance in high performance-demanding applications
- Worsening of network communication response
- Delay in accessing Wii console NAND memory

For cases where the above influences are expected, the automatic processes can be temporarily stopped by using the scheduler operation API. The API that performs the stop and restart processing for the automatic processes can be called at any time in relation to the application. For example, the automatic processes can be stopped before loading a large amount of data from Wii console NAND memory and then restarted after the load has completed. However, if an attempt to stop an automatic process is made while it is running, a short time is required before it stops.

7.1 Scheduler Operation API

The scheduler operation API provides the following functions.

Code 7-1 API to Operate the Scheduler

```
s32 NWC24SuspendScheduler( void );
s32 NWC24TrySuspendScheduler( void );
s32 NWC24ResumeScheduler( void );
```

Both the `NWC24SuspendScheduler` and `NWC24TrySuspendScheduler` functions temporarily stop the scheduler. The difference between the functions is that the former blocks processes until the scheduler stops, and the latter does not block processes.

The `NWC24ResumeScheduler` function restarts scheduler processes that have been temporarily stopped.

See the *NWC24 API* for details on each function.

7.1.1 Relation to the NWC24 Library

The scheduler operation API can be called at any time by the application regardless of the open/close status of the NWC24 library.

While the NWC24 library is open, the scheduler is in a stopped state. The scheduler does not resume immediately even if the `NWC24ResumeScheduler` is executed, but resumes after the NWC24 library closes.

7.1.2 Scheduler Initial State

Applications that incorporate the NWC24 library (that are linked with `nwc24[D].a`) have the scheduler in operational status (the `NWC24ResumeScheduler` function is called) at startup by default.

Applications that do not incorporate the NWC24 library have the scheduler in a paused state by default.

7.2 Scheduler Operational Conditions

The following two automatic processes are performed by the WiiConnect24 scheduler

- Send/Receive Message Process
- Download Process

These processes are started at fixed intervals on Wii consoles for which the Wii Network Service User's Agreement has been completed and the WiiConnect24 setting is ON. However, if sending/receiving messages has been prohibited by the user with the parental control feature settings, the message send/receive process is not started.

Use the `NWC24Check` function to make the determination of whether the conditions for these processes to operate have been met.

7.2.1 Start Timing for Send/Receive Message Process

The interval for starting the message send/receive process is set to one minute by default. In other words, one minute after the application starts up, the first message send/receive process starts. If communication with the server completes here with no problem, the startup interval is reset to the value specified by the server (normally, this is 10 minutes, but it may be changed depending on the situation). Thereafter, the message send/receive process starts at that reset interval.

If communication with the server fails, the startup interval is lengthened by one minute for each failure. In other words, if communication does not succeed even once after the application starts up, the interval between attempts is increased incrementally to 1, 2, 3, 4, ... (maximum of 10) minutes.

If the scheduler is in a paused state when the startup time arrives, the startup for that time is allowed to pass and a new startup interval begins, after which the next startup attempt will be made.

This startup timing may be changed in the firmware of future releases.

© 2007-2008 Nintendo
CONFIDENTIAL

7.2.2 Start Timing for the Download Process

The interval for starting the download process is set to two minutes by default. In other words, two minutes after the application starts up, the first download process starts.

The download process first determines whether there is a download task that has reached its time to be executed by checking the download task list. If there is a download task that should be executed, that task is processed immediately. If no such task exists, the startup interval is set to 11 minutes and the process terminates.

When several tasks have reached the time to be executed at the same time, the task with the highest priority is executed immediately, the interval is set to two minutes and then the process is terminated. In other words, the remaining tasks are executed every two minutes.

If the scheduler is in a paused state when the startup time arrives, the startup for that time is allowed to pass, and a new startup interval begins, after which the next startup attempt will be made.

This startup timing may be changed in the firmware of future releases.

© 2007-2008 Nintendo
CONFIDENTIAL

8 Miscellaneous Information

8.1 Operational Environment on the Production Version

Some Wii consoles cannot use communication features when shipped.

For WiiConnect24 to operate, an environment with system menu version 2.0 or later is required to be installed. Wii consoles that have successfully connected to the network are guaranteed to have version 2.0 or later because the Wii console update is automatically performed after the first successful test connection.

If the system menu is version 2.0 or later, the version information is displayed at the upper right of the Wii System Settings screen. If nothing is displayed, the system menu is a version older than 2.0.

8.2 Connection with Broadband

WiiConnect24-compatible game software assumes its connection to the Internet is a broadband connection (ADSL, FTTH, or cable).

It is alright if narrow band connections, such as through a modem, cannot be made.

© 2007-2008 Nintendo
CONFIDENTIAL

8.3 Checking the NWC24 Library Status

A check of whether the system can be used must be performed by calling the `NWC24Check` function after the NWC24 library is opened when performing the operations below using the WiiConnect24 features.

- When creating a Wii message to send to another Wii or an external e-mail address (this does not include posting messages to the Wii Message Board on one's own Wii).
- When prescheduling download tasks.
- When referencing the content obtained via a download task.
- When changing the settings of an application feature using WiiConnect24 from disabled to enabled.

When the return value of the `NWC24Check` function is an error, as indicated in Table 8-1, display the error message specified in the guidelines. However, if there was a problem connecting to the server or if the send box exceeded its capacity, normal status may be restored by calling the `NWC24Check` function again after some time passes.

Table 8-1 Error Codes Returned by the NWC24Check Function and Their Causes

Error Code (Return Value)	Error Cause
`NWC24_ERR_DISABLED`	The WiiConnect24 feature is not enabled in the WiiConnect24 settings screen
`NWC24_ERR_NETWORK`	Either a problem exists in the Internet related settings in the Wii System Settings or Internet connection is not possible due to a temporary problem
`NWC24_ERR_SERVER`	A problem occurred during connection to the WiiConnect24 server or there is some obstacle and connection is not possible
`NWC24_ERR_FULL`	The send box has exceeded capacity and a new message cannot be created

8.4 NWC24API Errors

There are occasions when "File corrupted" or "Other fatal error" is returned when calling a NWC24 library function. In these cases, please display the error message specified in the guidelines.

8.5 Messages to Those with whom a Friend Relationship is not Established

As a rule, do not set as the message destination any address that is not an address of a Wii Friend with whom a friend relationship is established and is registered in the Wii friend roster.

© 2007-2008 Nintendo
CONFIDENTIAL

8.6 Application-specific Wii Message Names

It is alright for application-specific Wii messages to be given a different name, such as "Letters." However, avoid using several different names such as "Letters", "Mail", and so on, to avoid confusing the user about what is being specified.

8.7 Definition of Text Data and Non-text Data

Character data that can be displayed on the Wii Message Board is defined as "Text Data"; character data that cannot be displayed is defined as "Non-Text Data."

8.8 Specifications Related to the Destination Region

WiiConnect24 specifications allow for communications regardless of the destination market of the game software.

For example, in cases where you want to limit communication partners to those in the same market, a mechanism must be incorporated into the application to determine the destination market from, for example, the Game Code, and perform communication only if the destination markets are the same. The same applies to the following cases.

- To communicate with users using only the same language.
- To limit the character sets to be prepared to the minimum required level.
- To limit communication partners to a specific physical area.

8.9 Communication Between Different Applications

Communication between different applications can be determined by each application involved. However, please contact support@noa.com with the details of the planned approach prior to development.

The other company and product names contained in this document are the trademarks or registered trademarks of the respective companies.

© 2007-2008 Nintendo

The contents of this document cannot be duplicated, copied, reprinted, transferred, distributed or loaned in whole or in part without the prior approval of Nintendo.