WiiConnect24 Programming Manual

Version 1.2.2

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

RVL-06-0271-001-D
Released: February 16, 2009

Table of Contents

Code

Tables

Figures

Revision History

Version	Revision Date	Description
1.2.2	2008/08/15	• Added an explanation on handling keys in section 5.4.2 Public Keys and Shared Keys.
1.2.1	2008/07/29	• Added section 5.4 Configuration Items for Individual Applications
1.2.0	2008/04/13	• Added Chapter 2 Structure of the WiiConnect24 System • Added section 3.1 Notes for Designing Applications • Added section 5.1 Splitting Usage with the RVL DWC(-DL) Library • Removed explanations of independent server use from section 5.3.2 Update Verification Interval • Added section 5.4.7 Executing Downloads Immediately • Added section 5.4.8 Getting the Errors Recorded for a Download Task • Added section 5.5 Precautions When Designing Receivable Content and section 5.6 Precautions During Development • Added section 8.4 Stopping the Scheduler
1.0.1	2007/12/06	• Added a note to section 3.2 Creating Messages concerning specification changes to slot illumination in Wii Menu version 3.0 • Added supplementary information to section 4.1.2 Download Task List and section 4.2.2 Update Verification Interval on the existence of restrictions in the Programming Guidelines • Changed section 4.3.3 Creating New Tasks to confirm that the download box was successfully mounted • In section 5.1.1 Message Format Requirements and section 5.3.2 Thumbnail Image, deleted the explanation stating that still images cannot be attached to messages for the Wii Message Board • Added an explanation to section 5.1.4 Detailed Control Features on how to delay displaying to the Wii Message Board
1.0.0	2007/07/24	Initial version.

1 Introduction

This document describes how to develop 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

To prepare the development environment, refer to the "Quick Start" section in the RevoEX README.

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

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 Wii 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: The Wii Menu’s network update feature cannot be used in the development environment.

1.2.3 Configuration Procedures to Use the Communication Features

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

1. Configure the network connection settings.
2. Test the network connection.
3. Agree to the End User License Agreement (EULA).
4. Configure the WiiConnect24-related settings.

For details on configuration methods and tools, see the *Network Development Environment* document included in the RevoEX package.

1.2.4 Support Codes and Errors

After completing the procedures above, a five-digit support code will be displayed. Ignore this code; it indicates the characteristics of the communication environment and is not necessarily an indication of an error.

For details on the error correspondence table or error generation methods, see the *Network Development Environment* document. Although descriptions of Wii network-related errors and WiiConnect24 errors are provided, see the *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

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

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

- WiiConnect24

  If OFF, WiiConnect24 cannot be used.

- Parental Controls

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

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

2 Structure of the WiiConnect24 System

WiiConnect24 features were created to be always operable, regardless of the operating state of the Wii console. Consequently, some of the components that constitute WiiConnect24 differ from other libraries.

An overview of the constituent components is given below.

2.1 NWC24 Firmware (Subsystems)

The NWC24 firmware exists within the Wii console's firmware and forms the core program of WiiConnect24. This program can run in the background while the Wii console is in standby mode and also while applications are running.

The NWC24 firmware is roughly split into three internal modules that have the following responsibilities.

- Mail Transmission Module

  Checks the outbox for queued outgoing messages and sends them to the server; also queries the incoming mail server for new messages and saves them to the inbox.

- Download Module

  Accesses the scheduling data (download task) for each application's registered downloads and performs actual download of tasks whose preset times have arrived.

- NWC24 Scheduler

  Runs the two aforementioned modules at fixed intervals. Applications running in the foreground are able to suspend this scheduler's operations and thereby prevent the WiiConnect24 firmware from running in the background.

2.2 NWC24 Library

The NWC24 library communicates with the NWC24 firmware, issues commands, and provides procedures for accessing NWC24-managed data. For details, see Chapter 3 NWC24 Library.

2.3 NWC24 and Wii Console NAND Memory

WiiConnect24 reads and writes a number of files in Wii console NAND memory to run across Wii console operating states and application barriers.

2.3.1 NWC24 System Files

The NWC24 system files contain system information required to run WiiConnect24. Account information for the incoming and outgoing mail servers, mailboxes to store composed and received messages, download scheduling data, and other information are stored in these system files.

The NWC24 library is used to access this information (some can be accessed only by the firmware). WiiConnect24 system files are saved in a special region of the NAND file system and cannot be accessed by normal means, such as through the NAND API. As a result, the application restrictions on NAND memory usage and inode count do not apply. Developers do not need to be conscious of the existence of these files while writing code.

The system files are created and initialized the first time the Wii Menu is started. Therefore, NWC24 system files must be created and initialized using a designated initialization tool on development consoles that do not have the Wii Menu installed.

2.3.2 NWC24 User Files

Apart from NWC24 system files, there are also files that each application creates in the save data directory to allow access to the NWC24 firmware. Since these files are created in the save data directory, they are affected by NAND usage and inode count restrictions for applications and must therefore be checked using the `NANDCheck` function in the same manner as normal save data.

These files correspond to the download box and download keys described later.

2.4 Wii Menu and Wii Message Board

The Wii Message Board is an application that comes pre-installed on the Wii console as a Wii Menu feature. Implemented using the WiiConnect24 system, it is a mechanism for exchanging messages between family members and with other friends' Wii consoles. Messages sent from applications can also be posted to the Wii Message Board.

3 NWC24 Library

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

3.1 Notes for Designing Applications

The NWC24 library is not thread-safe. Do not call NWC24 library functions from two or more threads at the same time. Also, some NWC24 library functions may block, so we strongly recommend preparing a NWC24 library processing thread that is solely dedicated to calling NWC24 library functions.

3.2 Opening and Closing the NWC24 Library

To use the functions for sending messages, managing download tasks, or accessing the Wii console's friend roster, you must first open the NWC24 library.

If some of the functions are used when the NWC24 library is not opened, an error is generated. While the NWC24 library is opened, the NWC24 scheduler is temporarily stopped to maintain consistency. Because the sending and receiving of messages cannot be performed if the NWC24 library is left open, be sure to close the library after using the message API or when closing the application. See Chapter 8 Scheduler Operation for details on stopping and restarting the NWC24 scheduler.

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

Code 3-1 Functions to Open and Close the NWC24 Library

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

The following sample code opens and closes the NWC24 library.

Code 3-2 Example for Opening and 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;
}
```

3.2.1 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 3-1, 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 3-1 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
Wii 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 Wii Menu, so the application does not need to perform restoration processes.

3.2.2 Closing the NWC24 Library

After the NWC24 library is closed, 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. See section 8.3.3 Startup Timing After Resuming the Scheduler for information on behavior when the automatic send/receive feature is resumed.

3.3 Checking the NWC24 System's Operational State

When taking the following types of actions using WiiConnect24 features, you must call the `NWC24Check` function after opening the NWC24 library to check that the system is in a usable state.

- Creating a Wii Message to send to another Wii console or an external e-mail address (this does not include posting to the local console's Wii Message Board)
- Scheduling a download task
- Accessing content that was obtained by a download task
- Enabling disabled application features that use WiiConnect24

When the `NWC24Check` function returns an error value indicated by Table 3-2, display the error message specified by the *Programming Guidelines*. However, if there is a problem connecting to the server or the outbox has exceeded its size quota, another call to the `NWC24Check` function after some time has passed may show that the operational state has returned to normal.

Table 3-2 Error Codes Returned by the NWC24Check Function and Their Causes

Error Code (return value)	Error Cause
NWC24_ERR_DISABLED	WiiConnect24 features are not enabled in the Wii System Settings.
NWC24_ERR_NETWORK	Either there is a problem with Internet-related Wii System Settings or some sort of temporary difficulty is preventing Internet connections.
NWC24_ERR_SERVER	Either there has been a problem connecting to the WiiConnect24 server or some other difficulty is continuing to prevent connections.
NWC24_ERR_FULL	The outbox has exceeded its size quota and new messages cannot be created.
NWC24_ERR_PROTECTED	Use is restricted by Parental Controls.
NWC24_ERR_FATAL	A fatal error has occurred.

See the *Function Reference Manual* for information on each processing sequence and other details.

4 Messages

The NWC24 message API provides the environment wherein messages that are exchanged between two Wii consoles or an external e-mail address can be created, viewed, and managed.

4.1 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 3.2 Opening and Closing the NWC24 Library.

4.2 Creating Messages

The sequence for creating messages is described using an example from NWC24 library sample code, as shown in Code 4-1.

Code 4-1 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. The type of message to create must be decided at the time the message object is initialized. 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. The buffer used to specify the e-mail address must not be released until the `NWC24CommitMsg` function is called and the message has finished being created.

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 `NWC24CommitMsg` function is called and the message has finished being created.

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 4-1, 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. When designating a string that cannot be represented using 7 bits, either specify `NWC24_ENC_BASE64` or convert the encoding of the designated string. See section 6.1.1 Message Format Requirements when sending messages to the Wii Message Board because several restrictions apply.

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 `NWC24CommitMsg` function is called and the message has finished being created.

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. 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 4-1, are also available.

With the `NWC24SetMsgAttached` function, up to two binary data files can be attached to a message. See Table 4-1 for types of data that can be attached.

Table 4-1 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	O	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 version 3.0, the slot can also be illuminated when an application posts a message directly to the Wii Message Board. However, with Wii Menu version 2.0, the slot can only be illuminated when the message was sent via the network. 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.

4.3 Obtaining the Message List

The sequence for obtaining the message list is described using an example from NWC24 library sample code (Code 4-2).

Code 4-2 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, numMsgs);
    if ( err != NWC24_OK ) {
        OSReport("NWC24GetNumMsgList(): error %d\n", err);
        MEMFreeToAllocator(&Allocator, idListBuf);
        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);
            MEMFreeToAllocator(&Allocator, idListBuf);
            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.

4.4 Viewing Messages

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

4.4.1 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 4-3 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;
}
```

4.4.2 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 4-4 Example of Obtaining the Recipient

```
/*------------------------------------------------------------------------*
   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;
}
```

4.4.3 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.

Code 4-5 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 4-5 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;
}
```

4.4.4 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 4-6 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;
}
```

4.4.5 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.

Code 4-7 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 4-7 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;
}
```

4.4.6 Obtaining the 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 4-8 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);
         MEMFreeToAllocator(&Allocator, buffer);
         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;
}
```

4.4.7 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.

4.5 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 4-9 to delete a message.

Code 4-9 Function to Delete Messages

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

4.6 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 `NWC24SetSearchCondMsgBox`, `NWC24SetSearchCondFromAddrId`, `NWC24SetSearchCondAppId`, and `NWC24SetSearchCondForMenu`.

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 4-10 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 4-10 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();

   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;
}
```

5 Downloading

While an application is executing or in standby mode, the Wii can automatically download content from the server and synchronize it with Wii console NAND memory. Using this feature, the application can obtain add-on data without making the user wait.

Using the *NWC24 Download API* to control scheduling information (download tasks) for this automatic synchronization, a developer can write a program to automatically download data (content) from a designated location (URL) on the network without needing to write network communication processing code. Using the *Download API* also guarantees that the downloaded data is not falsified.

5.1 Splitting Usage with the RVL DWC(-DL) Library

The ability to synchronize content using the NWC24 download feature significantly reduces the user's wait time.

However, there are several restrictions and disadvantages associated with using the NWC24 download feature, and depending on the intended purpose, the NWC24 download feature might not be the most appropriate feature to use. Switch between it and the download feature provided by the RVL DWC(-DL) library to suit the application.

5.1.1 Content Size Restrictions

The NWC24 download feature is not well suited for downloading large pieces of content, for the following reasons.

- Size restrictions from the Programming Guidelines

  Even when the Wii console is in standby mode, WiiConnect24 periodically downloads data and saves it to Wii console NAND memory. When dealing with large pieces of content, depending on their update frequency on the server, there is a danger that downloading these files with the NWC24 download feature will adversely affect the network's load or the lifetime of Wii console NAND memory. For this reason, size restrictions have been established in the Programming Guidelines.

  During use of the NWC24 download feature, a storage region of the maximum expected size must be reserved within the application's save data region. Considering the number of save data blocks required for this is another reason why the NWC24 download feature is not recommended for handling large data.

- Increase in download time

  NWC24 downloads might take longer than direct downloads using a library such as the RVL DWC(-DL) library. This difference is a result of first saving the obtained content in Wii console NAND memory and then requiring the application to reload the data that was written there. These reads and writes incur VF library overhead. The difference in download time grows more pronounced with the size of the content to handle.

When working with large content, divide it between the NWC24 and RVL DWC(-DL) libraries so that the NWC24 download feature handles update information and other control data that is small in size, and the RVL DWC(-DL) library handles the main data content.

5.1.2 Immediacy

NWC24 downloads are suitable for handling content that is preferably updated, but that does not necessarily need to be in the most recent state. The NWC24 firmware checks for updates at times entirely unrelated to user interaction.

The application can immediately start a download if the most recent possible content is required. This is useful as a backup strategy in situations such as when content is updated on the server and the user starts an application immediately afterwards, preventing the application from obtaining the most recent content. However, this download will take longer to process than an RVL DWC(-DL) download, as explained in section 5.1.1 Content Size Restrictions.

If a disc application always needs the most recent content possible, it is advisable to use only the RVL DWC(-DL) library. The reason is that because it takes time to load from a disc and perform other such processes, it is difficult for users to notice when data is being downloaded in the background.

Because WiiWare titles have short loading times on startup and can use Channel Script to directly handle content obtained through NWC24 downloads, NWC24 downloads are more compatible with WiiWare.

5.2 Terminology and Overview

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.

5.2.1 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.

5.2.2 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 rewrite time) is deleted. The upper limit for the data volume that can be downloaded at a single time is 500 kilobytes.

Restrictions on the registration count and data capacity depend on the *WiiConnect24 Programming Guidelines*. For details, see these guidelines.

5.2.3 Download Box

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

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

5.2.4 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 spoofing attacks, contents obtained through HTTP must include a signature. WiiConnect24 includes the signature in the content file to simplify handling files. However, content obtained through 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 uploaded to the server are divided into two types: with and without signatures.

In order for an application to use the digital signature feature, it is necessary to configure keys.

5.3 Download Task Configuration Items

The download feature was designed with consideration for keeping the impact on the server, network, and currently running applications as small as possible. For this reason, detailed parameter settings are possible, but developers need be concerned about only a few of them.

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

5.3.1 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).

5.3.2 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). Longer times will reduce the content update frequency, making it easier for content to be out-of-sync with the server for longer periods of time. However, this will also increase the maximum time period during which the download task will exist in the task list. For this reason, decide on a value according to the characteristics of the data to distribute. (For example, using a design that requires as little immediacy as possible and allows tasks to be registered for a long time is effective because it can bring back users who have not played in a while.)

Refer to the *WiiConnect24 Programming Guidelines*, provided separately, for established restrictions on the update verification interval and the priority value, discussed later in this document.

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 application restrictions.

5.3.3 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 between `NWC24_DL_PRIORITY_LIMIT` (128) and `NWC24_DL_PRIORITY_LOWEST` (255), with a default of `NWC24_DL_PRIORITY_DEFAULT` (192). Smaller values have a higher priority.

5.3.4 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).

5.3.5 File Name

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

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

5.3.6 Server Side Update Interval

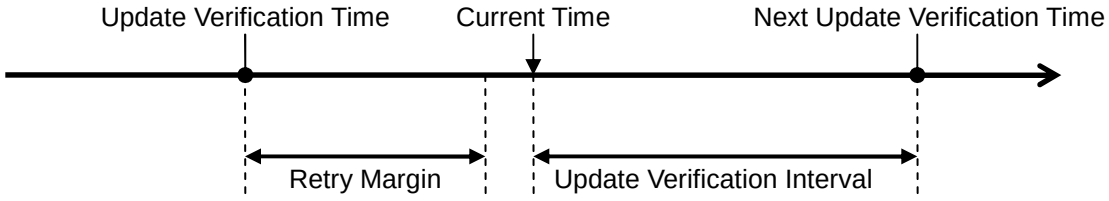
This specifies the interval by which the content specified in the URL is updated.

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

5.3.7 Retry Margin

When power to the Wii console is turned off or the NWC24 scheduler is stopped, each task's scheduled checks for updated content will no longer run. 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 5-1 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 developer does not have to specify the retry margin because the library automatically sets an appropriate value.

5.3.8 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`

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

- NWC24_DL_FLAG_USE_MYPUBLICKEY

  Verifies 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

  Decodes content using the secret key configured by the application.

- NWC24_DL_FLAG_GROUP_WRITABL

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

5.4 Configuration Items for Individual Applications

These are items configured separately for each application that handles download tasks. Each task contains information on the application that registered it, and this information can be used to reference individual application configuration items.

Because the items listed below are stored as files in each application's home directory (specifically, as NWC24 user files), it is necessary to call the NANDCheck function. These files can be created regardless of whether download tasks have been registered, so we recommend creating them along with save data when that is created.

5.4.1 Download Box

The application must prepare a VF archive to store pieces of content obtained by the tasks it has registered.

5.4.2 Public Keys and Shared Keys

Public keys are used to verify content signatures. Shared keys are used to decrypt encrypted content.

Although WiiConnect24 has public and shared keys in internal system regions that cannot be accessed, during normal operation each application should separately provide keys.

Application keys are stored by the NWC24SetDlKeys function in a file named wc24pubk.mod. This file uses 1 FS block and consumes 1 inode.

If a key has already been configured (there is already a key file), it will be overwritten with the new key. Each application can set only 1 public key and 1 shared key.

This file can be deleted by calling the NWC24ClearDlKeys function.

The key file is handled in the same way as save data. It will not be automatically deleted when a task disappears.

5.5 Using the Download API

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

5.5.1 Initializing the Download API

The NWC24 library must be opened, and the NWC24 system's operational state must be confirmed to use the *Download API*.

5.5.2 Obtaining Tasks

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

Code 5-1 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.

5.5.3 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.

The download box is an archive file handled by the VF library. It can be created and formatted and have its permissions changed by `NWC24CreateDIVf` in a single series of creation operations. Note that the specified file size includes space for the FAT region. The space that can actually be used is approximately 10 KB less than the specified file size.

Code 5-2 Example of Creating New Tasks

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

NWC24Err err;

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

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

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

char vfpath[64];
// Obtain the file name of the download box.
err = NWC24GetDlVfPathByTask(&mytask, vfpath, sizeof(vfpath));
if (err != NWC24_OK) {
   OSReport("NWC24GetDlVfPathByTask(): Error %d\n", err);
   OSHalt("Failed.\n");
}

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

Normally, when creating a download task, you must always specify the URL where the download content exists. The update verification interval should also be configured to a value that is appropriate for the content type. See the *NWC24 Library Function Reference* for details about the other settings.

Since the download box might have been deleted by the Wii 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.

5.5.4 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 5-3 Example of Setting the Remaining Download Count

```
#define MY_DLCOUNT 3

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

Code 5-3 sets the remaining download count to three. This process is executed regardless of whether the download task already existed or was new.

5.5.5 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 5-4 Example of Registering Tasks

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

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

5.5.6 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 function in Code 5-5 to delete a download task.

Code 5-5 Function to Delete a Download Task

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

5.5.7 Executing Downloads Immediately

Registered download tasks are automatically executed according to a controlled schedule. Although it is advisable to design applications to not be hindered if the most recent data is not always present, the following function can be used to immediately execute a desired download task if the most recent possible data is required.

Code 5-6 Function to Execute a Download Task

```
NWC24Err NWC24ExecDownloadTask( u32 operationFlags, u16 taskId, u32 subTaskMask );
```

This function should not be used any more often than can be avoided; it takes longer to process this than to use RVL DWC(-DL) download features or other such functionality. It also cannot be used if the scheduler has been stopped.

5.5.8 Getting the Errors Recorded for a Download Task

A download task can be executed even when the application that registered it is no longer running. In such cases it is impossible to know whether errors occurred when the download task was run.

Use the `NWC24GetDlError` function to check whether errors occurred when a download task was run. By using this function, it is possible to get the total number of errors that occurred as well as the error code of the last one to occur. However, information is available for server-related errors only. Access point connection failures and other errors that occur during network initialization are not recorded and thus cannot be known.

However, when the `NWC24ExecDownloadTask` function is used to immediately execute download tasks, even these network initialization error codes can be obtained. Because error information is also recorded in the task, information on server-related errors can also be obtained by using this method.

Error code information obtained in this way can be used to notify the user of server problems. Knowing this information is also useful to analyze behavior while debugging.

5.6 Precautions When Designing Receivable Content

5.6.1 How to Verify Content Updates from the Application

WiiConnect24 uses a method set by the HTTP 1.1 standard (the `If-Modified-Since` header) to check for updated server content and get as little duplicate content as possible. The time of the most recent update for previously received content is stored in the task list, so the update check will be properly performed even if the files in the download box are deleted.

Nevertheless, the user environment, proxy server settings, and other configurations might on rare occasions cause checks for updates using HTTP to be performed improperly and duplicate content to be obtained. You should design programs that will not fail when they receive the same content twice. The synchronization feature implemented by checking for updates should only be considered a measure for reducing the data volume transferred by the server.

Unless there is a particular reason to do otherwise, some kind of application-side screening for duplicate content is recommended. For example, embed sequence numbers in content and ignore any content that does not have a larger sequence number than what was previously received.

An embedded sequence number is recommended for the following example usage.

- Distribution of game items or other content that affects save data when it is obtained

  If files like these are processed by assuming all files in the download box are new data, blindly applying them to in-game content and then deleting them, it might be possible to acquire more than one valuable item.

An embedded sequence number is unnecessary for the following example usage.

- Distribution of ranking data or other content that is merely displayed once it is obtained

5.6.2 Independence from the Timing at Which Data Is Obtained

The time at which download tasks are executed might be delayed for various reasons. It is therefore safest to not assume that new content can be obtained at a specific time and to prevent the user from seeing any original dates and times within the distributed content.

Additionally, do not depend on the various download tasks executing at some relative time. For example, you cannot assume that tasks will continue to execute with approximately the same timing only because they check for updates at the same intervals and were registered at the same time. This is by design: to distribute network load, no more than a single task is run every two minutes; thus, tasks are prevented from running at overlapping times and will slowly become out of sync.

5.6.3 Preparing “Empty Content”

It is possible to remove files from the server or prohibit access to them to stop distributing certain content. However, this will cause update queries themselves to fail and then be processed internally as errors. This makes it impossible to use scheduling algorithms that use the most recent update time and prevents distinctions from being made between these errors and true errors.

It is therefore recommended to define “empty content” that will be ignored by the application if it is received, such as content that has only the control header used by the application, and stop distribution by using this “empty content” instead.

5.7 Precautions During Development

The settings and data contained in download tasks are stored in one of the following locations.

- Per-application save data directory (public keys, private keys, and download box)
- WiiConnect24 system region (items not mentioned above that are accessed by a dedicated API)

Although the NWC24 API maintains consistency among these files and storage, it might be lost due to specific operations during development, leading to states that were not intended by the developer. Be cautious when performing the following types of operations.

5.7.1 Changing the Development Environment

In addition to the registered application’s application ID, download tasks will internally maintain the home directory path; the values of each are decided when a new task is created. Access to download boxes and similar data is restricted using these values.

In this way, the download API assumes that each application has a different application ID and home directory. If the application ID or home directory changes in the process of developing a single application, this might cause operations that were unintended by the developer.

Specifically, caution must be exercised in the following situations.

- Switching between the Optical Disc Drive (DVD) library and the CNT library during NAND application development
- Rewriting the `GameCode` in the `DDF` file

In these cases, it is recommended to make a clean sweep by deleting all download tasks once.

5.7.2 Deleting Save Data

When save data is deleted within the DEVKIT system menu or Wii Menu versions prior to 3.0, download tasks will remain unchanged, but the keys and download boxes that they reference will no longer exist.

Beginning with version 3.0 of the Wii Menu, registered download tasks will also be searched for and deleted at the same time as save data. This behavior differs from the results of deleting data from the DEVKIT system menu.

6 Relationship Between the NWC24 API and the Wii Message Board

The Wii console's "Wii Message Board" is a standard feature of the Wii Menu.

The Wii Message Board is an application that comes pre-installed on the Wii console as a Wii Menu feature. Implemented using the WiiConnect24 system, it is a mechanism for exchanging messages between family members and with other friends' Wii consoles. Messages sent from applications can also be posted to the Wii Message Board.

This chapter explains methods for doing this.

6.1 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 directly to the local 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.

6.1.1 Message Format Requirements

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

Note: The following are requirements for displaying on the Wii Message Board. These requirements do not necessarily need to be met for unopened messages that are exchanged between applications without being displayed to the Wii Message Board.

- 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 6.3 Letterhead Template.)
- When letterhead data is not attached, the default letterhead is used.

6.1.2 Cautions When Posting Messages

Normally, specifications call for the sending player's nickname (defined in the Wii console's friend roster) to be displayed in the sender's field of messages displayed on the Wii Message Board. For this reason, even if a message is sent automatically, the nickname of the owner of the Wii console that originated it is used for the sender's field of the message when it is displayed. If a Wii console sends a message locally to itself, the sender's field will be empty.

To prevent this situation, a mechanism has been provided to display a different name (such as the game's name or an in-game character's name). 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 friend roster and is given precedence over the name set in the 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 that applications post to the Wii Message Board. This function can hide the reply button when messages are displayed in the Wii Message Board.

6.1.3 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.

6.1.4 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 can delay the display of a message on the local console's Wii Message Board. The range of time that can be specified for the delay is from 1 hour to 240 hours (10 days),and is 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 console with Wii Menu version 3.0 or later installed.

6.2 Timing When the Wii Message Board Processes Messages

The Wii Message Board is implemented as a part of the Wii 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 directly to the local console's Wii Message Board or received from an external source while the application is running are not reflected in the Wii Message Board until returning to the Wii Menu the next time.

6.3 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 they can also 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.

6.3.1 Composition of Letterhead Data

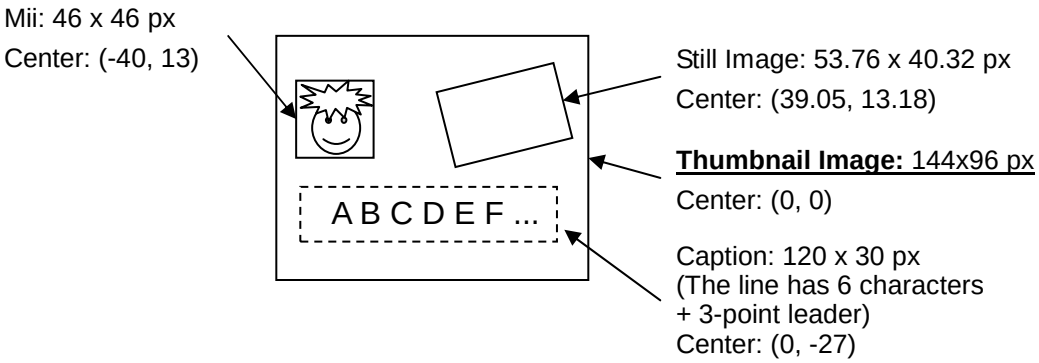
With the letterhead template, rendering is performed using a total of 10 images: one thumbnail image and nine images for enlarged display (three each of the header, body, and footer.) All the images are created in TPL format and with designated file names and folder structure.

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.

6.3.2 Thumbnail Image

The location of each part arranged in the thumbnail image to be displayed on the Wii Message Board is shown in Figure 6-1.

Figure 6-1 Thumbnail Image Part Distribution

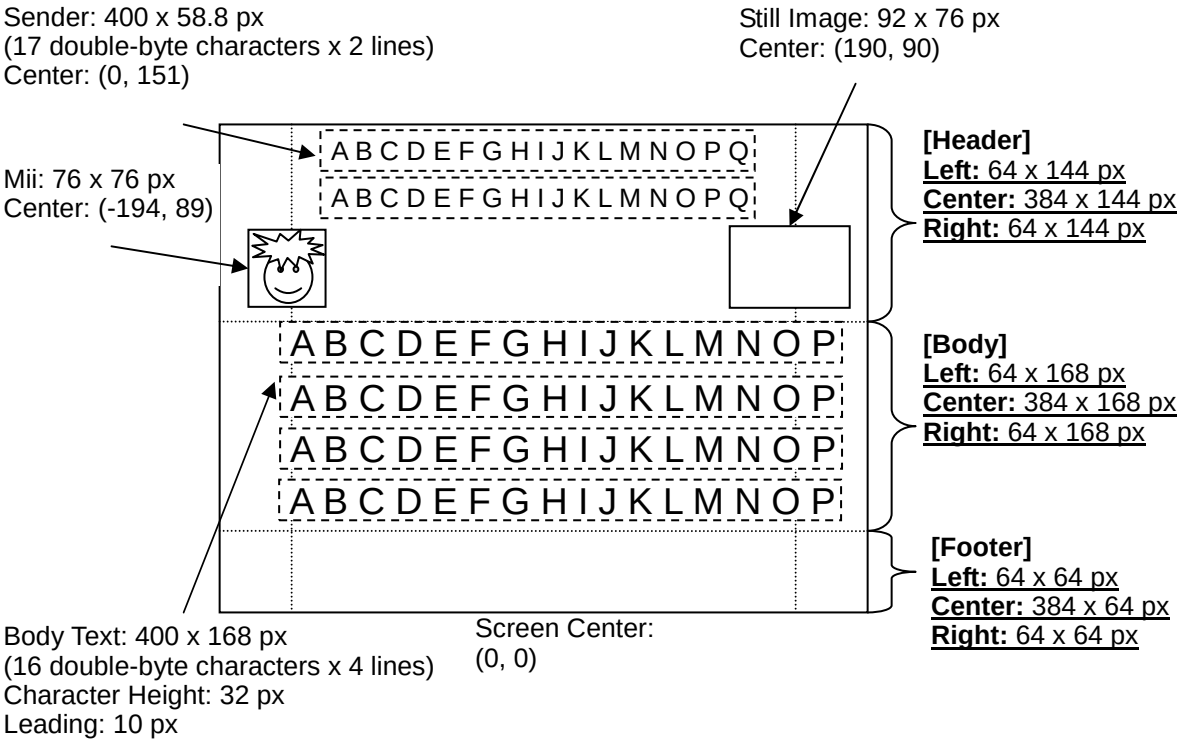


The caption portion displays the first 6 characters of the sender's name and a 3-point leader. (The nickname is used when the recipient is registered as a Wii Friend by the recipient. The alternate nickname has precedence when it is set with the `NWC24SetMsgAltName` function.)

6.3.3 Images for Enlarged Display

Figure 6-2 shows 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.

Figure 6-2 Part Distribution of the Image for Enlarged Display



The body text portion is displayed extending 8 pixels to the left and right of the body center part. When the body text exceeds 4 lines, the body portion is displayed in repeated 4-line units.

6.3.4 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 6.3.2 Thumbnail Image and 6.3.3 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.

For the texel format, use a format other than `GX_TF_RGBA8` that is supported by the layout library. 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

6.3.5 Creating Letterhead Data

First, create images for each part using `darchD.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` (27808 bytes) to `thumbnail_LZ.bin` (4580 bytes)

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

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

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 is attached to a message as letterhead data, the size of the letterhead template archive cannot exceed 120 KB. The `NWC24SetMsgAttached` function will return an error if this limit is exceeded.

6.3.6 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.

7 Friend Roster

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

The friend roster can be managed with the Wii address book, which 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.
- As a rule, prevents the Wii console from accepting messages from addresses that are not registered to the friend roster.

The *NWC24 Friend Roster API* provides the environment for accessing the friend roster.

7.1 Using the Friend Roster API

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

7.1.1 Initializing the Friend Roster API

To use the Friend Roster API, the NWC24 library must be opened.

7.1.2 Obtaining Friend Information

The following sample code (Code 7-1) stores only the friend information of those with whom a friend relationship is established to the list prepared from the specified item number.

Code 7-1 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));
    if( info == NULL ) return -1;
    // Obtains the number of friend information items that can be registered to the
    // Wii console's 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 console’s 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 `NWC24FriendInfo.attr.type` member. In particular, because messages to Wii Message Boards can only be sent to Wii Friends who have a registered Wii number, confirm this member value when obtaining friend information to select a send destination.

7.1.3 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. For details, see the NWC24 library in the *Revolution SDK Extensions Function Reference Manual*.

Code 7-2 Functions to Search for Friend Information

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

7.1.4 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). For details, see the NWC24 library in the *Revolution SDK Extensions Function Reference Manual*.

Code 7-3 Functions to Obtain the Number of Friend Information Items

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

7.2 Registering to the Friend Roster

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

7.3 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 "List of Characters That Can Be Input in the Address Book" in the NWC24 library in the *Revolution SDK Extensions Function Reference Manual* for the list of characters that can be entered in the address book.

8 Scheduler Operation

Even if an application is running, Wii consoles that have WiiConnect24 communication ON will automatically process communications—send and receive messages and download data—as needed in the background.

The fundamental concept of WiiConnect24 is that communications are performed periodically even while applications are running. Consequently, this periodic network processing is enabled (the scheduler is running) by default for applications that use the network.

8.1 Effect of Running the Scheduler

Although the scheduler starts each of the communication processes 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.

8.2 Scheduler Operation API

The scheduler operation API provides the following functions.

Code 8-1 Functions 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.

These functions also have an internal counter that allows nested calls to be made from one function to another. For example, if this feature is used and the `NWC24SuspendScheduler` function is called

twice while the scheduler is running, the `NWC24ResumeScheduler` function must be called twice for the scheduler to begin processing again.

For details on each function, see the NWC24 library in the *Revolution SDK Extensions Function Reference Manual*.

8.2.1 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. However, while the NWC24 library is open, the scheduler is always in a stopped state.

If scheduler operation functions are called while the NWC24 library is open, the scheduler's behavior will be affected by these functions only after the NWC24 library is closed.

8.2.2 Scheduler Initial State

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

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

8.3 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 Controls 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.

8.3.1 Start Timing for the 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.

8.3.2 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.

8.3.3 Startup Timing After Resuming the Scheduler

If the scheduler is in a stopped state when the startup time arrives, that startup is skipped. The next startup time will occur either when the startup interval has passed (this interval begins once the startup time was missed) or the moment that the scheduler is resumed.

Note: Processing will start immediately after the scheduler is resumed only with the firmware included in version 3.0 or later of the *Revolution SDK.*

8.4 Stopping the Scheduler

The scheduler cannot be stopped immediately while it is running (while the NWC24 firmware is in the middle of performing network communications). The application must wait until network processing has completed to stop the scheduler.

However, it is impossible to estimate how long of a wait is necessary. The reason is that these kinds of network processing depend on the network environment and the size of both the messages and the downloads being processed. However, it is extremely rare for 10 seconds or more to pass on a typical broadband connection.

If the scheduler takes some time to stop, the application’s response will strongly depend on the application’s specifications.

For example, if the scheduler is being stopped so the library can be opened for implicit use by the user, this whole operation can be omitted if the scheduler could not be stopped after waiting for a fixed period of time. On the other hand, if it is not possible to stop the scheduler within a fixed period of time for explicit use, it is polite to display a message prompting the user to retry the operation later.

Sometimes, it might be desirable to stop the scheduler to avoid affecting communication when the RVL DWC library is used to start peer-to-peer communications. In this case, peer-to-peer communication volume will dictate if communications should not be started until the scheduler has finished stopping or if communications should go ahead and be started while the scheduler is being stopped asynchronously in the background. Normally, the NWC24 firmware’s communication processing has an extremely small effect on an application's peer-to-peer communication.

9 Miscellaneous Information

9.1 Operational Environment on the Production Version

Some Wii consoles cannot use communication features when shipped.

For WiiConnect24 to operate, an environment with Wii 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 Wii 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 Wii Menu is a version older than 2.0.

9.2 Behavior Over a Broadband Connection

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

It is okay if WiiConnect24 features cannot operate over narrowband connections, such as through modems.

9.3 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.

9.4 Messages to Users 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 console’s friend roster.

9.5 Application-Specific Wii Message Names

It is okay 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.

9.6 Specifications Related to the Destination Region

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

For example, in cases where you want to limit communication partners to those in the same market,

the determination of how to handle any given Wii message and the filtering of Wii messages must be performed independently by the application. You may do this to accomplish goals such as the following:

- Communicate with users using only the same language
- Limit the character sets to be prepared to the minimum required level
- Limit communication partners to a specific physical area

There are a number of ways to filter messages. One possibility is to take the sender's game code from the sender's application ID, which is included in a Wii message's object information, and determine the destination region from that game code. Another possibility is to attach data to Wii messages that includes the sender's region information so that this information can be obtained from the attached data when it is received.

9.7 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.