Nintendo Wi-Fi Connection Revolution DWC Programming Manual

Version 1.3.2

Table of Contents

Code

Tables

Figures

Revision History

Version	Revision Date	Description
1.3.2	2007/06/29	• Added 3.1, DWC. • Added a description about time zones for the ranking management tool. • Added a description about proxy settings for the ranking management tool. • Added a description about obtaining higher and lower rankings. • Added a description about data transmission volume guides.
1.3.1	2007/05/11	Added a description about the thread created by `DWC_LoginAsync` to 5.1.
1.3.0	2007/04/27	• Added a description about the console friend link feature to 2.2.1, 2.2.3. • Support for argument changes for `DWC_InitFriendsMatch` and `DWC_InitLanMatch`. • Additions to "Table 11.3 Parameters Set with the Order Get Mode." • Additions to "11.2.4.3 Getting Communication Results: Ranking List Get Mode (Top, Nearby, Friend)." • Changed the maximum size of the user-defined data in "11.2.3 Uploading Scores" to 764 bytes. • Added support for Korea in the market specification of ranking.
1.2.3	2007/03/09	Revised a misprint in Table 11.2 (Incorrect: "near," correct: "nearby").
1.2.2	2007/02/22	• Corrected a misprint in which DWCRnkGetParam.nearby had appeared as "near." • Added a description about the base64 encoding method in the ranking CSV. • Added a description of the Top Ranking List Get Mode to "11.2.4.3 Getting Communication Results: Ranking List Get Mode (Top, Nearby, Friend)." • Corrected a misprinted file size in "12.5.3.1 New Registrations."
1.2.1	2007/01/18	• Added description about implementing ranking timeout. • Updated copyright notation.
1.2.0	2006/12/20	• Added information in "3.1 Initialization." • Revised the section regarding downloads.
1.1.0	2006/12/06	• Responded to changes in specifications to `DWC_Init` and `DWC_CreateUserData`. • Added description related to data transfer volume gauge.
1.0.0	2006/11/28	Regular Public Version.
0.0.4	2006/11/20	Added Chapter 2, "User Management when Using Revolution DWC."
0.0.3	2006/11/11	Made corrections to "8.3 Targets for the Buffer Size Specified by `DWC_InitFriendsMatch`."
0.0.2	2006/10/30	Revision following the integration of `DWC_SetMemFunc` to `DWC_Init`.
0.0.1	2006/08/25	Initial Version.

1 Introduction

Revolution DWC (hereafter referred to as DWC) is used to make Wii connections to Nintendo Wi-Fi Connection. DWC is available in two forms: the RVL DWC package, which includes all DWC functions, and the RVL DWC-DL package (DWC-DL in this document), which removes the functions associated with GameSpy servers and allows the use of only the download function.

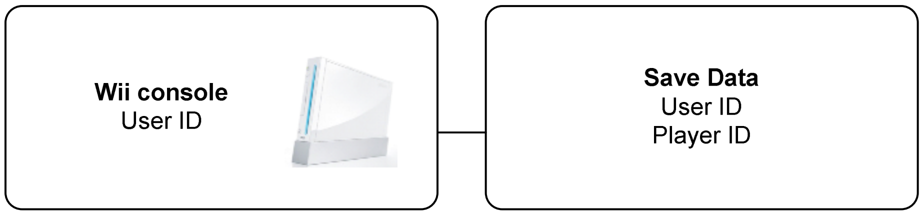
This document is shared by the RVL DWC and RVL DWC-DL packages and may include information not applicable to developers who use the RVL DWC-DL package.

2 Revolution DWC User Management

2.1 Managing Wi-Fi User Information

When Revolution DWC is used, user ID and Player ID are required for Nintendo Wi-Fi Connection authentication. This data is managed by pairing the Wii console with save data (see Figure 2-1).

Figure 2-1 Save Status of the User ID with the Wii Console and Save Data



- The User ID used for Nintendo Wi-Fi Connection authentication is saved on the Wii console.
- The User ID and Player ID used for Nintendo Wi-Fi Connection authentication are saved in the save data.

In contrast to Nitro DWC, it is not possible to move the save data to another Wii when when using titles compatible with the Wii's Nintendo Wi-Fi Connection. Furthermore, it is not possible to recreate or delete User IDs saved on the Wii console. No differences should arise between the User ID saved on the Wii console and the User ID saved in the save data.

2.1.1 User ID and Player ID

A User ID is issued by the authentication server the first time the user logs into Nintendo Wi-Fi Connection. This unique User ID is saved in the system region of the Wii console NAND memory.

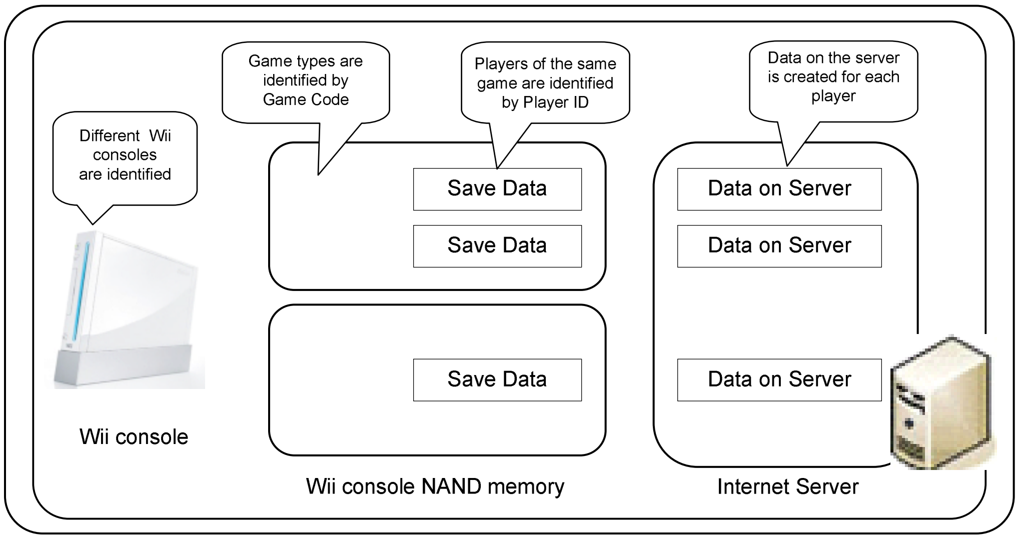
The Player ID is a 32-bit random number. Because the data on the Internet server is managed at the "User ID + Player ID + Game Code" level, no problem will arise as long as the Player ID is unique for the same User ID and the same Game Code. If duplicated, a Player ID that is not a duplicate will be assigned during authentication.

2.1.2 Differences between User ID and Player ID

Since the User ID is issued for the Wii console, all users who use the same Wii console will use a single User ID for many games.

Since the Player ID is issued for save data, different Player IDs are used for games that are on the same Wii console (User ID) and that have the same Game Code (see Figure 2-2).

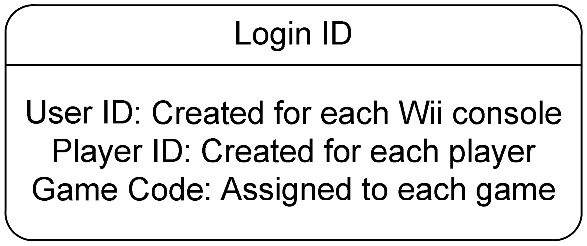
Figure 2-2 How Data is Maintained on the Internet


Different Wii consoles are identified
Game types are identified by Game Code
Players of the same game are identified by Player ID
Data on the server is created for each player
Save Data
Save Data
Save Data
Data on Server
Data on Server
Data on Server
Wii console
Wii console NAND memory
Internet Server

2.1.3 Player Information at the Game Level: Login ID

Taken together, the User ID, Player ID, and Game Code are collectively referred to as the *Login ID* (see Figure 2-3). Furthermore, user information saved on the Internet server is called a *profile*, and the ID used to manage profiles on the server is called the *Profile ID*.

Figure 2-3 Configuration of a Login ID


Login ID

User ID: Created for each Wii console
Player ID: Created for each player
Game Code: Assigned to each game

The Login ID or Profile ID is used to search for other user profiles on the Internet server from within DWC. Profile IDs are assigned to Login IDs with a one-to-one correspondence.

2.1.4 Information for Nintendo Wi-Fi Connection Authentication Saved by Games

Information used for Nintendo Wi-Fi Connection authentication must be saved in the Wii console NAND memory.

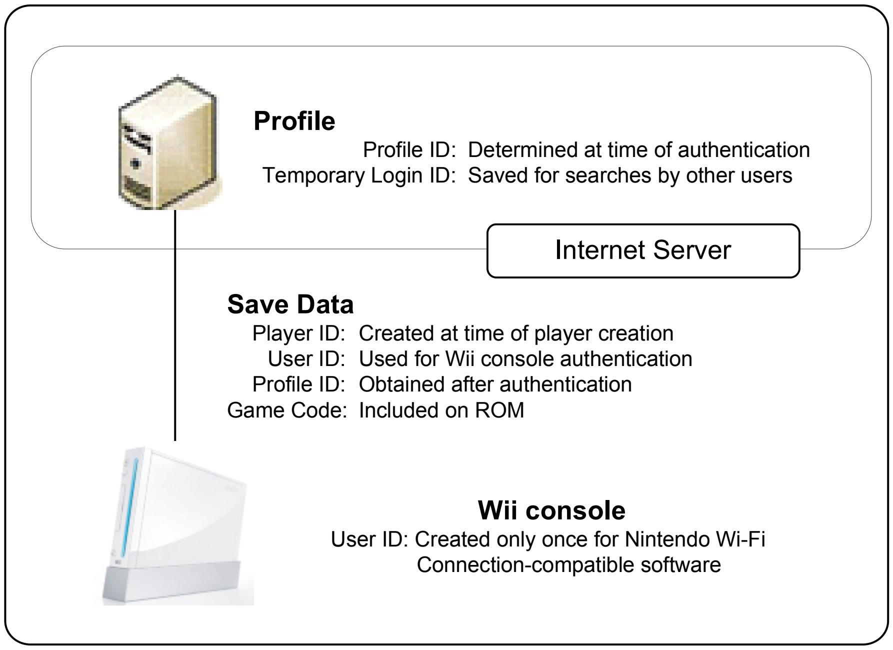
Note: The size of the information used for authentication is 64 bytes.

A temporary Login ID, pre-authenticated Login ID, and Profile ID are included in the information used for Nintendo Wi-Fi Connection authentication. Because this information is created and updated by DWC, the developer does not need to understand the details of what it contains.

Furthermore, if multiple players are allowed to save player data for a single title, information for Nintendo Wi-Fi Connection authentication must be saved for each player.

Terminology related to Nintendo Wi-Fi Connection authentication is shown in Figure 2-4.

Figure 2-4 Comprehensive Terminology for Nintendo Wi-Fi Connection Authentication


Profile
Profile ID: Determined at time of authentication
Temporary Login ID: Saved for searches by other users
Internet Server
Save Data
Player ID: Created at time of player creation
User ID: Used for Wii console authentication
Profile ID: Obtained after authentication
Game Code: Included on ROM
Wii console
User ID: Created only once for Nintendo Wi-Fi Connection-compatible software

2.2 Friend Management Overview

2.2.1 Constructing Friend Relationships

Friendships are formed on the Internet server so that communications with friends can be started easily when DWC is used. Friendships are formed by exchanging user information. Established friendships are saved in each user’s Profile.

The information that can identify a partner user for entering into a friend relationship is called a “friend code.” The exchange of these friend codes provides a mechanism for creating a friend relationship (see Figure 2-5):

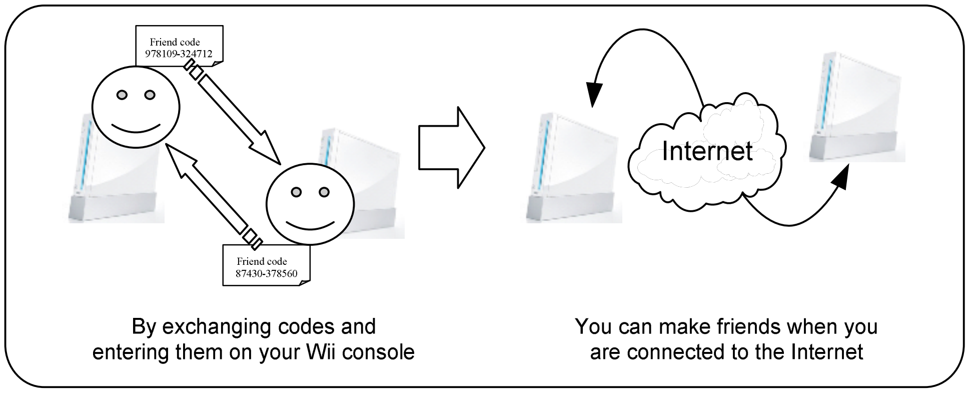
Direct exchange of friend codes by fellow users: friend codes are exchanged by sharing them with each other by phone or other means.

Linking with Wii Friends: friend codes are exchanged through Wii messages with a friend who is already a Wii friend.

Friend codes can be created by DWC, which includes a feature for automatically creating the most appropriate information based on that used for Nintendo Wi-Fi Connection authentication saved in the save data.

Problems may arise if the friend code is too long to be easily used by people. For this reason, the Profile ID obtained when you log on to the Internet for the first time is used to create the friend code rather than the Login ID.

Figure 2-5 Creating Friend Relationships with Friend Codes


Friend code 978109-324712
Friend code 87430-378560
Internet
By exchanging codes and entering them on your Wii console
You can make friends when you are connected to the Internet

A friend code is expressed as a 12-digit number. Make note of the following items:

- A user interface for issuing friend codes must be created. Since friend codes cannot be issued to users who have never connected to the Internet, it is necessary to display a message.
- A user interface for entering friend codes must be created. This user interface must save the entered information and allow it to be edited, as necessary, to handle the possibility that the friend code is incorrect.

2.2.2 Constructing Friend Relationships with Friend Codes

For the maximum number of players to be managed by the game, the exchanged friend information must be saved in the Wii console NAND memory. This allows a user to edit friendships without connecting to the Internet. It is necessary to save this data along with friend-related information managed by the game itself (nicknames and versus scores). DWC handles this as friend information without any awareness of the Login ID, Profile ID or type of friend code.

Note: Twelve bytes are required per person in order to save friend information.

2.2.3 Linking Wii Friends and Game Friends

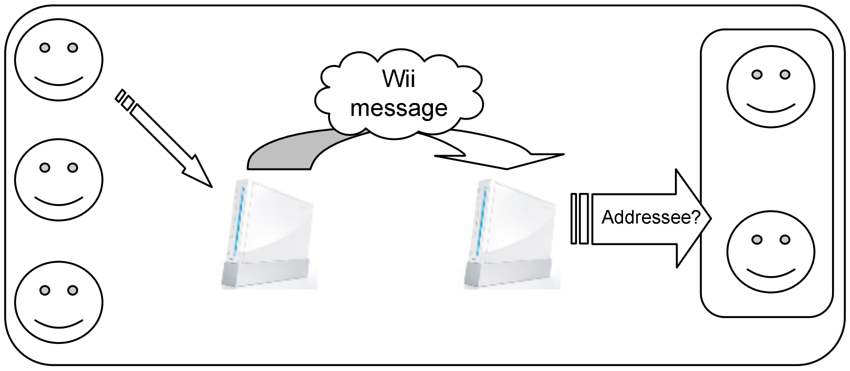
Inviting friends with whom a Wii friend relationship has already been established to be game friends not only alleviates the trouble of inputting friend codes, but is also effective for increasing new game users. However, because having two types of friends (Wii friends and game friends) may confuse users, it is preferable for developers to have a grasp of their various features and design a mechanism that is as seamless as possible.

Keep the following points in mind when exchanging friend codes through Wii messages.

- There is no way of knowing which user of the partner's Wii will see the sent messages
    - ✓ The recipient of a Wii message is a single Wii console, but there may be multiple users of that Wii.
    - ✓ A sentence such as "Hi [friend name], it's [user name]. Would you like to play a Wi-Fi game with me?" must be included within the message to determine the Wii user to whom the friend request is being made.
    - ✓ Recipients cannot be specified even for Wii messages that cannot be posted on the Wii Message Board or that can only be seen by game programs.
- It will take time for the Wii message to arrive

  Some time is required from the moment a Wii message is sent until it is received. Consequently, even if both parties are online, the exchange of friend codes through Wii messages does not happen in real time. For example, even if two users exchange friend codes through Wii messages while confirming on the phone, the requests will not be applied right away. Thus, building friend relationships does not happen at the very moment that friend codes are mutually entered.

Figure 2-6 Wii Messages Addressed to Wii Consoles Rather than to Friends



3 Initializing and Shutting Down DWC

3.1 Items to be Considered when Using DWC

Be aware of the following when using DWC:

- The API is not thread-safe

  Because DWC functions were not designed to be thread-safe, please perform appropriate exclusive processing so that the functions are not simultaneously called from multiple threads.

- The API may block for extended periods of time

  DWC functions perform communications via the internally, using sockets. For this reason, blocking may occur for an extended period of time depending on the communication environment and the IO processor load. Even functions that include “Async” in the name require time for the asynchronous processing to start. It is thus best to call DWC functions from threads that do not impact screen rendering.

3.2 Initialization

The DWC software library must be initialized before any DWC functions are called.

Initialize the DWC library using the `DWC_Init` function, which carries out the following processes:

- Initialization of the entire library
- Selection of the authentication server
- Initialization the DNS cache
- Configuration of the memory allocation/deallocation function

This is a blocking function.

When DNS cache is initialized, a DNS query of the servers that may be connected by the DWC is performed first.

DWC needs about 230 KB of memory when performing matchmaking for four persons. Every time the maximum number of persons for matchmaking is decreased by one, the amount of necessary memory decreases by approximately 20 KB (when the `sendBufSize` and `recvBufSize` arguments of the `DWC_InitFriendsMatch` function are each at the default of 8 KB).

The memory allocation/deallocation function specified here must have exclusive access control for the threads. The memory allocated may be either MEM1 or MEM2.

Code 3-1 DWC Initialization

```
void init_dwc( void )
{
   // DWC Initialization
   DWC_Init( DWC_AUTHSERVER_DEBUG
          “gamename”,    // game name
          ‘code’,        // initial code
          AllocFunc, FreeFunc );
   :
}

 void* AllocFunc( DWCAllocType name, u32 size, int align )
{
   void* ptr = NULL;

  (void)name;
  OSLockMutex( &s_mutex );
  ptr = MEMAllocFromExpHeapEx( s_hmem2heap, size, align);
  OSUnlockMutex( &s_mutex );

  return ptr;
}

void FreeFunc( DWCAllocType name, void* ptr, u32 size )
{
  (void)name;
  (void)size;

  // Allow deallocation request to an unallocated market (NULL pointer)
  if( ptr != NULL )
  {
      OSLockMutex( &s_mutex );
      MEMFreeToExpHeap( s_hmem2heap, ptr);
      OSUnlockMutex( &s_mutex );
  }
}
```

3.3 Shutdown

To shut down the use of DWC, be sure to call the `DWC_Shutdown` function. Calling `DWC_Shutdown` frees the memory used for the DNS cache.

The memory allocator passed to `DWC_Init` can also be freed.

To use DWC again, you must start by calling `DWC_Init`.

Code 3-2 Shutting Down DWC

```
void shutdown_dwc( void )
{
   // Initialize DWC
   DWC_Shutdown();
   :
}
```

4 Creating User Data

DWC performs the following typical functions based on user data.

- User authentication
- Creation of a friend relationship

Create user data with the `DWC_CreateUserData` function and save it as saved data when the user data either has not yet been created or is corrupted.

Allocate the memory to store the `DWCUserData` structure at the application level. When a single game supports multiple players, user data is needed for each player. When user data exists, use the `DWC_CheckUserData` function after loading the user data into memory to confirm its validity (see Code 4-1).

Code 4-1 Creating User Data

```
BOOL create_userdata( void )
{
   // Read Data from NAND
   LoadUserdataFromNAND( 0, &s_PlayerInfo, sizeof(s_PlayerInfo) );

   printf("Load From Backup\n");

   if ( DWC_CheckUserData( &s_PlayerInfo.userData ) )
   {
      DWC_ReportUserData( &s_PlayerInfo.userData );
      return TRUE;
   }

   // If Not Saved as Valid User Data
   printf("no Backup UserData\n");

   // Create User Data
   DWC_CreateUserData( &s_PlayerInfo.userData );

   printf("Create UserData.\n");
   DWC_ReportUserData( &s_PlayerInfo.userData );

   return FALSE;
}
```

Use the `DWC_CheckDirtyFlag` function to determine whether it is necessary to save the user data to saved data. Always clear `DirtyFlag` with the `DWC_ClearDirtyFlag` function prior to saving the user data (see Code 4-2).

Code 4-2 Saving User Data

```
void check_and_save_userdata( void )
{
    if ( DWC_CheckDirtyFlag( &s_PlayerInfo.userData ) )
    {
        DWC_ClearDirtyFlag( &s_PlayerInfo.userData );

        // Save User Data to NAND
        SaveUserdataToNAND( 0, &s_PlayerInfo.userData, sizeof(DWCUserData) );
    }
}
```

Use the following techniques to confirm the user data prior to connecting to the Internet:

- Use the `DWC_CheckHasProfile` function to confirm that the user data has connected to the Internet to get a profile. If a profile has not been obtained, the user data is updated and handled as a set by the Wii console.

Note: Review the flowchart included in the *Nintendo Wi-Fi Connection Programming Guidelines* for more details.

5 Connection Processes

The first time an Internet connection is established, a user ID is issued for the Wii console from the Nintendo authentication server. This user ID is stored in the Wii console NAND memory. Subsequently, each time a connection is made with the server, the user and player IDs are saved to the user data created by DWC, and a profile is created. The GS profile ID that corresponds to the created profile is also stored in the user data.

5.1 Connecting to the Nintendo Wi-Fi Connection Server

When connecting to the matchmaking server, use the `DWC_InitFriendsMatch` function to initialize the matchmaking and friend-related functionality (see Code 5-1).

The following are passed as arguments to `DWC_InitFriendsMatch`:

- User data
- Product ID provided by GameSpy
- Game name provided by GameSpy
- Secret key provided by GameSpy
- Size of the send/receive buffer for peer communications
- Wii friend roster
- Maximum number of elements for the friend roster

For more information about the send/receive buffer size, see Chapter 8, Sending and Receiving Data. When zero is specified, as in Code 5-1, a default of 8 KB is used.

The Wii friend roster is an array of friend information in the `DWCFriendData` structure. For details about the Wii friend roster and friend information, see Chapter 6, Sending and Receiving Data.

Use the `DWC_LoginAsync` function to connect to the server (see Code 5-1).

Note: The `DWC_LoginAsync` function internally creates a thread (priority 17) for communicating with the authentication server. This thread must be given processing time.

The first argument of this function is the screen name of the game. Specify a player name if it is used in the game. The game screen name is sent to the authentication server to be checked for validity. You can get the results of that check with the `DWC_GetIngamesnCheckResult` function (see Code 5-1).

Because the second argument is not currently used, pass a null to it. The remaining arguments are the login completion callback and its parameters.

After calling the `DWC_GetIngamesnCheckResult` function, call the `DWC_ProcessFriendsMatch` function in the order of each game frame to advance the login process (see Code 5-1).

The `DWC_ProcessFriendsMatch` function runs all of the communication processes associated with matchmaking and friend relationships until the `DWC_ShutdownFriendsMatch` function is called.

After login is complete, even when there are no explicit network processes for the application before connecting to another host, call that function anyway in case the Wii friend roster has been updated.

Code 5-1 Connecting to the Nintendo Wi-Fi Connection Server

```
static BOOL s_logined = FALSE;

void connect_to_wifi_connection( void )
{
    DWC_InitFriendsMatch(  NULL, DTUD_GetUserData(),
                           GAME_PRODUCTID, GAME_NAME, GAME_SECRET_KEY,
                           0, 0,
                           DTUD_GetFriendList(), FRIEND_LIST_LEN );
    // login using the authentication function
    s_logined = FALSE;
    if ( !DWC_LoginAsync( L”NAME”, NULL, cb_login, NULL ) )
    {
        // Failure during connection start.
        return;
    }

    // Connection completion polling
    while ( !s_logined )
    {
        DWC_ProcessFriendsMatch();

        if ( DWC_GetLastErrorEx( NULL, NULL ) )
        {
            // Error occurred.
            handle_error();
            return;
        }

        GameWaitVBlankIntr();
    }

    // Connection completed
    if ( DWC_GetIngamesnCheckResult() == DWC_INGAMESN_INVALID )
    {
        // Special processing for when an inappropriate screen name within a game is detected
        disp_ingamesn_warning();
    }
    :
}

// Login completion callback
void cb_login( void )
{
   if (error == DWC_ERROR_NONE)
   {
      check_and_save_userdata();
      s_logined = TRUE;
   }
}
```

The `DWC_ShutdownFriendsMatch` function ends matchmaking and friend relationship functionality and deallocates the memory allocated in the library.

The first time the user connects to the server with the user data configured with the `DWC_InitFriendsMatch` function, the Wii console and the user data are handled as a pair.

In addition, the user data is always updated when connecting to the server for the first time. Applications should use a login callback and, when the login is complete, check for the user data updates with `DWC_CheckDirtyFlag`. The application should be designed to take care of any user data that needs to be saved.

6 Creating Friend Rosters and Friend Information

The following procedures are available in DWC to create friend relationships among players.

- Using friend codes
  - ✓ Friend codes, which are Profile IDs that include error-checking information, are exchanged.
  - ✓ To use a Profile ID, the user must have connected to the Internet once.
- Linking with Wii friends

6.1 Exchanging Friend Codes

Any player who has connected (even once) to the Nintendo Wi-Fi Connection server is assigned a unique GS profile ID, which is saved in the user data.

Players with a GS profile ID can create a friend registration key (friend code) that adds error checking information to their GS profile ID (see Code 6-1). This code is expressed as a 12-digit decimal number. Exchanging the friend code with other players makes it easier to exchange friend data.

Exchange friend information with the input friend code using the `DWC_CreateFriendKeyToken` function, and save it in the friend roster (see Code 6-1).

The validity of the Friend Code input can be checked with the `DWC_CheckFriendKey` function, but because mistakes are possible, prepare a user interface that allows for multiple corrections (see Code 6-1).

Code 6-1 Exchanging Friend Codes

```
// Display the Friend Code
void disp_friend_key( void )
{
   u64 friend_key;

   // Create a Friend Code from your own user data
   if ( ( friend_key = DWC_CreateFriendKey( &s_userData ) ) != 0 )
   {
      // Display the Friend Code
      disp_message( "FRIEND CODE : %lld", friend_key );
   }
   else
   {
      // Displayed when there is no friend code
      disp_message( "FRIEND CODE : not available" );
   }
   :
}
```

```
// Create friend information from the Friend Code, and register to the friend roster
BOOL register_friend_key( void )
{
   u64  friend_key;
   DWCFriendData friendData;

   while ( 1 )
   {
      char friend_key_string[ 13 ];

      // Have the user manually input the Friend Code
      input_friend_key( friend_key_string );

      // Convert the input Friend Code string into a u64 value
      friend_key = charToU64( friend_key_string );

      // Check the validity of the friend code, and proceed if correct
      // If in error, display a message and prompt for correction
      if ( DWC_CheckFriendKey( s_userData, friend_key ) ) break;
      else disp_warning_message();
   }

   // Create friend information from a valid Friend Code
   DWC_CreateFriendKeyToken( &friendData, friend_key );

   {
      int index;
      // As with MP communication, search for openings or breaks in the friend roster
      // and register the friend information
      :
      s_friendList[ index ] = friendData;
      :
   }
}
```

6.2 Synchronizing Friend Rosters

To activate the Wii friend roster maintained by the application (hereafter, the local Wii friend roster) online, you must call the `DWC_UpdateServersAsync` function and update the Wii friend roster stored on the GameSpy server (hereafter, the server Wii friend roster) (see Code 6-2).

The login process performed by `DWC_LoginAsync` must be completed before performing this synchronization.

This function has the following arguments:

- Player name (from a past specification, specify a null here)
- Callback (when the Wii friend roster synchronization is complete) with its parameters
- Callback for the friend status change notification (described below) with its parameters
- Callback for the Wii friend roster deletion and its parameters

Synchronization of the Wii friend rosters works primarily by

- Sending requests for the new friend relationships that are present in the local Wii friend roster but not in the server Wii friend roster
- Deleting the friend relationships that are found in the server Wii friend roster but not in the local Wii friend roster

Even if the other party is offline, the request to create a friend relationship is saved on the GameSpy server and is delivered when the other party completes the next login using the `DWC_LoginAsync` function. The friend relationship is established only when the other party has the relationship initiator's information in the local Wii friend roster.

However, this only serves to register the other party as a friend. The other party receiving a request to create a friend relationship follows the same process to register the relationship initiator as a friend.

The Wii friend roster synchronization completion callback is called when both the local and the server Wii friend rosters are completely checked, necessary requests for friend relationship creation are sent, and any superfluous friend relationships are deleted. Just because the callback has been returned does not mean that a friend relationship has been established.

When the Wii friend roster synchronization completion callback `isChanged` argument is set to TRUE, it indicates that the friend information in the local Wii friend roster has been updated and that the local Wii friend roster must be saved. If other friend relationships have been established outside of the synchronization process, the friend relationship establishment callback set in the `DWC_SetBuddyFriendCallback` function is called.

In addition, if duplicate friend information for the same friend is found during Wii friend roster synchronization, duplicate information is automatically deleted, leaving a single entry. Whenever a deletion occurs, the index of the friend information's Wii friend roster that was deleted and the index of the friend with the same information are passed as arguments to the callback.

Code 6-2 Synchronizing Friend Rosters

```
BOOL s_update     = FALSE;
BOOL s_updateFriendList = FALSE;

void sync_friend_list( void )
{
   // Configure the friend relationship establishment callback
   DWC_SetBuddyFriendCallback( cb_buddyFriend, NULL );
```

```
// Synchronization of local Wii friend roster and server Wii friend roster
if ( !DWC_UpdateServersAsync( NULL,
                              cb_updateServers, NULL,
                              NULL, NULL,
                              cb_deleteFriend, NULL ) )
{
   // Synchronization failed to start
   return;
}

while ( !s_update )
{
   DWC_ProcessFriendsMatch();

   if ( DWC_GetLastErrorEx( NULL, NULL ) )
   {
      // An error occurred.
      handle_error();
      return;
   }

   GameWaitVBlankIntr();
}
:

while ( 1 )
{
   DWC_ProcessFriendsMatch();

   if ( DWC_GetLastErrorEx( NULL, NULL ) )
   {
      // An error occurred.
      handle_error();
      return;
   }

   // Because the Wii friend roster is updated irregularly, perform the following processes
   // when appropriate, and save the updated local Wii friend roster all at once.
   if ( s_updateFriendList )
   {
      // Save if the Wii friend roster has been updated
      s_updateFriendList = FALSE;
      save_friendList();
```

```
        }

        game_loop();

        GameWaitVBlankIntr();
    }
    :
}

// Friends list synchronization completion callback
void cb_updateServers( DWCError error, BOOL isChanged, void* param )
{
    if ( error == DWC_ERROR_NONE )
    {
        // Friends list synchronization successful
        s_update = TRUE;

        // If the Wii friend roster has been updated it must be saved
        if ( isChanged ) s_updateFriendList = TRUE;
    }
}

// Friends list deletion callback
void cb_deleteFriend( int deletedIndex, int srcIndex, void* param )
{
    printf( "friend[%d] was deleted (equal friend[%d]).\n",
               deletedIndex, srcIndex );
    s_updateFriendList = TRUE;
}

// Friend relationship establishment callback
void cb_buddyFriend( int index, void* param )
{
    printf( "Got friendship with friend[%d].\n", index );
    s_updateFriendList = TRUE;
}
```

6.3 Friend Information Types

The `DWC_GetFriendDataType` function gets the data types for the friend information; see Code 6-3. The data types are as follows.

• `DWC_FRIENDDATA_NODATA`:	No friend information is stored
• `DWC_FRIENDDATA_LOGIN_ID`:	An ID not yet connected

• `DWC_FRIENDDATA_FRIEND_KEY`:	The friend code
• `DWC_FRIENDDATA_GS_PROFILE_ID`:	The GS profile ID

Subsequently, once the other party has retrieved a GS profile ID and the Wii friend roster synchronization is completed, the data type changes to `DWC_FRIENDDATA_GS_PROFILE_ID`.

When the data type is `DWC_FRIENDDATA_FRIEND_KEY`, the GS profile ID registered with the friend code indicates that no friend relationships have been established. The data type then changes to `DWC_FRIENDDATA_GS_PROFILE_ID`.

The `DWC_IsBuddyFriendData` function can be used to determine whether a friend relationship will be established from the friend information (see Code 6-3).

Code 6-3 Getting Friend Information Types

```
void disp_friendList( void )
{
   int i;

   for ( i = 0; i < FRIEND_LIST_LEN; ++i )
   {
      // Get the friend data type
      int type = DWC_GetFriendDataType( &s_friendList[ i ] );
      printf( "friend[%d] type %d.\n", type );

      if ( type == DWC_FRIENDDATA_GS_PROFILE_ID )
      {
         // Display the friend relationship for a GS profile ID
         if ( DWC_IsBuddyFriendData( &s_friendList[ i ] ) )
         {
            printf( "Friendship is established.\n" );
         }
         else
         {
            printf( "Friendship is not yet established.\n" );
         }
      }
   }
   :
}
```

6.4 Getting Friend Status

Players participating in Nintendo Wi-Fi Connection store their own status, and the GameSpy server manages it. There are two types of player status that can be referenced by an application.

- Communication status
- A status string or binary data

Communication status is defined by the `DWC_STATUS_*` constant and is automatically set by DWC.

The status string, or binary data, can be set by the application using the `DWC_SetOwnStatusString` or `DWC_SetOwnStatusData` functions (see Code 6-4).

Any string that can be set for status must be null-terminated and can only be 256 characters in length, including the null termination. Binary data is converted within the function to a string with a data size that is roughly 50% larger in terms of character count.

Do not use delimiter characters such as “`/`” or ”`\\`”, because they are used by the library.

Once a friend relationship is established, you can get a friend’s current status. If the friend status change notification callback is specified as an argument in the `DWC_UpdateServersAsync` function, that callback is called every time the friend’s status changes, notifying you of that status (see Code 6-4).

For getting friend status, a set of functions is also available: `DWC_GetFriendStatus*` (see Code 6-4).

Because the functions in this set access the friend status list maintained by DWC, no communication occurs. However, because they require several hundred microseconds of processing time, be careful when calling them several times over a short period.

If the power is suddenly lost while the player status is being communicated, the previous status will remain for several minutes.

Code 6-4 Getting Friend Status

```
void sync_friend_list( void )
{
   int i;

   // Synchronization of the local and server Wii friend rosters
   if ( !DWC_UpdateServersAsync( NULL,
                                 cb_updateServers, NULL,
                                 cb_friendStatus, NULL,
                                 NULL, NULL ) )
   {
      // Synchronization failed to start
      return;
   }
   :
    // Friends list synchronization completed
    :

    // Set your own status string
    DWC_SetOwnStatusString( "location=city,level=1" );
    :

    for ( i = 0; i < FRIEND_LIST_LEN; ++i )
    {
        if ( DWC_IsValidFriendData( &friendList[ i ] )
        {
            u8    status;
            char* statusString;

            // Get friend status from valid friend information
            status = DWC_GetFriendStatus( &friendList[ i ], statusString );

            // Display the friend status
            disp_friend_status( status, statusString );
        }
    }
    :
}

// Friend status change notification callback
void cb_friendStatus( int index, u8 status, const char* statusString, void* param )
{
    printf( "Friend[%d] status -> %d (statusString : %s).\n",
                  index, status, statusString );
}
```

6.5 Friend Codes Notification using WiiConnect24

You can perform an exchange of friend codes in an application by embedding them in WiiConnect24 messages. This helps the exchange of friend codes between users who have formed a Wii console friend relationship. For details about the methods of sending and receiving, refer to the WiiConnect24 manual.

To embed one’s friend code in a WiiConnect24 message, use the `DWC_CfSetAppFriendKeyToNWC24Msg` function (Code 6-5). Calling this function will dynamically allocate the memory internally, but when the `DWC_Shutdown` function is called, this memory will be released automatically. If you want to explicitly release the memory, call the `DWC_CfReset` function.

However, you must release the memory only after the `NWC24CommitMsg` function is called for the `NWC24MsgObj` that has added information for exchanging game friends.

Code 6-5 Embedding friend codes

```
BOOL PostFriendMsg( void )
{
    NWC24Err       err;
    NWC24MsgObj    msgObj;

    // Initialize the message to the application
    err = NWC24InitMsgObj(&msgObj, NWC24_MSGTYPE_WII_APP);
    if ( err != NWC24_OK )
    {
        DWCDemoPrintf("NWC24InitMsgObj(): error %d\n", err);
        return FALSE;
    }

    // Set the recipient
    err = NWC24SetMsgToId(&msgObj, TestIdTo);
    if ( err != NWC24_OK )
    {
        DWCDemoPrintf("NWC24SetMsgToId(): error %d\n", err);
        return FALSE;
    }

    // Set the body text
    err = NWC24SetMsgText(&msgObj, “Hello”, (u32)strlen(TestMsgText),
                          NWC24_US_ASCII, NWC24_ENC_7BIT);
    if ( err != NWC24_OK )
    {
        DWCDemoPrintf("NWC24SetMsgText(): error %d\n", err);
        return FALSE;
    }

    // Embed the friend code in the message
    if(DWC_CfSetAppFriendKeyToNWC24Msg(&msgObj, &s_userData,
        DWC_CF_MSG_TYPE_REQUEST) != DWC_CF_ERROR_NONE)
    {
        DWCDemoPrintf("DWC_CfSetAppFriendKeyToNWC24Msg(): error %d\n", err);
        return FALSE;
    }
    // Store the message in the outbox
    err = NWC24CommitMsg(&msgObj);
    if ( err != NWC24_OK )
    {
        DWCDemoPrintf("NWC24CommitMsg: error %d\n", err);
        return FALSE;
    }

    DWCDemoPrintf("Posted a test message successfully.\n");

    return TRUE;
}
```

Use the `DWC_CfGetAppFriendKeyFromNWC24Msg` function to check whether a friend code is embedded in a received message. If a friend code has been embedded, it will be stored in the variable provided to the argument. (Code 6-6)

Code 6-6 Checking messages

```
BOOL SearchFriendMsgs( void )
{
    NWC24Err    err;
    u32         iObj;
    u32         numStored;
    u32         numRemain;
    u64         appFriendKey;
    u32         msgId;
    DWCCfMsgType msgType;
    NWC24MsgObj msgObjArray[MSGOBJ_ARRAY_SIZE];

    // Set the search conditions
    NWC24InitSearchConds();
    NWC24SetSearchCondMsgBox(NWC24_RECV_BOX);

    do
    {
        // Search for messages
        err = NWC24SearchMsgs( msgObjArray, MSGOBJ_ARRAY_SIZE,
            &numStored, &numRemain );
        if ( err != NWC24_OK )
        {
            DWCDemoPrintf( "NWC24SearchMsgs(): Error %d\n", err );
            return FALSE;
        }
        DWCDemoPrintf("[NWC24SearchMsgs(): Stored: %d Remain: %d]\n",
            numStored, numRemain);

        for ( iObj = 0 ; iObj < numStored ; ++iObj )
        {
            // Check the messages
            if( DWC_CfGetAppFriendKeyFromNWC24Msg( &msgObjArray[iObj],
                &s_playerinfo.userdata, &appFriendKey, &msgType ) == DWC_CF_ERROR_NONE)
            {
                ViewMessage( &msgObjArray[iObj] );    // Display the message
                DWCDemoPrintf("Received appFriendKey: %012llu msgType: %d.\n",
appFriendKey, msgType);
            }
        }
    }
    while ( numRemain > 0 );
    // Repeat until all messages that meet the search conditions have been found

    DWCDemoPrintf("Received message successfully.\n");
    return TRUE;
}
```

7 Matchmaking

DWC offers two types of structures for matchmaking: peer matchmaking and client/server matchmaking.

Peer matchmaking performs matchmaking without dividing Wii consoles into servers and clients, and is further classified into two types.

- Unspecified peer matchmaking (without friend specification)
- Friend-specified peer matchmaking (with friend specification)

7.1 Peer Matchmaking Without Friend Specification

Matchmaking is performed against an unspecified number of players.

To begin peer matchmaking without friend specification, use the `DWC_ConnectToAnybodyAsync` function (see Code 7-1). The following items are passed to this function as arguments:

- The desired number of connections, including the user
- A filter string used to include matchmaking conditions
- The matchmaking completion callback and its parameters
- The player evaluation callback (described below) and its parameters

The filter string is used to filter players to be considered as matchmaking candidates. The matchmaking indicator keys used in this filter string (the keys named `str_key` and `int_key` in the example below) must be registered in advance, using the `DWC_AddMatchKey*` functions (see Code 7-1). Key names are saved in the library. However, only pointers to key values are saved in the library, so the keys must be saved until matchmaking is complete. For examples of text that cannot be used for key names, see 7.6, Key Names that Cannot be Used for Matchmaking Indicator Keys.

Code 7-1 Peer Matchmaking Without Friend Specification

```
static BOOL s_matched  = FALSE;
static BOOL s_canceled = FALSE;
static const char* s_str_key = "anymatch_test";
static const int s_int_key = 10;

void do_anybody_match( void )
{
   // Set the matchmaking indicator keys
   DWC_AddMatchKeyString( 0, "str_key", s_str_key );
   DWC_AddMatchKeyInt( 0, "int_key", &s_int_key );

   // Begin matchmaking without friend specification
   DWC_ConnectToAnybodyAsync( 4,
                              "str_key = 'anymatch_test' and int_key = 10",
                              cb_anymatch, NULL,
                              NULL, NULL );

   // Matchmaking completion polling
   while ( !s_matched )
   {
      DWC_ProcessFriendsMatch();

      if ( DWC_GetLastErrorEx( NULL, NULL ) )
      {
         // An error occurred.
         handle_error();
         return;
      }

      GameWaitVBlankIntr();
   }

   // Matchmaking completion
   :
}

// Matchmaking completion callback
void cb_anymatch( DWCError error, BOOL cancel, void* param )
{
   if ( error == DWC_ERROR_NONE )
   {
      if ( cancel ) s_canceled = TRUE;
      else          s_matched  = TRUE;
   }
}
```

7.2 Peer Matchmaking With Friend Specification

Matchmaking is performed with friends registered in the Wii friend roster.

Use the `DWC_ConnectToFriendsAsync` function to begin peer matchmaking with friend specification (see Code 7-2).

The following items are passed to this function as arguments:

- Wii friend roster index array for the friends that you want to involve in matchmaking (hereafter, the index list)
- Number of elements in the index list
- Desired number of players, including the user
- Permission for matchmaking for friends of friends

- Matchmaking completion callback and its parameters
- Player evaluation callback (described below) and its parameters

When null is specified for the index list, all the friends on the Wii friend roster are considered candidates for matchmaking.

The Wii friend roster referenced during friend-specified peer matchmaking is specified with the `DWC_InitFriendsMatch` function.

Since it is highly likely that each player will have a different Wii friend roster or will have specified different index lists, the success rate for matchmaking of friends' friends (when permitted) will decrease drastically.

Code 7-2 Peer Matchmaking with Friend Specification

```
static BOOL s_matched  = FALSE;
static BOOL s_canceled = FALSE;

void do_friend_match( void )
{
   // Begin matchmaking with friend specification
   DWC_ConnectToFriendsAsync( NULL, 0, 4, TRUE,
                              cb_friendmatch, NULL,
                              NULL, NULL );

   // Matchmaking completion polling
   while ( !s_matched )
   {
      DWC_ProcessFriendsMatch();

      if ( DWC_GetLastErrorEx( NULL, NULL ) )
      {
         // An error occurred.
         handle_error();
         return;
      }

      GameWaitVBlankIntr();
   }

   // Matchmaking completion
   :
}

// Matchmaking completion callback
void cb_friendmatch( DWCError error, BOOL cancel, void* param )
```

```
{
   if ( error == DWC_ERROR_NONE )
   {
      if ( cancel ) s_canceled = TRUE;
      else          s_matched  = TRUE;
   }
}
```

7.3 Evaluating Matchmaking Candidates

During matchmaking, multiple players found as matchmaking candidates are evaluated with indicators unique to the game. Consequently, candidates can be ranked for matchmaking.

When an evaluation callback is specified as an argument for the peer matchmaking start function, a specified evaluation callback is called each time a matchmaking candidate is found. In this callback, the matchmaking indicator key registered with the `DWC_AddMatchKey*` functions can be referenced using the `DWC_GetMatch*Value` functions (see Code 7-3). Evaluate the player based on that value, and return the ranking.

An evaluation value of zero or less removes the player as a matchmaking candidate. This does not mean that the player with the top evaluation value is *always* selected, but the higher the evaluation value, the more *likely* it is that the player will be selected.

Code 7-3 Evaluating Matchmaking Candidates

```
static const char* s_str_key = "anymatch_test";
static const int s_int_key = 10;

void do_anybody_match( void )
{
   // Set the matchmaking indicator keys
   DWC_AddMatchKeyString( 0, "str_key", s_str_key );
   DWC_AddMatchKeyInt( 0, "int_key", s_int_key );

   // Begin matchmaking without friend specification
   DWC_ConnectToAnybodyAsync( 4,
                              "str_key = 'anymatch_test'",
                              cb_anymatch, NULL,
                              cb_eval, NULL );
   :
}

// Player evaluation callback
int cb_eval( int index, void* param )
{
    int eval_int;

    // Get the value for the matchmaking indicator key, int_key
    eval_int = DWC_GetMatchIntValue(index, "int_key", -1);

    if ( eval_int >= 0 )
    {
        // How close that value is to your own is the evaluation value
        return MATH_ABS( s_int_key - eval_int ) + 1;
    }
    else
    {
        // Players without the int_key key are not considered for matchmaking
        return 0;
    }
}
```

7.4 Server-Client Matchmaking

Server-client matchmaking is performed among friends by clearly classifying each host as either a server or a client. It is similar to peer-to-peer matchmaking in that the completed network is a mesh network.

Server hosts call the `DWC_SetupGameServer` function, specifying the following items as arguments:

- Desired maximum number of players, including the user
- Matchmaking completion callback and its parameters
- New connection client notification callback and its parameters

It then waits for client-host connections to appear (see Code 7-4).

Once a client host calls the `DWC_ConnectToGameServerAsync` function, whose arguments are specified with the Wii friend roster index for friends to connect with, the matchmaking completion callback and its parameters, and the new connection client notification callback and its parameters. The client host can then connect if a friend has begun matchmaking as a host (see Code 7-4).

When server-client matchmaking is complete, it is possible that some of the client hosts connected to the server hosts will be friends of friends on the server host, even though all of the client hosts will be friends of the server host.

The matchmaking completion callback is called not only when your connection to the server host is successful, but also when another client host is added to the mesh network to which you belong.

The new connection’s client notification callback is called when a new client host starts a new connection to the mesh network to which you already belong.

Code 7-4 Server-Client Matchmaking

```
static BOOL s_matched = FALSE;

void do_server_match( void )
{
   // Begin matchmaking as the server host
   DWC_SetupGameServer( 4,
                        cb_sc_match, (void *)CB_CONNECT_SERVER,
                        cb_sc_new, NULL );

   while ( 1 )
   {
      DWC_ProcessFriendsMatch();

      if ( DWC_GetLastErrorEx( NULL, NULL ) )
      {
         // An error has occurred.
         handle_error();
         return;
      }

      if ( s_matched )
      {
         // When a connection is completed with a new connection client
         init_new_connection();
         s_matched = FALSE;
      }

      GameWaitVBlankIntr();
   }
   :
}

void do_client_match( void )
{
   // Begin matchmaking as a client host
   DWC_ConnectToGameServerAsync( 0,
                                 cb_sc_match, (void *)CB_CONNECT_CLIENT,
                                 cb_sc_new, NULL );

   // Matchmaking completion polling
   while ( !s_matched )
   {
      DWC_ProcessFriendsMatch();

      if ( DWC_GetLastErrorEx( NULL, NULL ) )
```

```
        {
            // An error has occurred.
            handle_error();
            return;
        }

        GameWaitVBlankIntr();
    }

    // Matchmaking completion
    :
}

// Matchmaking completion callback
void cb_sc_match( DWCError error, BOOL cancel, BOOL self, BOOL isServer, int index, void* param )
{
    if ( error == DWC_ERROR_NONE )
    {
        if ( !cancel )
        {
            // Connection successful
            s_matched = TRUE;
        }
        else if ( self || isServer )
        {
            // If you cancel matchmaking or if you are a client host
            // and the server host cancels matchmaking
            s_canceld = TRUE;
        }
        // Do nothing if a new connection client cancels matchmaking
    }
}

// New connection client notification callback
void cb_sc_new( int index, void* param )
{
    printf( "Newcomer : friend[%d].\n", index );
}
```

Server-client matchmaking, like peer matchmaking, creates a mesh network. Therefore, even if the server host disconnects, the remaining client hosts can communicate with each other. However, server-client matchmaking cannot continue without a server host; thus we recommend that you implement the software such that all hosts disconnect as soon as the server host is disconnected. In addition, by calling the `DWC_StopSCMatchingAsync` function during matchmaking, the server host can deny client connections.

7.5 Increasing Matchmaking Speed

With unspecified peer matchmaking, when a matchmaking candidate list is retrieved from the matchmaking server, using filtering can increase matchmaking speed (see Code 7-1).

The matchmaking candidate list on the matchmaking server is a list that combines various conditions. As a result, getting a list with no conditions or filtering matchmaking candidates in the evaluation callback can increase the possibility of failed matchmaking. Repeated attempts to get the list waste time.

Because filtering can increase the possibilities for a workable candidate list, it can serve to increase matchmaking speed.

With extreme filtering in the evaluation callback (for same-level battles, same-region battles, or other conditions under which the number of candidates are thought to be few), the success rate for matchmaking goes down.

When you try to increase the matchmaking speed, consider the following:

- Using filtering to make a workable candidate list from the retrieved matchmaking candidates list
- Creating specifications that optimize candidates without using extreme filtering in the evaluation callback

7.6 Key Names that Cannot be Used for Matchmaking Indicator Keys

There are several key names, used by the library and server, which cannot be used as the matchmaker indicator keys registered with the `DWC_AddMatchKey*` functions. Do not use the key names in Table 7-1.

Table 7-1 List of Key Names Prohibited for Use as Matchmaking Indicator Keys

country	region	hostname	gamename	gamever	hostport
mapname	gametype	gamevariant	numplayers	numteams	maxplayers
gamemode	teamplay	fraglimit	teamfraglimit	timeelapsed	timelimit
roundtime	roundelapsed	password	groupid	player_	score_
skill_	ping_	team_	deaths_	pid_	team_t
score_t	dwc_pid	dwc_mtype	dwc_mresv	dwc_mver	dwc_eval

8 Sending and Receiving Data

8.1 Peer-to-Peer Data Sending and Receiving

When matchmaking is completed, connections are established mutually among the hosts, creating a mesh network. Some preparation is needed before communication among hosts on this network can begin.

First, the receipt buffer must be set to receive data from the hosts. Although the `DWC_SetRecvBuffer` function is used to do this, the AID (identifier for each host)[1] is specified for its `aid` argument (see Code 8-1).Any data received before the receipt buffer is set is destroyed.

Next, the send/receive callbacks must be set using the `DWC_SetUserSendCallback` and `DWC_SetUserRecvCallback` functions (see Code 8-1). The receive callback is called when data is received from another host. The send callback is called immediately after the data specified for sending has been completely sent. “Completely sent” refers to the complete passing of data to the low-layer send functions. It does not refer to the arrival of that data at its intended recipient.

Another callback, the connection closed callback, is set to be called when either your device or that of another host properly disconnects from the network. This uses the `DWC_SetConnectionClosedCallback` function (see Code 8-1).

These settings are not cleared until the `DWC_ShutdownFriendsMatch` function is called. They do not need to be set immediately after the matchmaking has completed.

Code 8-1 Preparing to Send and Receive Data

```
static u8 s_RecvBuffer[ 3 ][ SIZE_RECV_BUFFER ];

void prepare_communication( void )
{
   u8* pAidList;
   int num = DWC_GetAIDList( &pAidList );
   int i, j;

   for ( i = 0, j = 0; i < num; ++i )
   {
      if ( pAidList[i] == DWC_GetMyAID() )
      {
         j++;
         continue;
```

[1] AID is a numerical value that ranges from zero to the number of devices in the network, minus 1. For example, if four players have completed matchmaking, there are four devices numbered 0, 1, 2, and 3. If the person with AID = 1 leaves, the remaining AID values are 0, 2, and 3.

```
        }

        // Set a receive buffer for an AID other than your own
        DWC_SetRecvBuffer( pAidList[i], &s_RecvBuffer[i-j], SIZE_RECV_BUFFER );
    }

    // Set the send callback
    DWC_SetUserSendCallback( cb_send );

    // Set the receive callback
    DWC_SetUserRecvCallback( cb_recv );

    // Set the connection closed callback
    DWC_SetConnectionClosedCallback( cb_closed, NULL );
}

// Send data callback
void cb_send( int size, u8 aid )
{
    printf( "to aid = %d  size = %d\n", aid, size );
}

// Data receive callback
void cb_recv( u8 aid, u8* buffer, int size )
{
    printf( "from aid = %d  size = %d  buffer[0] = %X\n",
                  aid, size, buffer[0] );
}

// Connection close callback
void cb_closed( DWCError error, BOOL isLocal, BOOL isServer, u8  aid, int index, void* param)
{
    if ( error == DWC_ERROR_NONE )
    {
        if ( isLocal )
        {
            printf( "Closed connection to aid %d
                        (friendListIndex = %d).\n", aid, index );
        }
        else
        {
            printf( "Connection to aid %d
                           (friendListIndex = %d) was closed.\n", aid, index );
        }
    }
```

The two types of data transmission, reliable and unreliable, both use UDP communications. However, reliable transmissions, like TCP communications, have no packet loss. Although you cannot rearrange the order in which packets arrive, send confirmation occurs for each packet's arrival, which lengthens the send time.

Unreliable transmissions use unmodified UDP communications. Although both of the above problems may occur, unreliable transmission is faster because there is no data arrival confirmation or re-sending.

When sending data is delayed in a layer lower than DWC, the data is stored in the send buffer, which has the size specified with the `DWC_InitFriendsMatch` function. If you try to use reliable transmission and there is not sufficient room in the buffer, the data that could not be sent is held; when enough room becomes available in the buffer, this data is sent from the `DWC_ProcessFriendsMatch` function.

In addition, the maximum size of the data that can be sent at one time is also set (1,465 bytes by default). Attempting to send more than this amount will result in partitioning of the data and in holding it for sending. Although this maximum size can be changed using the `DWC_SetSendSplitMax` function, increasing it risks losing compatibility with the settings of communication devices.

Do not destroy the send buffer while the send data is being held. In addition, the next data cannot be sent while data is being held.

The `DWC_IsSendableReliable` function can be used to check whether a reliable transmission is possible, including checking that there is space in the send buffer and that the recipient AID is valid (see Code 8-2).

With unreliable transmission, if you attempt to send more than the data size noted above, the send fails and FALSE is returned.

Code 8-2 Sending Data

```
static u8  s_SendBuffer[ SIZE_SEND_BUFFER ];

void send_data( void )
{
    // An unreliable transmission of data to all connected hosts
    // Your own AID will be ignored even if passed.
    DWC_SendUnreliableBitmap( DWC_GetAIDBitmap(), s_SendBuffer, SIZE_SEND_BUFFER );
    :

    // Determine whether a reliable transmission to the host whose AID = 0 can be done
    if ( !DWC_IsSendableReliable( 0 ) ) return;
    // A reliable transmission of data to a specified host.
    DWC_SendReliableBitmap( 0, s_SendBuffer, SIZE_SEND_BUFFER );
    :
}
```

8.2 Terminating a Connection

To disconnect all host connections on a mesh network, call the `DWC_CloseAllConnectionsHard` function. Once the close process has run, the connection-closed callback set in the `DWC_SetConnectionClosedCallback` function is called before exiting. The connected hosts receive a close notification simultaneously, and the connection-closed callback is called.

This function can also be called for server hosts in the server-client matchmaking model, even when no hosts are currently connected. In this case, any remaining matchmaking memory is deallocated and the communication status is restored to an online state.

The connection to a Wi-Fi Connection Server is not lost when this function is called.

The `DWC_CloseConnectionHard` function disconnects the connection for a specified AID, and the `DWC_CloseConnectionHardBitmap` simultaneously closes multiple connections for a specified AID bitmap.

It is assumed that these functions can be used to close connections in abnormal circumstances, such as when communication with a host is no longer possible due to a loss of power.

8.3 Targets for the Buffer Size Specified by DWC_InitFriendsMatch

The buffer size specified by the `DWC_InitFriendsMatch` function becomes the size of the buffer used internally by DWC. The send buffer is used to store reliable transmission data for which an ACK has not been returned. The receive buffer is used to store data that did not arrive in the correct order.

For reliable communications, you need the largest send and receive buffers permitted by the game specifications for the duration of power fluctuations to support network power fluctuations to the greatest extent possible. For unreliable transmission, send and receive buffers are normally not used. However, because the DWC uses reliable transmission internally during peer-to-peer connections, a minimum of 1 KB is needed for the send buffer and 128 bytes for the receive buffer.

Table 8-1 Communication Content and Buffer Size Targets

Communication Type		Buffer Size Targets	Notes
Reliable transmission	Send buffer size	[Time (in seconds) allowed in game specifications for power fluctuations] * [amount of reliable data per second] * [reliable data size] (reliable data size = 7 * the number of transmission data partitions * the transmission data size + 15)	At least 1 KB
	Receive buffer size		At least 128 bytes
Unreliable transmission	Send buffer size	Maximum unreliable transmission data size + 2 bytes	At least 1 KB
	Receive buffer size	Minimum of 128 bytes	

Note: The number of transmission data partitions refers to the partitions created in the transmission data when its size exceeds the maximum size for a single transmission (set in the `DWC_SetSendSplitMax` function, with a default of 1,465 bytes).

Assuming that game specifications allow for power fluctuations lasting 1 second, that communication occurs at a frequency of once every three frames, and that the maximum amount of data that can be sent at once for a game is 64 bytes, the size of the buffer needed for a reliable transmission of 100 bytes of data is calculated as follows:

1 (second) x (60 (frames) ÷ 3) x (7 x 2 (partitions) x 100 (bytes) + 15) = 28,300 (bytes)

8.4 Emulations of Packet Loss and Delays

DWC can emulate the delays and packet loss in sending and receiving data.

However, for transmission delays, data is lost and is not delivered when the connection is closed, because the transmission data is copied to a separate buffer for a specified time. We therefore recommend using only receive delays.

The following example (Code 8-3) specifies the packet loss rates (as a percentage), the delay time (in milliseconds), and the target host AID.

Code 8-3 Delays and Packet Loss Emulation

```
void set_trans_emulation( void )
{
   DWC_SetSendDrop( 30, 0 );
   DWC_SetRecvDrop( 30, 0 );

   DWC_SetSendDelay( 300, 0 );
   DWC_SetRecvDelay( 300, 0 );
   :
}
```

8.5 Data Send and Receive Volumes (When Wireless Communications Are Used)

Details on the amount of data transferred during reliable and unreliable transmissions are given in the table below.

Table 8-2 Communication Content and Buffer Size Targets

Transfer Data Item	**Transfer Data Size**			
Preamble	192 bits (24 Bytes)			
MAC	24 Bytes			
LLC	8 Bytes			
IP	20 Bytes			
UDP	8 Bytes			
DATA	Reliable Communications			Unreliable Transmission
	Header transfer	Data transfer	Receive check	Data transfer
	15Byte	7 + XXX Bytes	5 Bytes	XXX Bytes
FCS	4 Bytes			
B (random time for avoiding packet collisions)	MAX 600 µsec			

Note: As a part of reliable transmission, a header is sent and a receive check is performed before and after data transfer.

The data transfer time given for each transmission can be found by calculating

Preamble + (MAC + LLC + IP + UDP + DATA + FCS) x 4 + B [µsec]

However, it is difficult to accurately calculate the amount of data sent and received due to the fact that the transfer time varies depending on such factors as the number of retries caused by bandwidth conditions, the number of sent packets, and the amount of standby time required to avoid collisions.

- Unlike NITRO-DWC, hardware performance is not a bottleneck issue when using Revolution DWC.

According to actual measurements in a test where 1424 bytes per frame were sent to two other players in an environment where three consoles connected by different Internet lines had been matched together, we confirmed that communications without packet loss are possible. (Data was both sent and received at a rate of approximately 170 KB/sec.)

In game development, the design will benefit by integrating the following considerations:

1. Network Environment
    a. Internet Transmission Latency and Packet Loss (domestic and international)
        - Transmission latency is fairly uniform domestically, but tends to increase in international communications.

    b. Congestion in Wireless Environments
        - With large numbers of packets, congestion occurs more easily in wireless environments than in wired environments.

    c. Mixed Wireless and Wired Environments
        - When wireless and wired environments are mixed and the number of packets is large, the wireless side may experience a delayed transmission or reception even if the wired side's transmission and reception are reliable.

2. Communication delays due to other IOP processing loads
    a. When the IO processor is monopolized by disc access or another process, the socket will receive buffer overflows.
        - The socket operates on the IO processor. Thus, if processing is not transferred to the socket within a fixed time, the receive buffer will overflow and packets may be dropped.

Based on the above considerations, the transmission packet size and the number of packets transmitted in a given unit of time should be fine-tuned for each game. If, to ensure the game contents, you have to restrict the connections due to the connection quality, refer to *The Nintendo Wi-Fi Connection Programming Guidelines for Wii* for an appropriate course of action.

9 HTTP Communications

DWC uses the GHTTP library for HTTP uploading and downloading of data. It can be used independently of matchmaking and of friend relationship functionality.

9.1 Preparing to Use the GHTTP Library

Before using the GHTTP library, you must first call the `DWC_InitGHTTP` function (see Code 9-1).

Specify a null value for the argument. The return value is always TRUE.

The GHTTP library can be used after the `DWC_InitGHTTP` function is called and the Internet connection is established.

Code 9-1 Initializing the GHTTP Library

```
void init_ghttp( void )
{
   // DWC initialization
   init_dwc();

   // Initialize GHTTP
   DWC_InitGHTTP( NULL );
}
```

9.2 Uploading Data

To upload data to an HTTP server, you must first use the GHTTP library to create a `DWCGHTTPPost`-type object with the `DWC_GHTTPNewPost` function. (see Code 9-2). Next, with the `DWC_GHTTPPostAddString` function, add to that object the data to upload (see Code 9-2).

Specify the following pointers as arguments to the `DWC_GHTTPPostAddString` function:

- Pointer to the `DWCGHTTPPost`-type object
- Pointer to the key string that identifies the data
- Pointer to the data (a value string) that you want to add

Both the key and value strings are copied and saved in the library. In addition, both strings must be null-terminated. If a null value string is specified, it has the same effect as specifying a null-terminated empty string (“”).

Use the `DWC_PostGHTTPData` function to begin the data upload (see Code 9-2). The following arguments are passed to this function:

- Upload target URL
- Pointer to the `DWCGHTTPPost`-type object
- Completion callback and its parameters

Call this function approximately every game frame, because all communication processes after the beginning of the upload occur in the `DWC_ProcessGHTTP` function (see Code 9-2).

Once the data upload is complete, the completion callback is called.

The `DWCGHTTPPost`-type object is deallocated once the upload is complete and immediately after exiting the completion callback.

In the example in Code 9-2, the data uploaded to the HTTP server is as follows:

```
  "key1=value1&key2=value2"
```

Continuing to add data to the same `DWCGHTTPPost`-type object, the following string is added:

```
  "key1=value1&key2=value2&key3=value3&key4=value4…"
```

Code 9-2 Uploading Data

```
static int s_send_cb_level = 0;

void post_ghttp_data( void )
{
    int req;
    DWCGHTTPPost post;

    // Create a DWCGHTTPPost-type object
    DWC_GHTTPNewPost( &post );

    // Set the data to be uploaded to the DWCGHTTPPost-type object
    DWC_GHTTPPostAddString( &post, "key1", "value1" );
    DWC_GHTTPPostAddString( &post, "key2", "value2" );

    // Begin uploading the data
    s_send_cb_level++;
    req = DWC_PostGHTTPData( "http://www.test.net", &post, cb_post, NULL );

    if ( req < 0 )
    {
        // An error occurred.
        handle_error();
        return;
    }

    while ( s_send_cb_level )
    {
        // Proceed with the upload process
        DWC_ProcessGHTTP();
        GameWaitVBlankIntr();
    }

    if ( DWC_GetLastErrorEx( NULL, NULL ) )
    {
        // An error occurred.
        handle_error();
        return;
    }

    // Data upload succeeded
    :
}

// Upload completion callback
void cb_post( const char* buf, int buflen, DWCGHTTPResult result, void* param)
{
    s_send_cb_level--;
}
```

9.3 Downloading Data

There are two functions for downloading data from an HTTP server. `DWC_GetGHTTPData` is the simpler function, whereas `DWC_GetGHTTPDataEx` offers an expanded range of features (see Code 9-3).

The following arguments are passed to `DWC_GetGHTTPDataEx`:

- URL from which the data is to be downloaded
- Size of the receive buffer (even if the buffer is deallocated after downloading has completed)
- Get communication status callback and its parameters
- Completion callback and its parameters

When the receive buffer size is specified as zero, the initial 2,048 bytes of memory are allocated, with additional blocks of 2,048 allocated as required by the received data. Data can be received up to the heap memory amount allocated by the application.

When the get communication status callback is specified, it is called every time there is a change in the download sequence status (such as when sending the request and receiving data). It can also be used to confirm the size of the received data while data is still being received.

The completion callback is called when the download has been completed.

If the receive buffer is set to be deallocated when the download is complete, copy the received data before using it, because the receive buffer is deallocated immediately after exiting the completion callback.

If the receive buffer is not set to be deallocated, the receive buffer pointer passed as an argument to the completion callback can be deallocated whenever convenient for the application, because the GHTTP library does not deallocate the receive buffer. Use the `DWC_Free` function to deallocate that buffer.

The `DWC_GetGHTTPData` function works identically to the `DWC_GetGHTTPDataEx` function when the latter's `bufferlen` argument is set to 0, `buffer_clear` is set to TRUE and `progressCallback` is NULL.

Call this function for every game frame, because after the download begins, all communication processes occur in the `DWC_ProcessGHTTP` function (see Code 9-3).

Code 9-3 Downloading Data

```
static char s_recvBuffer[ 2 ][ SIZE_RECV_BUFFER ];
static int  s_get_cb_level = 0;

void get_ghttp_data( void )
{
    // Begin downloading data with the simple function
    s_get_cb_level++;
    req = DWC_GetGHTTPData( "http://www.test.net", cb_get, GET_TYPE_NORMAL );

    if ( req < 0 )
    {
        // An error occurred.
        handle_error();
        return;
    }

    // Begin downloading data with the expanded function
    s_get_cb_level++;
    req = DWC_GetGHTTPDataEx( "http://www.test.net",
                             RECV_SIZE, TRUE,
                             NULL, cb_get, GET_TYPE_EX );

    if ( req < 0 )
    {
        // An error occurred.
        handle_error();
        return;
    }

    while ( s_get_cb_level )
    {
        // Proceed with the download process
        DWC_ProcessGHTTP();
        GameWaitVBlankIntr();
    }

    if ( DWC_GetLastErrorEx( NULL, NULL ) )
    {
        // An error occurred.
        handle_error();
        return;
    }

    // Data download successful
    :
}

// Download completion callback
void cb_get( const char* buf, int buflen, DWCGHTTPResult result, void* param)
{
    s_get_cb_level--;

    if ( result == DWC_GHTTP_SUCCESS )
    {
        if ( (int)param == GET_TYPE_NORMAL )
        {
            MI_CpuCopy8( buf, s_recvBuffer[ 0 ], SIZE_RECV_BUFFER );
        }
        else if ( (int)param == GET_TYPE_EX )
        {
            MI_CpuCopy8( buf, s_recvBuffer[ 1 ], SIZE_RECV_BUFFER );
        }
    }
}
```

9.4 Closing the GHTTP Library

To terminate the GHTTP library, call the `DWC_ShutdownGHTTP` function. If the `DWC_InitGHTTP` function is called multiple times, `DWC_ShutdownGHTTP` must be called an equal number of times to deallocate all the memory allocated to the GHTTP library.

10 Network Storage Support

DWC can store data on storage servers on the GameSpy network. To access these storage servers, first complete the login process with the `DWC_LoginAsync` function. Next, use the `DWC_LoginToStorageServerAsync` function to log in to the storage servers (see Code 10-1).

Data saved on the storage server can take either a public or a private attribute. Data saved using the `DWC_SavePublicDataAsync` function takes a public attribute, and other players can reference its data (see Code 10-1). Conversely, data saved using the `DWC_SavePrivateDataAsync` function takes a private attribute, and other players cannot reference its data.

When loading storage server data, you can use the `DWC_LoadOwnPublicDataAsync` function to load public data, the `DWC_LoadOwnPrivateDataAsync` function to load private data, or the `DWC_LoadOthersDataAsync` function to load data from friends who are specified in a Wii friend roster (see Code 10-1).

When saving and loading is complete, the callback function set with the `DWC_SetStorageServerCallback` function is called (see Code 10-1). Callback functions are always called in the same order as the save and load functions.

Save data is specified as a string that is a combination of keys and values. It is delimited by `\\`, with the key and its value alternating. For example, the string `\\name\\mario\\stage\\3` would be saved in the storage server database with the `name` key set to a value of “`mario`” and the `stage` key set to a value of `3`.

When loading data, specify the keys, separated by the `\\` delimiter, for the values you want to obtain. For example, given `\\name\\stage`, the string that the load callback gets would be formatted as `\\name\\mario\\stage\\3`.

When attempting to load either non-existent keys or keys that a friend has saved with the `private` attribute, the callback argument `success` is set to FALSE. However, if only some of the specified keys are unavailable as above, `success` returns TRUE, and the loaded data does not include the unavailable data.

After all the storage server processes have completed, use the `DWC_LogoutFromStorageServer` function to log out of the storage server (see Code 10-1).

Code 10-1 Accessing a Storage Server

```
static int  s_cb_level = 0;
static BOOL s_storage_logined = FALSE;

void access_net_storage( void )
{
   // Log in to the storage server
   if ( !DWC_LoginToStorageServerAsync( cb_storage_login, NULL ) )
   {
      printf( "DWC_LoginToStorageServerAsync() failed.\n" );
```

```
        return;
    }

    // Waiting for storage server login to complete
    while ( !s_storage_logined )
    {
        DWC_ProcessFriendsMatch();

        if ( DWC_GetLastErrorEx( NULL, NULL ) )
        {
            // An error occurred.
            handle_error();
            return;
        }

        GameWaitVBlankIntr();
    }

    // Set the save, load completion callbacks for the storage server
    DWC_SetStorageServerCallback( cb_save_storage, cb_load_storage );

    // Save public data
    s_cb_level++;
    if ( !DWC_SavePublicDataAsync( "\\name\\mario\\stage\\3", NULL ) )
    {
        printf( "DWC_SavePublicDataAsync() failed.\n" );
        return;
    }

    // Save private data
    s_cb_level++;
    if ( !DWC_SavePrivateDataAsync( "\\id\\100", NULL ) )
    {
        printf( "DWC_SavePrivateDataAsync() failed.\n" );
        return;
    }

    // Waiting for save completion
    while ( s_cb_level > 0 )
    {
        DWC_ProcessFriendsMatch();
        if ( DWC_GetLastErrorEx( NULL, NULL ) )
        {
            // An error occurred.
            handle_error();
            return;
        }

        GameWaitVBlankIntr();
    }

    // Load your own public save data
    s_cb_level++;
    if ( !DWC_LoadOwnPublicDataAsync( "\\name", NULL ) )
    {
        printf( "DWC_LoadOwnPublicDataAsync() failed.\n" );
        return;
    }

    // Load your own private save data
    s_cb_level++;
    if ( !DWC_LoadOwnPrivateDataAsync( "\\id", NULL ) )
    {
        printf( "DWC_LoadOwnPrivateDataAsync() failed.\n" );
        return;
    }

    // Load someone else’s saved data
    s_cb_level++;
    if ( !DWC_LoadOthersDataAsync( "\\name", 0, NULL ) )
    {
        printf( "DWC_LoadOthersDataAsync() failed.\n" );
        return;
    }

    // Waiting for load completion
    while ( s_cb_level > 0 )
    {
        DWC_ProcessFriendsMatch();

        if ( DWC_GetLastErrorEx( NULL, NULL ) )
        {
            // An error occurred.
            handle_error();
            return;
        }

        GameWaitVBlankIntr();
    }

    // Log off the storage server
    DWC_LogoutFromStorageServer();
    :
}

// Storage server login completion callback
void cb_storage_login( DWCError error, void* param )
{
    if ( error == DWC_ERROR_NONE )
    {
        s_storage_logined = TRUE;
        s_cb_level        = 0;
    }
}

// Storage server save completion callback
void cb_save_storage( BOOL success, BOOL isPublic, void* param )
{
    printf( "result %d, isPublic %d.\n", success, isPublic );
    s_cb_level--;
}

// Storage server load completion callback
void cb_load_storage( BOOL success, int index, char* data, int len, void* param )
{
    printf( "result %d, index %d, data '%s', len %d\n",
                  success, index, data, len );
    s_cb_level--;
}
```

11 General Ranking Library

11.1 Introduction

This chapter describes how to use the Revolution DWC general ranking library, how to use management tools, and the specifications of the Web development interface.

The following capabilities are provided by the general ranking library:

- Uploading of scores
- Getting one's own ranking
- Getting a top ranking list
- Getting a ranking list of scores close to one's own
- Getting a ranking list consisting of friends and rivals

The general ranking library provides functions for carrying out communications with the ranking server that is used to implement these capabilities. The general ranking library uses `DWC_GHTTP` internally to carry out communications with the ranking server. The general ranking library and `DWC_GHTTP` cannot be used at the same time. Furthermore, error codes depend on `DWC_GHTTP`, because all communication errors are generated by `DWC_GHTTP` when using general ranking. During actual use, be sure to follow the instructions given in *Nintendo Wi-Fi Connection Programming Guidelines for Wii*.

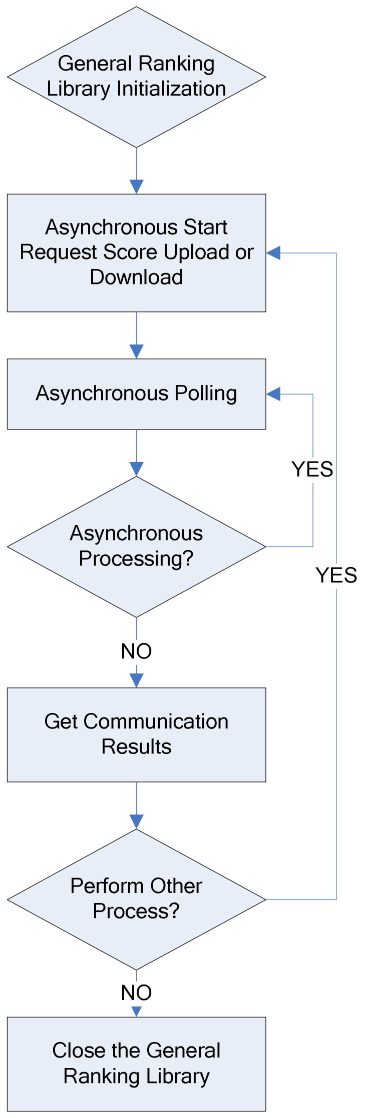
11.2 Using the General Ranking Library

This section describes the use of the general ranking library.

11.2.1 Process Flow

The process flow for the general ranking library is shown in Figure 11-1.

Figure 11-1 Process Flow


General Ranking Library Initialization
Asynchronous Start Request Score Upload or Download
Asynchronous Polling
YES
Asynchronous Processing?
YES
NO
Get Communication Results
Perform Other Process?
NO
Close the General Ranking Library

Note: When the asynchronous process is canceled or an error occurs, the general ranking library must be terminated just once.

11.2.2 Initializing the General Ranking Library

To initialize the general ranking library, begin by calling the `DWC_RnkInitialize` function. The following arguments are passed to this function:

- Initialization Data

  This specifies the general ranking library initialization data string, sent by Nintendo after receipt of your design statement. A unique string is provided for each title and must therefore be strictly managed.

- User Data

  This specifies the user data `DWCUserData` structure. This account data must be for an authenticated account. Attempts to use unauthorized accounts that have never connected to Nintendo Wi-Fi Connection result in a failed initialization.

Two general ranking servers are available: one for development and one for released products. Which server you will be connected to is decided at initialization, based on the authentication server logged into. When `DWC_CONNECTINET_AUTH_RELEASE` is specified in the `DWC_Init` function, you will connect to the released products server. When `DWC_CONNECTINET_AUTH_TEST` is specified, you will connect to the development server.

The databases for the two servers are separate and are constructed so that ranking is different on each.

Be sure to use the released product server for released products.

Code 11-1 Initializing the Ranking Library

```
DWCRnkError res;

// Initialize the ranking library
res = DWC_RnkInitialize( RANKING_INITDATA,
                                 &userdata );

if( res != DWC_RNK_SUCCESS ){

           break; // Initialization failure, error processing

}
```

11.2.3 Uploading Scores

Scores are uploaded in an asynchronous process. To begin the process, call the `DWC_RnkPutScoreAsync` function. Only one asynchronous process can be run at a time.

Polling continues from the beginning of the asynchronous process until the `DWC_RnkProcess` function, described below, is completed.

Scores to be uploaded can specify the category ID, market, score, and user-defined data.

The category ID is an identifier for game titles that have multiple ranking lists and can be in the range of zero to 100 (`DWC_RNK_CATEGORY_MAX`).

For the market, you can specify `DWC_RNK_REGION_JP` (Japan), `DWC_RNK_REGION_US` (North America), `DWC_RNK_REGION_EU` (Europe), or `DWC_RNK_REGION_KR` (Korea). The specified market can be used as a filter when downloading scores. For example, it can be used to get rankings only in Japan, or to combine rankings for Japan and the USA.

The user-defined data can specify any binary up to 764 bytes (`DWC_RNK_DATA_MAX`) in length.

When a score is already registered for the same category ID, the newly uploaded score always overwrites an older score.

Code 11-2 Initial Request for Uploading a Score

```
DWCRnkError res;

// Begin the score upload request
res = DWC_RnkPutScoreAsync(      10,             // category
                         DWC_RNK_REGION_JP,     // market
                         1234,                  // score
                         (void*)"test data",    // user defined data
                         strlen("test data") + 1 );

if( res != DWC_RNK_SUCCESS ){

break; // Failure, error processing

}
```

11.2.4 Downloading Scores

Scores are downloaded in an asynchronous process. To begin the process, call the `DWC_RnkGetScoreAsync` function. Only one asynchronous process can be run at a time.

Polling continues from the beginning of the asynchronous process until the `DWC_RnkProcess` function, described below, is completed.

There are four get modes for downloading scores:

- Order
- Top ranking list
- Nearby ranking list
- Friends ranking

Table 11-1 Get Modes for Downloading Scores

Order	Gets your own ranking. Gets the order by comparing registered scores. Ascending and descending are specified as parameters.
Top Ranking List	Gets the specified number of rankings from the top.
Nearby Ranking List	Gets the specified number of rankings nearest to your own.
Friends Ranking List	Gets the ranking for a maximum of 64 friends.

11.2.4.1 Specifying the DWCRnkGetParam Structure

The following is a description of the specification of the `DWCRnkGetParam` structure parameter passed as an argument to the `DWC_RnkGetScoreAsync` function. This structure is defined as the one that contains multiple unions, and its parameters must be set to the appropriate fields, based on the get mode:

DWCRnkGetParam.size field

This specifies the size of the structure for all get modes. The size for each get mode is shown in Table 11-2.

Table 11-2 Size Specified for Each Get Mode's Parameter

Order `(DWC_RNK_GET_MODE_ORDER)`	`sizeof( DWCRnkGetParam.order )`
Top Ranking List `(DWC_RNK_GET_MODE_TOPLIST)`	`sizeof( DWCRnkGetParam.toplist )`

Nearby Ranking List
`(DWC_RNK_GET_MODE_NEAR)`
`(DWC_RNK_GET_MODE_NEAR_HI)`
`(DWC_RNK_GET_MODE_NEAR_LOW)`
	`sizeof( DWCRnkGetParam.nearby )`

Friends Ranking List `(DWC_RNK_GET_MODE_FRIENDS)`	`sizeof( DWCRnkGetParam.friends )`

DWCRnkGetParam.order field

This specifies the size of the structure for the Order Get Mode. The parameter set for each Order Get Mode is shown in Table 11-3.

Table 11-3 Parameters Set with Order Get Mode

`DWCRnkGetParam.order.sort`	Specifies the scores' sorting order: • `DWC_RNK_ORDER_ASC:` ascending • `DWC_RNK_ORDER_DES:` descending
`DWCRnkGetParam.order.since`	Gets the rankings updated within the specified past number of minutes. Specifying 0 gets the ranking for all data. For example, specifying 180 will get the rankings that have been updated in the last 180 minutes (3 hours). Even if a given user hasn't uploaded scores within the specified time interval, the ranking within the specified time interval will be obtained for the score that was uploaded last.

DWCRnkGetParam.toplist field

This specifies the size of the structure for the Top Ranking List Get Mode. The parameter set for each Top Ranking List Mode is shown in Table 11-4.

Table 11-4 Parameters Set with Top Ranking List Mode

`DWCRnkGetParam.toplist.sort`	Specifies the score sorting order: • `DWC_RNK_ORDER_ASC`: ascending • `DWC_RNK_ORDER_DES`: descending
`DWCRnkGetParam.toplist.since`	Gets the rankings updated in the specified past number of minutes. Specifying zero gets the ranking for all data. For example, specifying 180 gets the rankings updated within the last 180 minutes (3 hours).
`DWCRnkGetParam.toplist.limit`	Specifies the maximum number of ranking lists to get. A numerical value between 1 and 10 (`DWC_RNK_GET_MAX`) can be specified.

DWCRnkGetParam.nearby field

This specifies the size of the structure for the Nearby Ranking List Get Mode. The parameter set for each Nearby Ranking List Get Mode is shown in Table 11-5.

Table 11-5 Parameters Set with Nearby Ranking List Get Mode

`DWCRnkGetParam.nearby.sort`	Specifies the score sorting order: • `DWC_RNK_ORDER_ASC`: ascending • `DWC_RNK_ORDER_DES`: descending
`DWCRnkGetParam.nearby.since`	Gets the rankings updated in the specified past number of minutes. Specifying zero gets the ranking for all data.
`DWCRnkGetParam.nearby.limit`	Specifies the maximum number of ranking lists to get. A numerical value between 2 and 10 (`DWC_RNK_GET_MAX`) can be specified. The values start at 2 because your score is always listed first.

DWCRnkGetParam.friends field

This specifies the size of the structure for the Friend Ranking List Get Mode. The parameter set for each Friend Ranking List Get Mode is shown in Table 11-6.

Table 11-6 Parameters Set with Friend Ranking List Get Mode

`DWCRnkGetParam.friends.sort`	Specifies the score sorting order: • `DWC_RNK_ORDER_ASC`: ascending • `DWC_RNK_ORDER_DES`: descending
`DWCRnkGetParam.friends.since`	Gets the rankings updated in the specified past number of minutes. Specifying zero gets the ranking for all data.
`DWCRnkGetParam.friends.limit`	Specifies the maximum number of ranking lists to get. A numerical value between 2 and 10 (`DWC_RNK_GET_MAX`) can be specified. The values start at 2 because your score is always listed first.
`DWCRnkGetParam.friends.friends[64]`	Specifies the friends' GS profile ID list. A maximum of 64 (`DWC_RNK_FRIENDS_MAX`) can be specified. If less than 64, store from the beginning, and fill in the remainder with zeros.

Code 11-3 Initial Request for Scores Download (Order Get)

```
DWCRnkError res;

// Begin order get request
DWCRnkGetParam    param;         // Parameters when getting rankings
param.size = sizeof( param.order );
param.order.since = 0;
param.order.sort = DWC_RNK_ORDER_ASC;

res = DWC_RnkGetScoreAsync(      DWC_RNK_GET_MODE_ORDER,        // mode
                                 10,                            // category
                                 DWC_RNK_REGION_JP,             // market
                                 &param );                      // parameter

if( res != DWC_RNK_SUCCESS ){

          break; // Failure, error processing

}
```

Code 11-4 Initial Request for Scores Download (Top Ranking List Get)

```
DWCRnkError res;

// Begin order get request
DWCRnkGetParam    param;         // Parameters when getting rankings
param.size = sizeof( param.toplist );
param.toplist.since = 0;
param.toplist.sort = DWC_RNK_ORDER_ASC;
param.toplist.limit = 10;

res = DWC_RnkGetScoreAsync(      DWC_RNK_GET_MODE_TOPLIST,      // mode
                                 10,                            // category
                                 DWC_RNK_REGION_JP,             // market
                                 &param );                      // parameter

if( res != DWC_RNK_SUCCESS ){

         break; // Failure, error processing

}
```

11.2.4.2 Getting Communication Results: Order Get Mode

When score downloading in Order Get Mode ends normally, you can get the communication results by calling the `DWC_RnkResGetOrder` function. This function fails if the `DWC_RnkGetScoreAsync` function is called in any mode other than Order Get Mode.

If your score is not uploaded when the get occurs, zero is returned.

11.2.4.3 Getting Communication Results: Ranking List Get Mode (Top, Nearby, Friend)

When score downloading in Ranking List Get Mode ends normally, you can use the `DWC_RnkResGetRowCount` function to get the number of rows in the retrieved score. By getting the data for each row with the `DWC_RnkResGetRow` function, you can access the retrieved ranking list. (See Note 1 below.)

The order is only stored in your score's row in the `DWCRnkData` structure's order field. Devise an appropriate numbering on the game side because a value of zero is stored in the order field for the other rows. (See Note 2 below.) An exception results when zeroes are stored in the order field of all rows in Top Ranking List Get Mode.

The list ordering of multiple users having the same score is undefined.

If you attempt to get Nearby or Friends Ranking Lists without having registered your score, it is treated as if your score were zero. In addition, the market code is returned as -1, and the user-defined data is blank.

The list's first index in both the Nearby and Friends Ranking List Get Modes always contains your score, regardless of the interval specified since.

Note 1: The pointer to user-defined data (`void* userdata`) in the `DWCRnkData` structure retrieved by the `DWC_RnkResGetRow` function directly references the internal communications buffer. Therefore, when the ranking library is closed or the next asynchronous process begins, buffer contents are lost.

Note 2: When the number of users above you with the same score cannot be determined in Nearby Ranking List Get Mode, the exact ordering of these users sometimes cannot be specified. When this happens (for certain display methods) the user may notice discrepancies with the list order that was retrieved in the Top Ranking List Get Mode. Given the loads on the server, this occurs by design and there are no basic workarounds. Employ other methods of compensating, including limiting the display of scores to yours alone. This reduces the possibility for duplicate scores through range increase or through manual numbering of the scores, even though there is a possibility for error.

11.2.4.4 Getting Communication Results: Getting Parameters

Calling the `DWC_RnkResGetTotal` function gets the ranking parameters retrieved with the `DWC_RnkGetScoreAsync` function. The parameters retrieved are the numbers of scores that match the filtering conditions specified when the `DWC_RnkGetScoreAsync` function is called.

11.2.4.5 Getting Ranking Lists Among Rivals

It is possible to designate the rivals' GS profile IDs in the GS profile ID list specified for Friends Ranking List Get Mode. Doing this allows the building of a ranking list among rivals.

However, you must follow *Nintendo Wi-Fi Connection Concept Guidelines for Wii* for handling data among users who are not friends.

Code 11-5 Accessing Retrieved Ranking Lists

```
DWCRnkError res;
u32 count;

// Get the number of rows in the retrieved list
res = DWC_RnkResGetRowCount( &count );

if( res != DWC_RNK_SUCCESS ){

           goto exit; // Failure, error processing

}

// Get row-by-row and output to debugging
for( i=0; i<count; i++ ){
          DWCRnkData data;

          if( DWC_RnkResGetRow( &data, (u32)i ) != DWC_RNK_SUCCESS ){
                    break;       // error
          }

          printf("%dth score=%d pid=%d rgn=%d update=%d data=%s \n",
                            data.order, data.score, data.pid,
                            data.region,data.lastupdate, data.userdata );
}
```

11.2.4.6 Using Asynchronous Polling for Getting Communication Result Status

Once an asynchronous process begins, call the `DWC_RnkProcess` function at regular intervals to carry out polling. Call once per each game frame. (It also works at 30 fps.)

The `DWC_RnkProcess` function returns `DWC_RNK_SUCCESS` during asynchronous processing. If there are no more tasks to be processed, `DWC_RNK_PROCESS_NOTASK` is returned. You can tell when an asynchronous process has ended by monitoring this return value.

If an error occurs, `DWC_RNK_IN_ERROR` is returned. Subsequent processing cannot continue once an error occurs, so you must close the general ranking library and re-initialize it.

Depending on the network conditions, there are times when the `DWC_RnkProcess` function does not exit. Cancel processing due to timeout must be implemented on the application side.

Code 11-6 Asynchronous Polling

```
// Asynchronous processing
while( (res = DWC_RnkProcess()) == DWC_RNK_SUCCESS ){

        // Timeout processing
        If( timedout() ){
              res = DWC_RnkCancelProcess();
              goto exit;
        }

        // V-Blank waiting
        GameWaitVBlankIntr();

}

switch( res ){

case DWC_RNK_PROCESS_NOTASK:     // Asynchronous process completed
      break;

case DWC_RNK_IN_ERROR:           // Failure, error processing
      goto exit;

}

switch( DWC_RnkGetState() ){

case DWC_RNK_STATE_COMPLETED:    // Success
      break;

case DWC_RNK_STATE_ERROR:        // Failure, error processing
      goto exit;

}
```

11.2.5 Closing the General Ranking Library

To close the library after all of the processes have completed or after an error has occurred, call the `DWC_RnkShutdown` function. Calling this function deallocates the library's memory and performs an internal reset.

Because the pointer returned by the `DWC_RnkResGetRow` function directly references the receive buffer, it is invalidated when the general ranking library closes. If necessary, follow an escape procedure prior to closing.

11.2.6 Terminating Processes and Error Processing

You can cancel any current asynchronous process by calling the `DWC_RnkCancelProcess` function.

When a process is canceled, the general ranking library enters an error state. Because subsequent processing is not performed once an error occurs, close the general ranking library, and then re-initialize it.

Even when the library is used correctly, a network error may occur due to the issues related to connection quality. Therefore, always denote the appropriate error processes.

11.2.7 Error Codes

The general ranking library uses the `DWC_GHTTP` library for internal communications. Therefore, the DWC error codes specific to communication are set by `DWC_GHTTP`.

11.2.8 Sample Code (Ranking)

The DWC package includes sample code for using the general ranking library.

The initialization data (game name `dwctest`) used by this sample is shared among all developers. Therefore, the ranking data registered on the server can be seen or modified by other developers. In game development, the use of the sample's initialization data should be avoided.

11.2.9 Dependency on DWC_GHTTP

The general ranking library uses the `DWC_GHTTP` library for internal communications. Therefore, the general ranking library initialization and closing functions internally call the initialization and closing APIs for `DWC_GHTTP`, so communication using `DWC_GHTTP` and the general ranking library cannot occur at the same time.

11.2.10 Amount of Memory Used

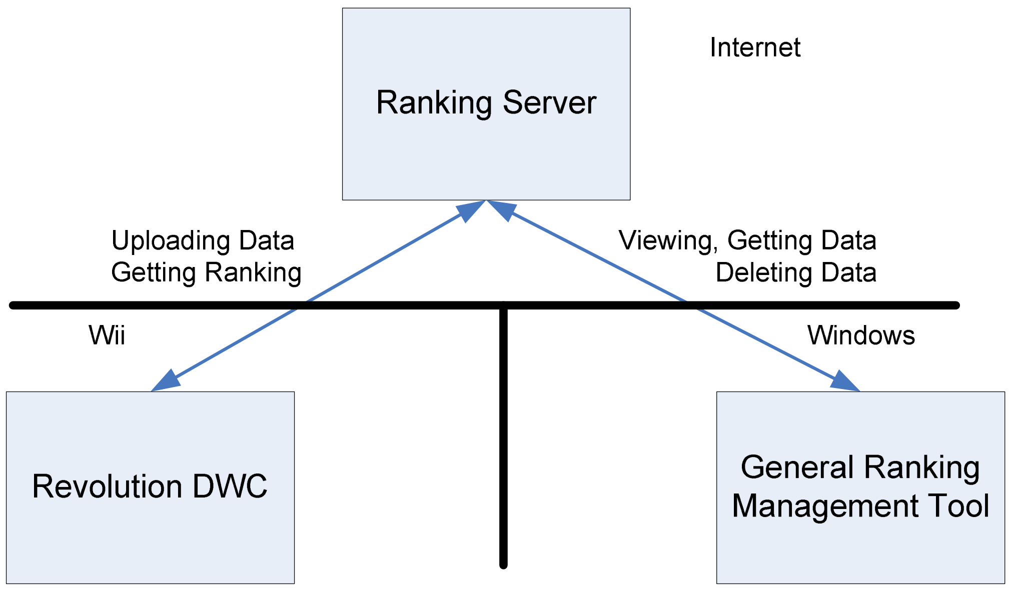
The general ranking library uses the `DWC_Alloc` function for internally allocating memory. For score uploading, the target amount of memory to be allocated equals 2 x (the size of the user-defined data + approximately 200 bytes). For score downloading, the target is about 200 bytes, in addition to the total volume of the score data to be downloaded. (This depends on the size of the user-defined data and on the highest number in the ranking list.)

11.3 General Ranking Management Tool

11.3.1 Overview

General Ranking Management Tool is a Windows application for managing the data uploaded to the ranking server by the DWC general ranking library. It can view, get, and delete data.

Figure 11-2 Structure Diagram



11.3.2 File Structure

The General Ranking Management Tool comprises the following files:

- `DWCRankingAdmin.exe`: the application executable file
- `DWCRankingAdmin.exe.config`: the file in which application settings are saved

11.3.3 Run Environment

11.3.3.1 Install the .NET Framework

Microsoft .NET Framework, Version 2.0 or later must be installed on the computer running the General Ranking Management Tool. If it is not currently installed, install it using Windows Update.

11.3.3.2 Install the Configuration File

An encryption key is needed because the General Ranking Management Tool uses encrypted communications with the ranking server.

The configuration file distributed by Nintendo is `admin_setting.txt`. Place this file in the same folder that contains `DWCRankingAdmin.exe`.

11.3.3.3 Communication Settings

The General Ranking Management Tool communicates with the Internet when it is run on a computer that has Internet access. The General Ranking Management Tool uses Internet Explorer settings as its communication settings.

11.3.4 Communication Load on the Ranking Server

Because the General Ranking Management Tool can be used to get large amounts of data at once, it places a large load on the ranking server. Tool users should be managed on a project basis, with only a few people using it at a time.

To further diminish the load, get data in 100-case blocks when displaying it.

11.3.5 Time Zones for Acquired Data

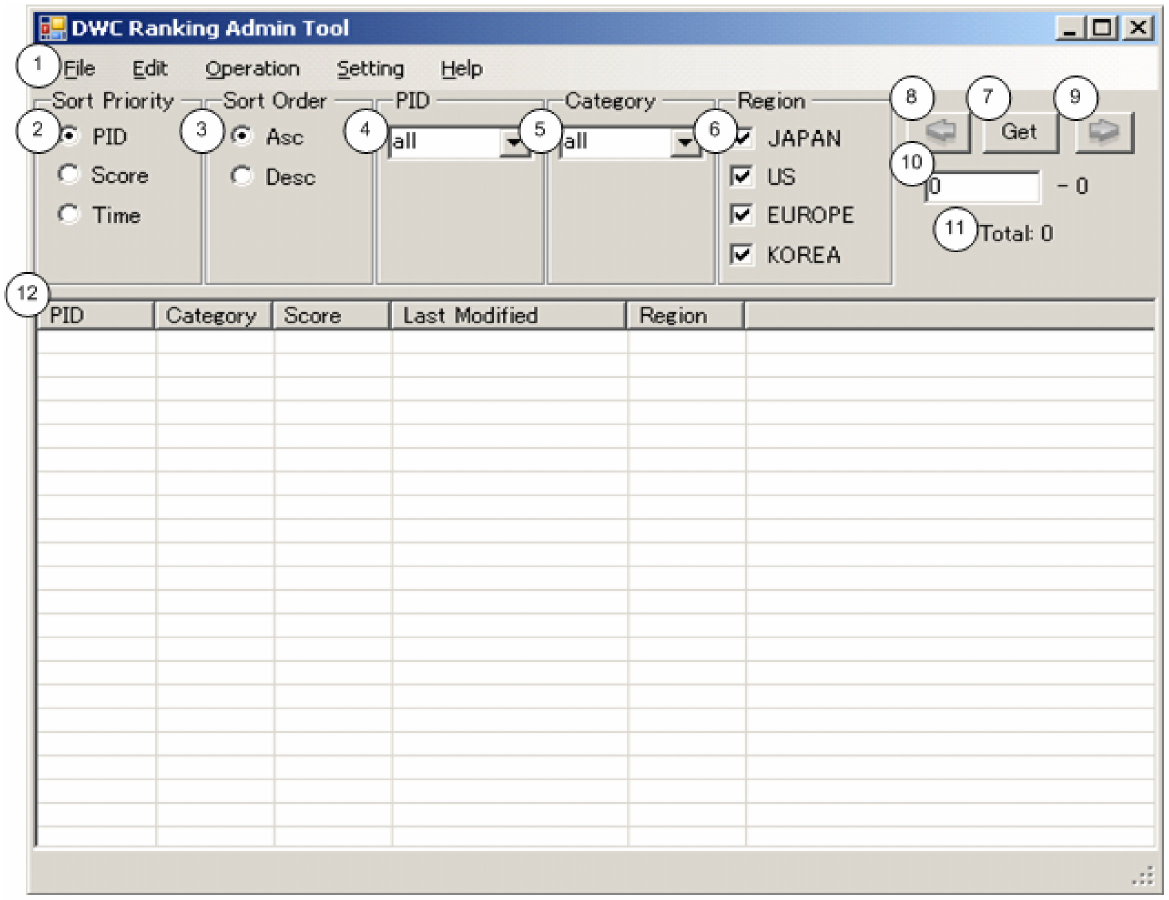
- Time displayed in "Last Modified" in the management tools list
  - ✓ When "use UTC" is checked, displayed in UTC
  - ✓ When "use UTC" is not checked, displayed in the time zone of the local machine
- When obtained with `Get CSV` in the management tool
  - ✓ When "use UTC" is checked, sets the time zone obtained as UTC
  - ✓ When "use UTC" is not checked, sets the time zone obtained as the local time zone
  - ✓ The time in the obtained data is specified by the time zone of the server (PDT or PST)
- Web Service
  - ✓ Sets the time zone obtained in UTC
  - ✓ The time in the obtained data is specified by the time zone of the server (PDT or PST)

11.3.6 Launching the General Ranking Management Tool

Launching `DWCRankingAdmin.exe`. runs the General Ranking Management Tool.

11.3.7 Screen Composition

Figure 11-3 Main Form



The main form (shown in Figure 11-3) displays data and specifies the conditions for that display. The following is a description of the available functionality.

1. Application menus. Menu functions are described in 11.3.8, Menu Composition.

2. Sorting priority of the data. The following choices are available:

   - PID: sort by GS profile ID
   - Score: sort by score
   - Time: sort by the most recently updated time

3. Sorting order. The following choices are available.

   - Asc: sort in the ascending order
   - Desc: sort in the descending order

4. PID of the data to be displayed. You can enter either a decimal numerical value or “all”. For the latter, all PID data is displayed.

5. Category ID for the data to be displayed. You can enter either a decimal numerical value or “all”. For the latter, all category ID data is displayed.
6. Market for the data to be displayed. Each of the markets can be toggled on and off.
    - JAPAN: Japanese market
    - US: North American market
    - EUROPE: European market
    - KOREA: Korean market
7. Data display button. Pressing this button displays the data for the specified conditions in the list in item 12.
8. Previous display button. This button is active, when the offset is greater than zero and the data is displayed. Pressing this button displays the offset data with an offset value that is set to 100 less than the current value.
9. Next display button. This button is active when the displayed data is followed by the ordered data. Pressing the button displays the offset data with an offset value that is set to 100 greater than the current value.
10. Textbox for entering the offset value for the displayed data. Entering a value here displays 100 data entries from that value.
11. Indicates how many data entries there are for the specified conditions. Displaying the data updates this value.
12. The data list. The data retrieved is displayed here as a list.

11.3.8 Menu Composition

11.3.8.1 File Menu

- Exit: closes the application

11.3.8.2 Edit Menu

- Copy: copies to the clipboard any data selected on the list.
- Select All: selects all of the data displayed on the list.

11.3.8.3 Operation Menu

- Get CSV: opens the Get CSV dialog box (see 11.3.9.1 Get CSV Dialog Box).
- Get UserData: opens the Get User Data dialog box (see 11.3.9.2 Get UserData Dialog Box).
- Delete Entry: opens the Delete Entry dialog box (see 11.3.9.3 Delete Entry Dialog Box).

11.3.8.4 Setting Menu

- Use UTC: when activated, displays time in the Coordinated Universal Time (UTC) format.
- Proxy Server Settings: opens the ProxyServerSettings dialog box to make settings for communication via a Proxy server.
- Public Server: when activated, connects to the release product-ranking server.

11.3.8.5 Help Menu

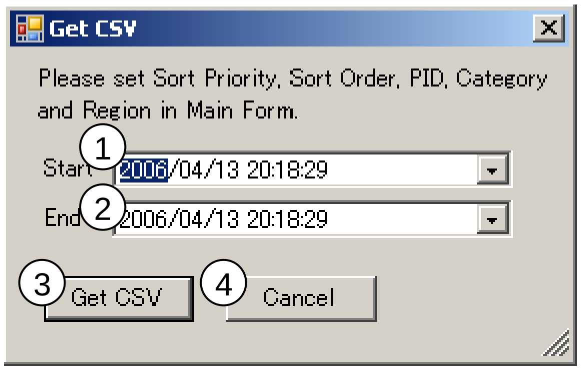
- Version: displays version information

11.3.9 Dialog Boxes

11.3.9.1 Get CSV Dialog Box

Figure 11-4 shows the dialog box for getting the data in a CSV file format.

Figure 11-4 Get CSV Dialog Box


Get CSV
Please set Sort Priority, Sort Order, PID, Category and Region in Main Form.
Start
2006/04/13 20:18:29
End
2006/04/13 20:18:29
Get CSV
Cancel
1
2
3
4

Of the conditions used for getting a CSV file, those for the sorting order, PID, category ID, and market are as specified on the main form.

1. Gets data starting from the specified date and time.
2. Gets data ending with the specified date and time.
3. Begins the process of getting the CSV file.
4. Cancels this operation. Closes the dialog box without getting a CSV file.

Once the data is retrieved, the File Save dialog box opens for you to specify the save location.

A CSV file includes data for each entry row in the following format:

```
100000 (TAB) 10 (TAB) 4 Thu Apr 13 00:38:41 PDT 2006 (TAB) 5758 (TAB) dGVzdCBkYXRhAA==
```

The order is PID, category ID, time last updated, score, and user-defined data in base64 encoding, all of which are delimited by tabs.

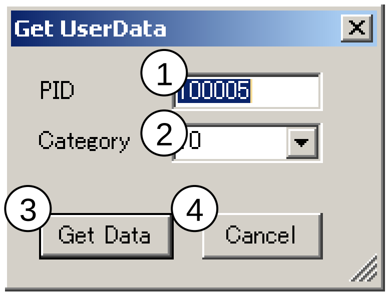
Note: The base64 encoding used in this CSV file replaces “+” with “-“ and “/” with “_”.

You can only get up to 5,000 entries for a single CSV file. A warning appears if you exceed that limit. If this occurs, shorten the time frame, or find some other way to reduce the number of entries, then try again.

If you need many entries, this can take a minute or longer. Because intense processing is required to obtain a CSV, do this at most once every thirty minutes.

11.3.9.2 Get UserData Dialog Box

Figure 11-5 Get UserData Dialog Box



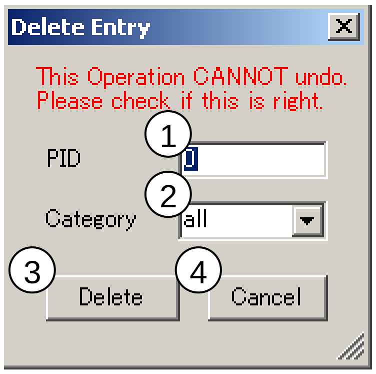
This dialog box is used for getting the user-defined data. Data is saved in a file in binary format.

1. Specify the PID for the data to be retrieved. If the data is already selected in the data list when this dialog box opens, the PID for the selected data will appear here.
2. Specify the category ID for the data to be retrieved. If the data is already selected in the data list when this dialog box opens, the category ID for the selected data will appear here. “All” cannot be specified here.
3. Begins the process of obtaining the user-defined data file.
4. Cancels the request for the user-defined data file and returns to the main form.

Once the data is retrieved, a File Save dialog box opens for specifying the save location.

11.3.9.3 Delete Entry Dialog Box

Figure 11-6 Delete Entry Dialog Box



This Operation CANNOT undo.
Please check if this is right.

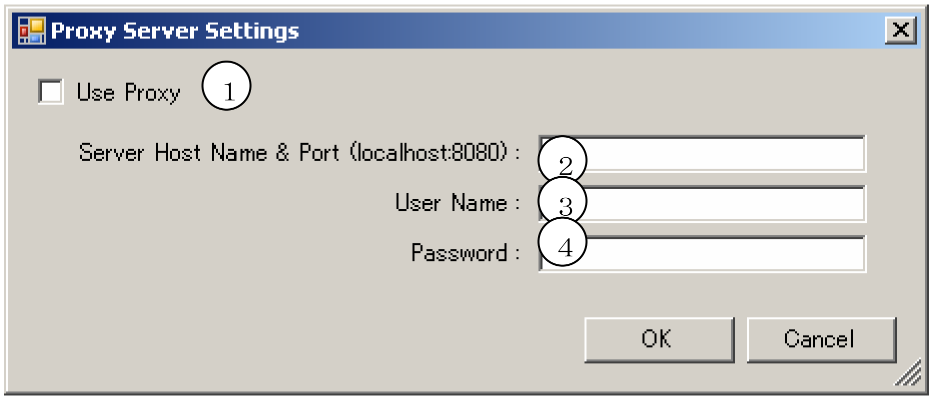
This dialog box is used for deleting data. Be aware that this operation cannot be undone.

1. Specify the PID for the data that you want to delete. If the data is already selected in the data list when this dialog opens, the PID for the selected data appears here.
2. Specify the category ID for the data that you want to delete. If the data is already selected in the data list when this dialog opens, the category ID for the selected data appears here. Specify "all" to delete from all category IDs.
3. Begins the process of data deletion.
4. Cancels the deletion of data and returns to the main form.

When data is deleted, the data displayed in the list is automatically updated.

11.3.9.4 Proxy Server Settings Dialog Box

Figure 11-7 Proxy Server Settings Dialog Box



This dialog box is used for setting the proxy server.

1. If checked, the specified proxy server is used. If not checked, the Internet Explorer settings are used. Because proxy authentication cannot be performed with the Internet Explorer settings, this field must be checked when proxy authentication is required.
2. Specify the proxy server host name and port number.
3. Specify the user name for proxy authentication. Leave blank if proxy authentication is not used.
4. Specify the password for proxy authentication. Leave blank if proxy authentication is not used.

Caution: The user name and password that are set here are stored in the management tool's configuration file and are valid even after restarting. However, security can be an issue because the configuration file is not encrypted. When using the feature, keep this issue in mind until it is addressed in a future revision.

11.3.10 Shortcut Keys

The following shortcut keys can be used with the General Ranking Management Tool.

- CTRL+C: Copies to clipboard the data selected in the list

- CTRL+A: Selects all the data displayed in the list
- CTRL+S: Opens the Get CSV dialog box to get a CSV file
- CTRL+U: Opens the Get UserData dialog box to get user-defined data
- CTRL+D: Opens the Delete Entry dialog box to delete data

11.4 Web Services Development

11.4.1 Web Services

The general ranking library has an interface for getting collected data over the Internet. Use of this functionality permits access to such services as displaying the ranking data on a game Web site.

11.4.2 Using Web Services

11.4.2.1 Access

To get the general ranking library data, access the URLs below. You can then get a tab-delimited CSV file as an HTTP response.

For released products:

```
http://gamestats.gs.nintendowifi.net/[gamename]/web/admin/getcsv.asp
```

For development:

```
http://sdkdev.gamespy.com/games/[gamename]/web/admin/getcsv.asp
```

Note: The gamename is a unique string assigned to each title and is noted on the design statement returned from Nintendo.

11.4.2.2 Security

For security purposes, there are limitations on the IP addresses that can access the general ranking library Web services. When using these services, contact Nintendo with the global IP for the accessing device.

There are no access limitations for the development server.

11.4.2.3 Data Formats

The data available from the general ranking library Web services is in tab-delimited CSV files, with a single entry for each row.

In order, the row elements include the GS profile ID, the category ID, the last time updated, the score, and the user-defined data in base64 encoding.

Note: The base64 encoding used in this CSV file replaces “+” with “-“ and “/” with “_”.

11.4.2.4 Load on the Server

The use of Web services places a large load on the ranking server. Limit getting data to once every 30 minutes, use the `since` get parameter appropriately, and avoid getting the same data twice.

11.4.2.5 Get Parameters

You can specify the get method by providing parameters in the request to the ranking server.

The request parameters have the form of strings appended to the end of the URL.

```
http://…/getcsv.asp?[parameter name 1]=[value 1]&[parameter name 2]=[value 2]&...
```

The parameters and values that can be specified are shown in Table 11-7. You do not have to specify all the parameters. Default values are applied for any omitted parameters.

Table 11-7 Get Parameters

Parameter	Description	Values
`sort`	Specifies the data sort method.	• The default is 0. • 0 sorts the retrieved data in ascending order by score. • 1 sorts the retrieved data in descending order by score.
`region`	Specifies the market for the retrieved data.	The following values are OR'd: The default value is 255 (all markets). 1 = Japan 2 = North America 4 = Europe 8 = Korea
`pid`	Specifies the PID for the retrieved data. If omitted, data is retrieved for all PIDs.	
`category`	Specifies the category ID for the retrieved data. If omitted, data is retrieved for all category IDs.	
`limit`	Specifies the maximum number of entries to get.	1 to 5000. The default is 100.
`since`	• Gets the data only for the dates and times after the specified date and time. • By default, filtering is not applied. • The format is [year]-[month]-[day]-[hour]-[minute]-[second]. • Specify in Coordinated Universal Time (UTC) format.	

For example, to get a maximum of 1,000 entries with a category ID of 10 updated after 13:00 on April 1, 2006, specify the following parameters.

```
http://…/getcsv.asp?category=10&limit=1000&since=2006-4-1-13-0-0
```

12 Download Service

12.1 Introduction

This chapter provides details specific to Nintendo Wi-Fi Connection Download Service, which refers to content management from a PC through the web and content download from the Wii console using the DWC Download library.

Nintendo Wi-Fi Connection Download Service provides the following capabilities:

- Secure communications using HTTPS
- Extraction of files by adding properties
- Adding of descriptive text for downloadable games
- Specifying a downloadable date and time
- Restricting downloadable access points
- Registering files up to 3 MB in size
- Registering up to 100 separate content items

To use Nintendo Wi-Fi Connection Download Service, take the following steps. First, be sure to contact Nintendo at support@noa.com about the use of Nintendo Wi-Fi Connection Download Service:

- Submit an application for using Nintendo Wi-Fi Connection Download Service
- Obtain a game password and a Game Code for accessing the download server from a Wii console
- Obtain a URL for the Contents Management screen, an account name, and an administrative password

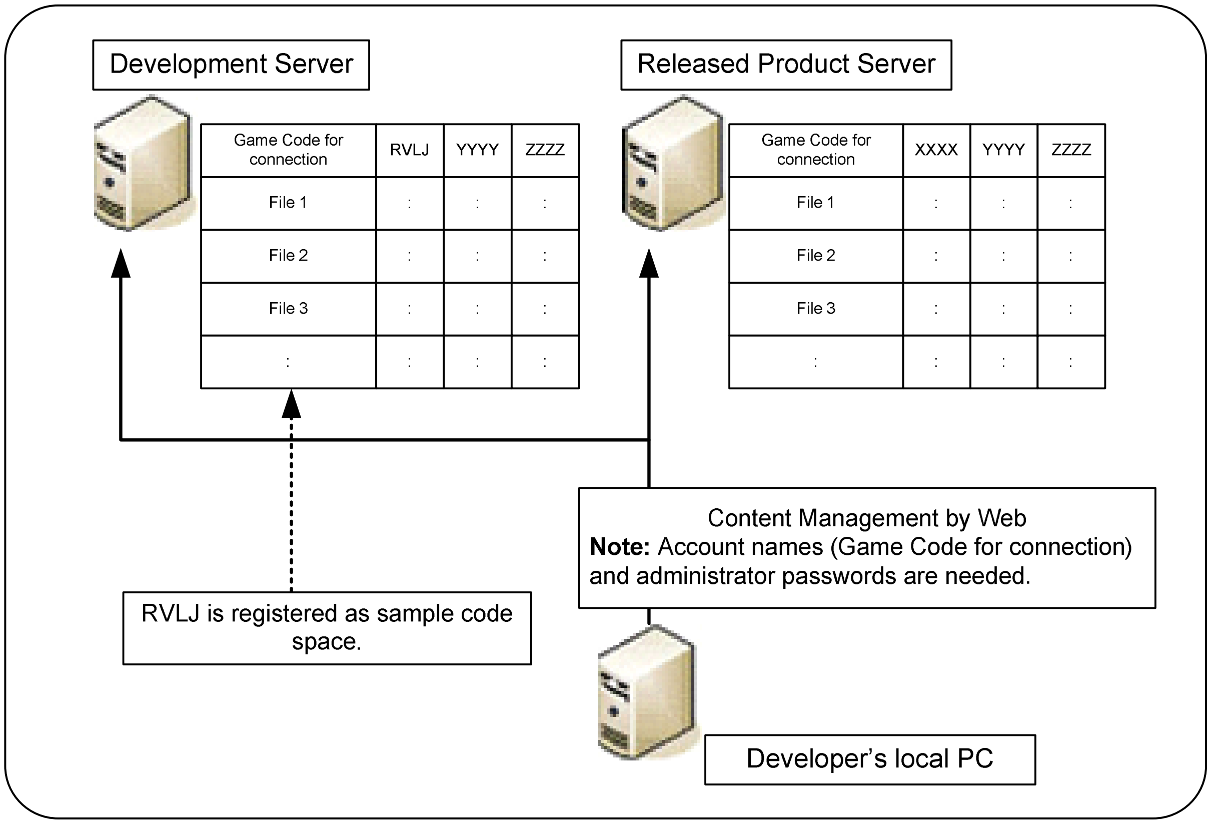
12.2 Overview

Nintendo Wi-Fi Connection Download Service operates on the following content.

12.2.1 Structure of the Download Server

As shown in Figure 12-1, two types of download servers are used for content registration: the development and the released product servers. Development and debugging takes place on the development server, while a released product's ROM uses the released product server. On the client side, switching of the download server is linked to the settings in the `DWC_Init` function.

Figure 12-1 Structural Diagram of the Server

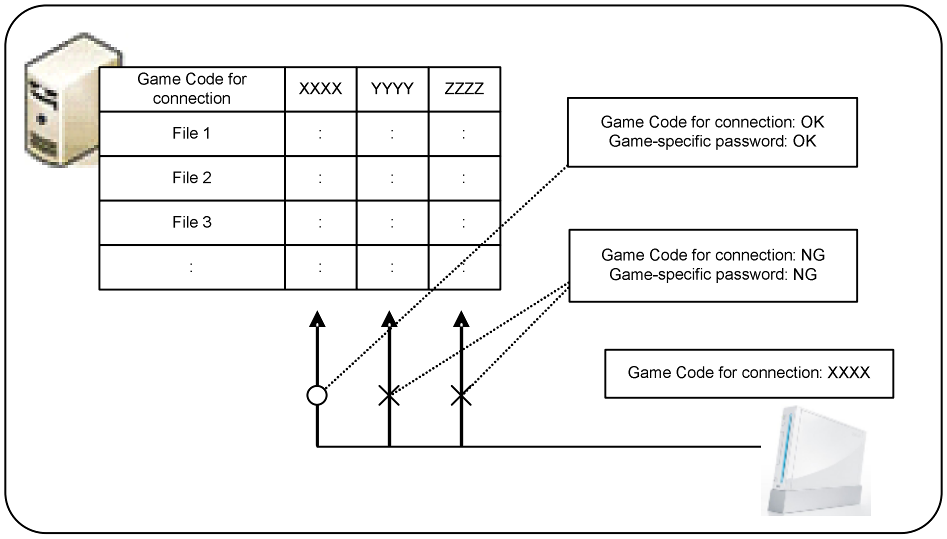

Development Server
Released Product Server
Game Code for connection
RVLJ
YYYY
ZZZZ
XXXX
File 1
File 2
File 3
Content Management by Web
Note: Account names (Game Code for connection) and administrator passwords are needed.
RVLJ is registered as sample code space.
Developer's local PC

After applying to use the Nintendo Wi-Fi Connection Download Service, the disk space and an administrative screen are assigned for each game on every server. Content registration can be completed over the Internet by connecting to the Admin screen. However, this requires the URL, account name, and administrator password provided by Nintendo.

12.2.2 Content Confidentiality

Content is divided by the connection game code for each of the registered disk spaces. However, when the client connects to the download server, access is controlled with a game-specific password to maintain the confidentiality of each game. Nintendo provides the password after receiving your application to use Nintendo Wi-Fi Connection Download Service.

Figure 12-2 Password-Enabled Access Limitations


Game Code for connection
XXXX
YYYY
ZZZZ
File 1
File 2
File 3
Game Code for connection: OK
Game-specific password: OK
Game Code for connection: NG
Game-specific password: NG
Game Code for connection: XXXX

Note: When different titles share the same disk space, one of the connection Game Codes and game-specific passwords is shared.

12.2.3 Content Attributes

Content attributes refer to the up to three attributes that can be added to the content registered on the download server. Correspondingly, the same get attributes can be specified on the client end, which could then be called the get file attributes. When downloading or getting file lists, a comparison of the content attributes and the get file attributes can determine if the get operation is possible. As a result, controlling these attributes lets you filter out files or use the get restrictions.

Comparisons take place on the following content.

- Gets can occur when the content and get file attributes match exactly
- Unconditional gets are possible when the get file attributes are NULL strings

Table 12-1 provides some examples of how content attributes are combined to determine what can be downloaded. Note that the two adjacent quotes, “”, in the table denote the NULL string.

Table 12-1 Sample Attribute Comparisons

		Attribute 1	Attribute 2	Attribute 3	Results
Ex. 1	Content attributes	“A”	“B”	“C”	Complete match; get is possible.
	Get file attributes	“A”	“B”	“C”	
Ex. 2	Content attributes	“A”	“B”	“C”	Attributes 1, 2, and 3 are ignored; get is possible.
	Get file attributes	“”	“”	“”	

		Attribute 1	Attribute 2	Attribute 3	Results
Ex. 3	Content attributes	“A”	“B”	“C”	Attribute 1 is ignored; get is possible because attributes 2 and 3 match.
	Get file attributes	“”	“B”	“C”	
Ex. 4	Content attributes	“A”	“B”	“C”	Attributes 1 and 3 are ignored, but get is not possible because attribute 2 does not match.
	Get file attributes	“”	“1”	“”	
Ex. 5	Content attributes	“”	“B”	“C”	Get is not possible because attribute 1 does not match. When a content attribute is blank, the matching get file attribute must also be blank.
	Get file attributes	“A”	“B”	“C”	

12.2.4 DWC Download Library Functionality

The functionality available in the DWC Download library (the set of functions whose names begin with `DWC_nd`) is as follows.

- Setting get file attributes
- Getting the number of files
- Getting the file list
- Downloading files
- Checking on download progress

12.2.5 Demo Programs

The connection game code for the demo program is “RVLJ”, and there is content for the demo program on the development server (see Table 12-2). Configure the Internet setting for the client system before launching the demo program.

As soon as the demo program is launched, the connection operation to the Internet will be started. The get file attributes are then set with `DWC_NdSetAttr`. Once set, a file list can be retrieved from the server by comparing attributes to see which can be downloaded.

When one of the retrieved files is selected, its download begins.

Table 12-2 Content for Demo Programs

File Name	File Size (in Bytes)	Attribute 1	Attribute 2	Attribute 3
`64k_1.txt`	65536	a		
`64k_2.txt`	65536	a	b	
`128k_1.txt`	131072	a	b	c
`128k_2.txt`	131072	b		

File Name	File Size (in Bytes)	Attribute 1	Attribute 2	Attribute 3
`256k_1.txt`	262144	b	b	
`256k_2.txt`	262144	b	b	c
`512k_1.txt`	524288	c		
`512k_2.txt`	524288	c	b	
`1024k_1.txt`	1048576	c	b	c
`1024k_2.txt`	1048576	aaaaaaaaaa	bbbbbbbbbb	cccccccccc

12.3 Connecting to the Nintendo Wi-Fi Connection

If you are using the download-specific package (DWC-DL), it is not necessary to build the user data or a friend relationship. Therefore, the processes for connecting to the Internet are simplified, compared to the standard DWC package. Use this section as a reference when DWC-DL is used to connect to Nintendo Wi-Fi Connection.

12.3.1 Initialization

You must initialize DWC-DL with the `DWC_Init` function as you would for the normal package.

12.3.2 Creating User Data

There is no need to create user data for DWC-DL.

12.3.3 Logging in to the Nintendo Authentication Server

When logging in to the Nintendo authentication server, use the `DWC_NASLoginAsync` and not the `DWC_LoginAsync` function. (Initialization of matchmaking or friend relationships with the `DWC_InitFriendsMatch` function is unneeded.) After calling this function, call the `DWC_NASLoginProcess` function once every game frame to advance to login process. Login is complete when the `DWC_NASLoginProcess` function has a return value of `DWC_NAL_STATE_SUCCESS`.

12.3.4 Monitoring Communication Status

Communication status is monitored with the `DWC_ProcessFriendsMatch` function for the normal DWC package. It cannot be used for DWC-DL because it includes communication features specific to matchmaking and friend relationships.

For DWC-DL, the `DWC_NdProcess` function is used to monitor communication status.

Once DWC-DL is initialized with `DWC_NdInitAsync`, call this function once each game frame.

12.3.5 Disconnecting from the Internet

The disconnection process is the same as for the normal DWC package.

12.3.6 Close Process

Once DWC-DL is closed through the `DWC_NdCleanupAsync` function, disconnection will be performed the same way as with the standard DWC package.

12.4 Downloads

12.4.1 Initialization

After connecting to Nintendo Wi-Fi Connection and completing the authentication process, call the `DWC_NdInitAsync` function and initialize the DWC Download library (the set of functions beginning with `DWC_nd`) (see Code 12-1).

Because the initialization process will perform HTTP communications in the background, make sure to give enough processing time to threads with lower priority than the main thread. After calling the `DWC_NdInitAsync` function, call the `DWC_NdProcess` function about once every game frame. Once the initialization process is completed, the `DWC_NdProcess` function will return `DWC_ND_STATE_COMPLETE`.

Code 12-1 Initializing the DWC Download Library

```
char gamecd[] = “RVLJ”;          // The connection game code
char passwd[] = “ABCDEF”;        // The game-specific password provided by Nintendo
// Go forward with the initialization process
BOOL process_nd( void )
{
   while(TRUE)
   {
      switch(DWC_NdProcess())
      {
      case DWC_ND_STATE_BUSY:
        break;
      case DWC_ND_STATE_COMPLETE:
        return TRUE;
      case DWC_ND_STATE_ERROR:
        return FALSE;
      }
      Wait_for_vblank(); // Wait for V-Blank
   }
}
void init_dwc_nd( void )
{
   if ( !DWC_NdInitAsync( gamecd, passwd ) )
   {
      error();           // error processing
      return;
   }
```

```
    if ( !process_nd() )
    {
        error();           // error processing
        return;
    }
}
```

12.4.2 Extracting Files with Attribute Specifications

If you specify the get file attributes with the `DWC_NdSetAttr` function, you can choose the files to download (see Code 12-2). Selection takes place with three attribute strings. However, if no attributes are assigned, selection does not occur and all files are seen as downloadable. The attribute string must be 10 characters or fewer, and null-terminated.

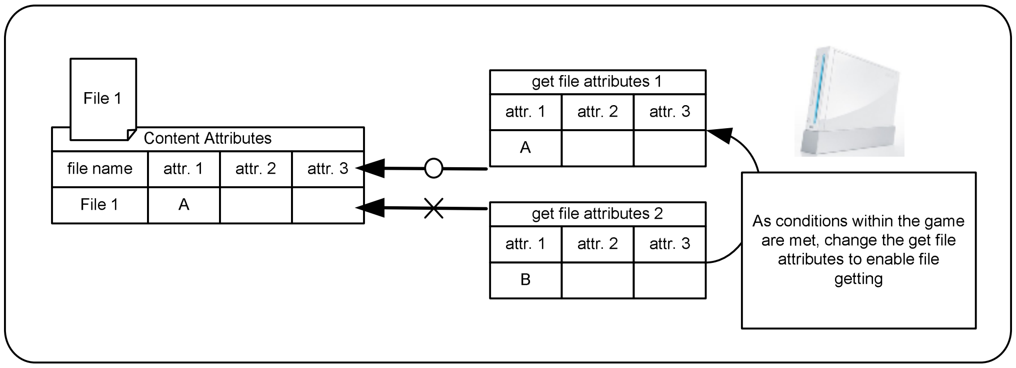
Code 12-2 Specifying Attributes

```
char attr1[] = “A”;
char attr2[] = “B”;
char attr3[] = “C”;

void set_attr( void )
{
    if ( DWC_NdSetAttr( attr1, attr2, attr3 ) == FALSE )
    {
        error();           // error processing
    }
}
```

You can also use this selection to apply certain acquisition restrictions (for example, allowing file downloads after an event in the game, or allowing downloads at level 10 or higher.)

Figure 12-3 Get Limitations Based on Get File Attributes



12.4.3 File Downloads

During the file download, a call to the `DWC_NdGetFileListNumAsync` function gets the total number of downloadable files. Similarly, a call to the `DWC_NdGetFileListAsync` function gets either some or all of the downloadable file lists. The file name, game description, attributes, and file size are stored in the file list, so you can clearly indicate to the user which files can be downloaded.

Start the file download as follows (see Code 12-3):

1. Specify the `DWCNdFileinfo` file information structure as an argument to the `DWC_NdGetFileAsync` function, which indicates the file that you want to download from the file list that was acquired.
2. Call the `DWC_NdGetFileAsync` function.

Code 12-3 File Downloads

```
DWCNdFileInfo info[FILE_NUM};
char buffer[1024*1024];
// Perform Download Process
int get_file( void )
{
      int no, num;

   // Obtain File Count
   if( !DWC_NdGetFileListNumAsync( &num ) )
   {
      return FALSE;
   }
   if( !ProcessNd() )
   {
      return FALSE;
   }

   // Obtain File List
   if( !DWC_NdGetFileListAsync( info, 0, num ) )
   {
      return FALSE;
   }
   if( !ProcessNd() )
   {
      return FALSE;
   }

   // Select File to be Obtained
   no = select_download_file( num );
```

```
    // Obtain File
    if( !DWC_NdGetFileAsync( &info[ no ], buffer, sizeof( buffer )) )
    {
        return FALSE;
    }
    if( !ProcessNd() )
    {
        return FALSE;
    }
    // Download Complete
    return TRUE;
}
```

12.4.4 Cancel Processing

You can cancel file count acquisition, file list acquisition, and file download processes by calling the `DWC_NdCancelAsync` function (see Code 12-4).

Code 12-4 Cancel Processing

```
bool cancel;
BOOL process_nd( void )
{
    int             errorCode;
    DWCErrorType    errorType;
    while(TRUE)
    {
        switch(DWC_NdProcess())
        {
        case DWC_ND_STATE_BUSY:
            // Cancel Process
            if( cancel )            // A user cancel request yields TRUE
            {
                if( !DWC_NdCancelAsync() )
                {
                    error();        // Error Process
                }
                break;
            }
            break;
        case DWC_ND_STATE_COMPLETE:
            return TRUE;
        case DWC_ND_STATE_ERROR:
            if(DWC_GetLastErrorEx( &errorCode, &errorType ) != DWC_ERROR_NONE)
            {
```

```
            if( errorCode == (DWC_ECODE_SEQ_ADDINS + DWC_ECODE_FUNC_ND +
                             DWC_ECODE_TYPE_ND_CANCEL) )
            {
               canceled();      // Process at Cancellation
            }
            else
            {
               error();         // Error Process
            }
         }
         // Clear Error
         DWC_ClearError();
         return FALSE;
      }
      Wait_for_vblank();        // Wait for V-Blank
   }
}
```

When the `DWC_NdCancelAsync` function's return value is TRUE, cancel processing begins. Afterward, the `DWC_ECODE_TYPE_ND_CANCEL(-31040)` error code will be set when the `DWC_NdProcess` function returns `DWC_ND_STATE_ERROR`. After performing proper operations use the `DWC_ClearError` function to clear the errors. The initialization and cleanup processes cannot be cancelled.

12.4.5 Progress Level Checking

You can check the download progress by calling the `DWC_NdGetProgress` function during file downloads (see Code 12-5).

Code 12-5 Confirming Download Progress Level

```
void check_progress( void )
{
   u32 received,contentlen;

   if ( DWC_NdGetProgress( &received, &contentlen ) == TRUE )
   {
      printf( "Download %d/100\n", ( received*100)/contentlen ));
   }
}
```

12.4.6 Termination

To close the DWC Download library, call the `DWC_NdCleanupAsync` function.

After the completion of this operation, the `DWC_NdProcess` function will return `DWC_ND_STATE_COMPLETE` (see Code 12-6).

Code 12-6 Termination

```
BOOL cleanup;

BOOL process_nd( void )
{
   while(TRUE)
   {
      switch(DWC_NdProcess())
      {
      case DWC_ND_STATE_COMPLETE:
         return TRUE;
      }
      Wait_for_vblank(); // Wait for V-Blank
   }
}

void cleanup_dwc_nd ( void )
{
   DWC_NdCleanupAsync();

   ProcessNd();
}
```

12.5 Contents Management

12.5.1 Connecting to the Nintendo Wi-Fi Connection Download Server Management Screen

Connect to the Nintendo Wi-Fi Connection Download Server Management screen as follows:

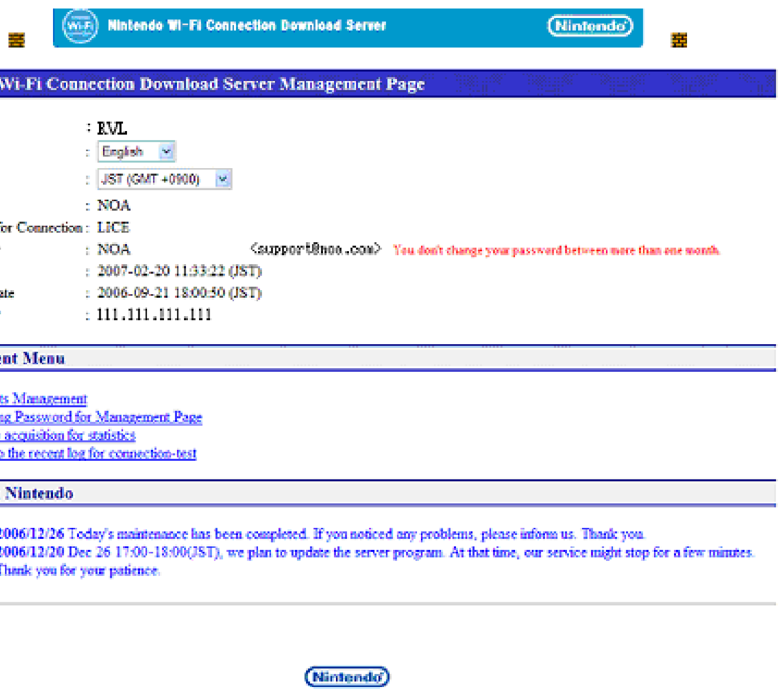
1. Using a Web browser on a PC, go to the URL sent by Nintendo.
2. At the authentication screen, enter the account name and administrator password issued by Nintendo.

The administrator password can be changed from the Download Server Management screen.

12.5.2 Download Server Management Screen

The screen in Figure 12-4 is displayed when you log in to the Download Server Management screen.

Figure 12-4 Download Server Management Screen



Nintendo Wi-Fi Connection Download Server Management Page

Model	: RVL
Language	: English
Time Zone	: JST (GMT +0900)
Game Name	: NOA
Game Code for Connection	: LICE
Administrator	: NOA <support@noo.com> You don't change your password between more than one month.
Now Date	: 2007-02-20 11:33:22 (JST)
Last Login Date	: 2006-09-21 18:00:50 (JST)
Last Login IP	: 111.111.111.111

Management Menu

- Contents Management
- Changing Password for Management Page
- Log file acquisition for statistics
- Refer to the recent log for connection-test

News from Nintendo

- 2006/12/26 Today's maintenance has been completed. If you noticed any problems, please inform us. Thank you.
- 2006/12/20 Dec 26 17:00-18:00(JST), we plan to update the server program. At that time, our service might stop for a few minutes. Thank you for your patience.

12.5.2.1 Information

The following information is shown on the Download Server Management screen.

- Model: the model of the target machine is displayed. RVL represents Wii and NTR represents DS
- Language: select the language (either Japanese or English) to be used
- Time Zone: select your current time zone to be used for the management screen
- Game Name: the target gamename is displayed
- Game Code for Connection: the target connection Game Code is displayed
- Administrator: the registered administrator name and e-mail address are displayed
- Current Time: current date and time
- Last Login: the date and time of the last login are displayed in the set time zone
- Last Login IP: the IP address from the prior login is displayed

12.5.2.2 Admin Menu

The following is the content of the Admin menu.

- Contents Management: a link to the Contents Management screen
- Change Admin Password: a link to the Change Admin Password screen
- Get Statistical Log File: a link to the Get Statistical Log File screen

Note: This is only available for the production server.

- Nearest Log Reference: a link to the Nearest Log Reference screen for Connection Tests

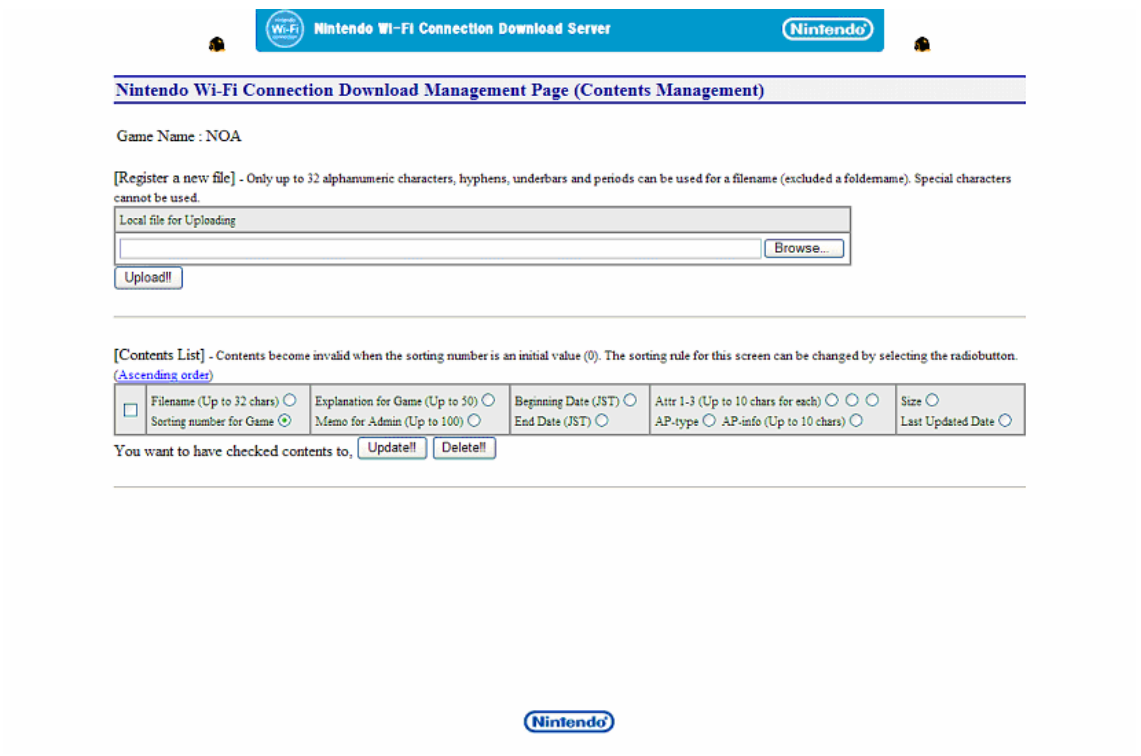
12.5.2.3 News from Nintendo

Notifications from Nintendo are displayed here.

12.5.3 Contents Management Screen

You can register, confirm, and configure downloadable content on the Contents Management screen.

Figure 12-5 Contents Management Screen



Nintendo Wi-Fi Connection Download Management Page (Contents Management)

Game Name : NOA

[Register a new file] - Only up to 32 alphanumeric characters, hyphens, underbars and periods can be used for a filename (excluded a foldername). Special characters cannot be used.

Local file for Uploading

[Contents List] - Contents become invalid when the sorting number is an initial value (0). The sorting rule for this screen can be changed by selecting the radiobutton.
(Ascending order)

☐	Filename (Up to 32 chars) ○ Sorting number for Game ◉	Explanation for Game (Up to 50) ○ Memo for Admin (Up to 100) ○	Beginning Date (JST) ○ End Date (JST) ○	Attr 1-3 (Up to 10 chars for each) ○ ○ ○ AP-type ○ AP-info (Up to 10 chars) ○	Size ○ Last Updated Date ○

You want to have checked contents to,

12.5.3.1 Register a New File

This is used to register downloadable content. Either enter the file under File or click Browse… to select a target file on your hard drive. Next, click Upload to register the content.

Note: Do not upload more than 100 files, and do not exceed a file size of 3 MB.

12.5.3.2 Contents List

A list of registered content is displayed. Registered content can be configured and deleted here. To change settings or to delete content, select the checkbox to the left of the target content, and click Update or Delete.

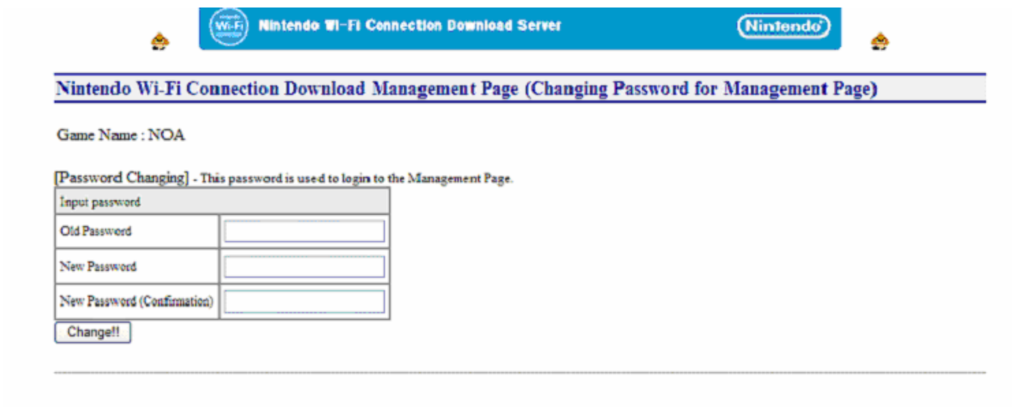
Table 12-3 Description of Fields on the Contents Management Screen

Field	Description
Filename	File name of 32 or fewer characters, referenced by Wii.
Game Sort Number	List sort order (ascending) reflected when the file list was retrieved. If zero is specified, the content is invalidated.
Game Description	Description of 50 or fewer characters that can be retrieved with the file list. Referenced as a UTF16LE string.
Admin Notes	Space for the administrator to add notes.
Beginning Date (JST) End Date (JST)	• Valid period for the content. Set as "YYYY-MM-DD HH:MM:SS" • This is the expiration date of the content/contents **Note:** If "0000-00-00 00:00:00" is specified, it is treated as if no value were specified. • When this is specified as the start date, the valid period will be "unspecified ~ end date" • When this is specified as the end date, the valid period will be "start date ~ unspecified"
Attributes	Content attributes, used to filter file lists and file getting (see 12.2.3 Content Attributes).
Size	Size of the registered file.
Last Updated Date	Date and time of the last content update.

12.5.4 Change Admin Password Screen

The password for the download server administrator can be changed.

Figure 12-6 Change Administrator Password Screen



Nintendo Wi-Fi Connection Download Management Page (Changing Password for Management Page)

Game Name : NOA

[Password Changing] - This password is used to login to the Management Page.

Input password	
Old Password	
New Password	
New Password (Confirmation)	

Change!!

12.5.4.1 Changing Passwords

To change passwords, use the following procedure:

1. In the Old Password field, enter the current password.
2. In the New Password field, enter the new desired password.

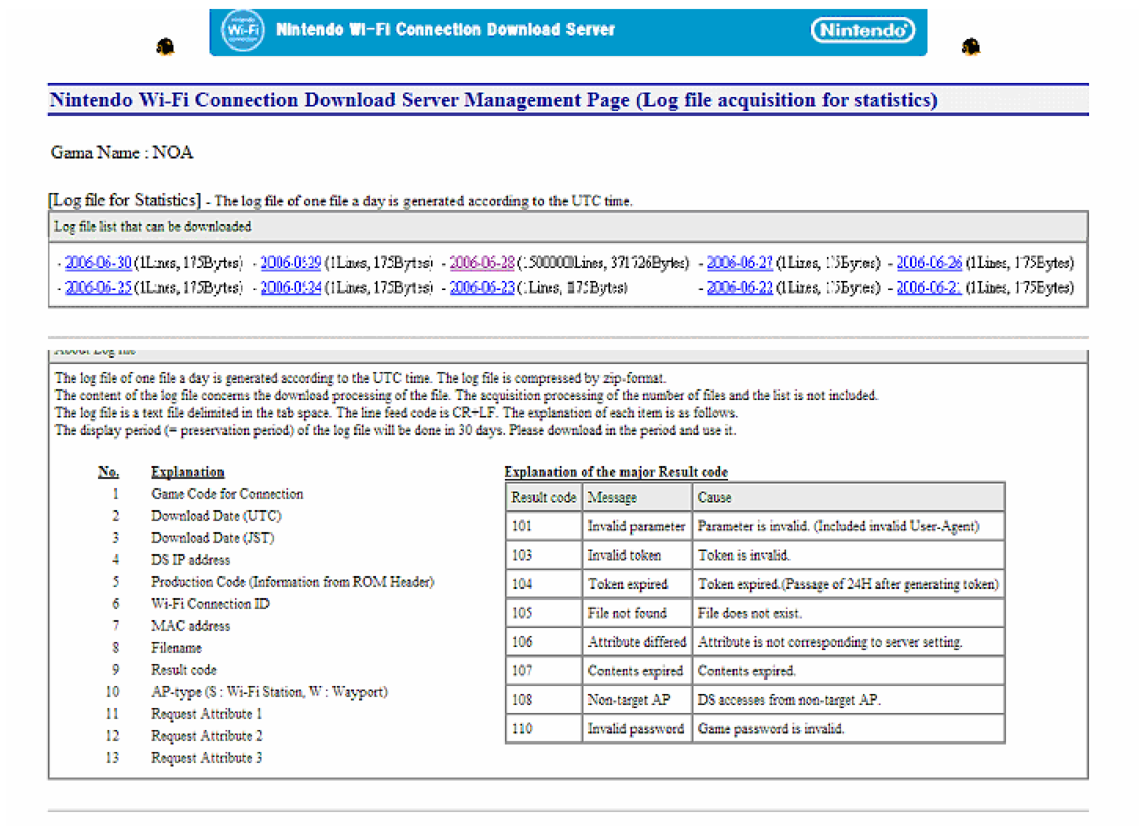
3. In the New Password (Confirmation) field, enter the new password one more time to confirm it.
4. Click Change to change your password.

12.5.5 Get Statistical Log File Screen

From the Get Statistical Log File screen, you can get the download server access log (a tab-delimited text file that represents a single day).

Note: This page can only be used for released product servers, not for development servers.

Figure 12-7 Get Statistical Log File Screen



Nintendo Wi-Fi Connection Download Server Management Page (Log file acquisition for statistics)

Gama Name : NOA

[Log file for Statistics] - The log file of one file a day is generated according to the UTC time.

Log file list that can be downloaded

- 2006-06-30 (1Lines, 175Bytes) - 2006-06-29 (1Lines, 175Bytes) - 2006-06-28 (1500000Lines, 371726Bytes) - 2006-06-27 (1Lines, 175Bytes) - 2006-06-26 (1Lines, 175Bytes)
- 2006-06-25 (1Lines, 175Bytes) - 2006-06-24 (1Lines, 175Bytes) - 2006-06-23 (1Lines, 175Bytes) - 2006-06-22 (1Lines, 175Bytes) - 2006-06-21 (1Lines, 175Bytes)

The log file of one file a day is generated according to the UTC time. The log file is compressed by zip-format.
The content of the log file concerns the download processing of the file. The acquisition processing of the number of files and the list is not included.
The log file is a text file delimited in the tab space. The line feed code is CR+LF. The explanation of each item is as follows.
The display period (= preservation period) of the log file will be done in 30 days. Please download in the period and use it.

No.	Explanation
1	Game Code for Connection
2	Download Date (UTC)
3	Download Date (JST)
4	DS IP address
5	Production Code (Information from ROM Header)
6	Wi-Fi Connection ID
7	MAC address
8	Filename
9	Result code
10	AP-type (S : Wi-Fi Station, W : Wayport)
11	Request Attribute 1
12	Request Attribute 2
13	Request Attribute 3

Explanation of the major Result code

Result code	Message	Cause
101	Invalid parameter	Parameter is invalid. (Included invalid User-Agent)
103	Invalid token	Token is invalid.
104	Token expired	Token expired.(Passage of 24H after generating token)
105	File not found	File does not exist.
106	Attribute differed	Attribute is not corresponding to server setting.
107	Contents expired	Contents expired.
108	Non-target AP	DS accesses from non-target AP.
110	Invalid password	Game password is invalid.

12.5.5.1 Log File for Statistics

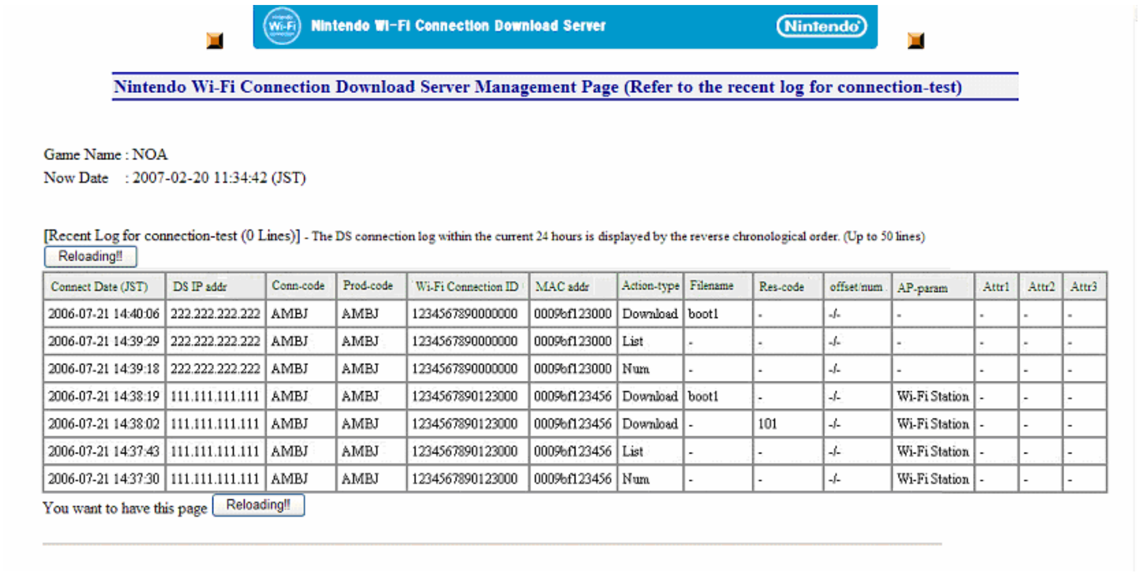
You can get a log file by clicking the dates displayed in the downloadable log file list.

Note: The content of the log file differs from that described in 12.5.6, Nearest Log Reference Screen for Connection Tests, and is only for download processes. The retrieval of file lists and files is not logged.

12.5.6 Nearest Log Reference Screen for Connection Tests

In the Nearest Log Reference screen for Connection Tests, you can see the log information for the 50 most recent server accesses.

Figure 12-8 Nearest Log Reference Screen for Connection Testing


Nintendo Wi-Fi Connection Download Server

Nintendo Wi-Fi Connection Download Server Management Page (Refer to the recent log for connection-test)

Game Name : NOA
Now Date : 2007-02-20 11:34:42 (JST)

[Recent Log for connection-test (0 Lines)] - The DS connection log within the current 24 hours is displayed by the reverse chronological order. (Up to 50 lines)

Reloading!!

Connect Date (JST)	DS IP addr	Conn-code	Prod-code	Wi-Fi Connection ID	MAC addr	Action-type	Filename	Res-code	offset/num	AP-param	Attr1	Attr2	Attr3
2006-07-21 14:40:06	222.222.222.222	AMBJ	AMBJ	1234567890000000	0009bf123000	Download	boot1	-	-/-	-	-	-	-
2006-07-21 14:39:29	222.222.222.222	AMBJ	AMBJ	1234567890000000	0009bf123000	List	-	-	-/-	-	-	-	-
2006-07-21 14:39:18	222.222.222.222	AMBJ	AMBJ	1234567890000000	0009bf123000	Num	-	-	-/-	-	-	-	-
2006-07-21 14:38:19	111.111.111.111	AMBJ	AMBJ	1234567890123000	0009bf123456	Download	boot1	-	-/-	Wi-Fi Station	-	-	-
2006-07-21 14:38:02	111.111.111.111	AMBJ	AMBJ	1234567890123000	0009bf123456	Download	-	101	-/-	Wi-Fi Station	-	-	-
2006-07-21 14:37:43	111.111.111.111	AMBJ	AMBJ	1234567890123000	0009bf123456	List	-	-	-/-	Wi-Fi Station	-	-	-
2006-07-21 14:37:30	111.111.111.111	AMBJ	AMBJ	1234567890123000	0009bf123456	Num	-	-	-/-	Wi-Fi Station	-	-	-

You want to have this page Reloading!!

12.5.6.1 Nearest Log for Connection Testing

The following is the content of the fields:

- Connection Date/time (JST): date and time of the connection
- Wii IP Address: IP address for the connected Wii console
- Connection Code: the connection Game Code provided by Nintendo to the download server
- Product Code: Game Code provided by Nintendo
- Nintendo Wi-Fi Connection ID: displays the Nintendo Wi-Fi Connection ID for the connected Wii console
- MAC Address: unique MAC address for the connected Wii console
- Action Type: “Download” indicates a file download process, “List” – getting a file list, and “Num” – getting the number of files
- Filename: displays the name of the downloaded file
- Result Code: error message returned by the server; for more information, see the description of the major result codes in Figure 12-7
- offset/num: parameter for getting file lists
- Attribute 1: get file attribute 1
- Attribute 2: get file attribute 2
- Attribute 3: get file attribute 3
- Country Code: country code of the connected Wii console. This will be a two-digit alphabet that follows the ISO3166 standardization.
- Region: the market code for the connected Wii console (00 – JPN, 01 – North America, and 02 – EUR)

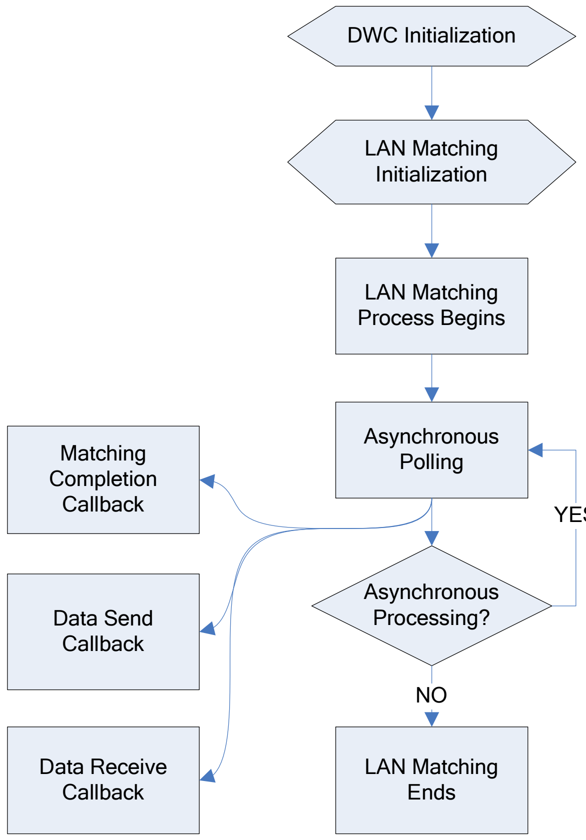
13 Intra-LAN Matchmaking

This chapter is a description of intra-LAN matchmaking (hereafter, LAN matchmaking). LAN matchmaking is a feature provided for possible testing purposes, allowing the construction of simple mesh P2P networks on a LAN without using a server.

The P2P networks created with LAN matchmaking offer a stable communication environment, because there is no Internet connection. Furthermore, by introducing a router on which packet delays and drops can be managed, you can also test various other communication environments.

13.1 Process Flow

Figure 13-1 LAN Matching Process Flow



13.1.1 Initializing a LAN Match

Call the `DWC_InitLanMatch` function to initialize a LAN match, then call the `DWC_SetRecvLanMatchCallback` and `DWC_SetSendLanMatchCallback` functions to specify the callback functions called when sending and receiving data.

Code 13-1 Initializing a LAN Match

```
// LanMatch initialization
DWC_InitLanMatch( &cnt );

// Set the send/receive callbacks
DWC_SetRecvLanMatchCallback( recvCallback );
DWC_SetSendLanMatchCallback( sendCallback );
```

13.1.2 Starting a LAN Match

Call the `DWC_StartLanMatch` function to begin matchmaking. The following are specified as arguments:

- Number of peers to match
- Callback function to call when matchmaking is complete

LAN matchmaking does not complete until connections are established for the number of peers specified here.

The `DWC_StartLanMatch` function is asynchronous. Calling this function begins the asynchronous LAN matchmaking process. Asynchronous polling is performed with the `DWC_ProcessLanMatch` function. This process is bound to the sending and receiving of the P2P data. Thus, after the process that was initiated with `DWC_StartLanMatch` begins, continue polling with the `DWC_ProcessLanMatch` function until the P2P communication has completed.

Code 13-2 Starting a LAN Match

```
// Begin matchmaking
DWC_StartLanMatch( CONNECTION_NUM, matchedCallback );

// main loop
while (1) {

        DWC_ProcessLanMatch();

        :

}
```

13.1.3 Matchmaking Completion

When matchmaking is complete, the callback function specified in the `DWC_StartLanMatch` function is called.

Code 13-3 Matchmaking Completion Callback

```
void matchedCallback( DWCLanMatchResult result )
{

        if ( result == DWC_SUCCESS ) {
                    // LAN match complete
        } else {
                    // LAN match failure
        }

}
```

13.1.4 Sending and Receiving Data

Once the matchmaking is complete, data can be sent among all peers by calling the `DWC_SendLanMatch` function.

To specify a peer for communication, an identifier called an AID is used. AIDs are numerical values assigned from zero in order; for example, 0, 1, 2 and 3 after matching 4 players. The assignment of AIDs to clients follows no particular order. For more information on AIDs, see paragraph 8.1.

You can get your own AID by using the `DWC_GetMyAIDLanMatch` function.

The total number of peers determined with matchmaking is retrieved with the `DWC_GetConnectNumLanMatch` function.

The following sample sends data to all peers except you.

Code 13-4 Sending Data to All Peers in LAN Matchmaking

```
for ( aid = 0; aid < DWC_GetConnectNumLanMatch(); aid++ ) {

          if ( aid != DWC_GetMyAIDLanMatch() ) {

                       DWC_SendLanMatch( aid, buf, sizeof( buf ), TRUE );
          }

}
```

Once the data has been sent and received, the callback functions are called.

Code 13-5 Callback Function Called when Sending and Receiving Data

```
/**
 * Callback called when data is received
 */
void recvCallback( s32 aid, u8* buf, s32 len )
{
        printf( "recvCallback from %d ( len = %d )\n", aid, len );
}

/**
 * Callback called when (after) data is sent
 */
void sendCallback( s32 aid, s32 len )
{
        printf( "sendCallback to %d ( len = %d )\n", aid, len );
}
```

13.1.5 Terminating LAN Matchmaking

To terminate LAN matchmaking, call the `DWC_ShutdownLanMatch` function. Doing so immediately ends communication and deallocates the used memory. To use LAN matchmaking again, perform the same processes for LAN matchmaking, starting with initialization.

13.2 LAN Matchmaking and DWC Errors

Because LAN matchmaking is designed for testing purposes, it is independent of the DWC module error processing. Thus, any errors that occur during LAN matchmaking do not affect any DWC errors.

13.3 LAN Matchmaking and Other Identical DWC Functions

Because LAN matchmaking is a feature provided for testing purposes, it is assumed not to be included in released products, nor is it supported when combined with other features.

14 Communication Errors

In the DWC, there is a single system for simultaneous handling of errors for each DWC module. Therefore, DWC errors can be handled in the same way as application errors.

14.1 Error Processing

You can get the status of DWC errors with the `DWC_GetLastErrorEx` function (see Code 14-1). The return value is the error type, and the arguments are the error code and a pointer to the location at which the error process type is stored.

An error code can be a zero or a negative number. When displaying them, invert the sign and display them as positive numbers. However, an error display is unnecessary when the error is recoverable and the Nintendo Wi-Fi Connection is not broken.

The error process type indicates the necessary restore process after an error occurs. Fixed error processes can be created for each value.

Once an error state is entered, DWC no longer accepts most functions. To recover from an error state, call the `DWC_ClearError` function (see Code 14-1).

Code 14-1 Error Processing

```
void main_loop( void )
{
    while ( 1 )
    {
        DWC_ProcessFriendsMatch();

        handle_error();  // error processing

        GameWaitVBlankIntr();
    }
    :
}

// error processing
void handle_error( void )
{
    int dwcError, gameError;

    dwcError  = handle_dwc_error();
    gameError = handle_game_error();
    :
}
```

```
int handle_dwc_error( void )
{
   int errcode;
   DWCError err;
   DWCErrorType errtype;

   // Get the error
   err = DWC_GetLastErrorEx( &errcode, &errtype );

   // If there is no error, do nothing and return
   if ( err == DWC_ERROR_NONE ) return 0;

   // Clear the error
   DWC_ClearError();

   // Display an error message
   disp_error_message( err );
   // If the error code is less than 0, display as a positive number
   if ( errcode <= 0 ) disp_message( "%d", -1*errcode );

   if ( errtype == DWC_ETYPE_SHUTDOWN_FM )
   {
      // End the FriendsMatch process
      DWC_ShutdownFriendsMatch();
   }
   else if ( errtype == DWC_ETYPE_DISCONNECT )
   {
      // End FriendsMatch process and perform cleanup for Internet connection
      DWC_ShutdownFriendsMatch();
      disconnect_func();
   }
   else if ( errtype == DWC_ETYPE_FATAL )
   {
      // Since this is a fatal error, disable after recommending a power cycle
      while (1) ;
   }
   // If a minor error, just clear it and you can restart the FriendsMatch process

   return err;
}
```

14.2 Error Code List

The following is a list of the major error codes returned during matchmaking and friend relationships.

Note: Errors with the last three digits of `010` or `020` occur most frequently when the GameSpy servers are unstable, such as when undergoing maintenance.

- 61010: A communication error with the GP server occurred while logging into the GameSpy GP server.
- 61020: A communication error with the GP server occurred while logging into the GameSpy GP server.
- 61070: A login timeout error occurred while logging into the GameSpy SP server.
- 71010: A communication error with the GameSpy GP server occurred while synchronizing the Wii friend roster.
- 80430: A connection failure occurred, possibly due to a loss of power to the connection server host or one of the connecting client hosts during server-client matchmaking for a client host.
- 81010: A communication error with the GameSpy GP server occurred during matchmaking.
- 81020: A communication error with the GameSpy GP server occurred during matchmaking.
- 84020: Communication was stopped from the GameSpy master server during matchmaking. Either the master server is down, or the firewall is blocking UDP.
- 85020: A communication error with the GameSpy master server occurred during matchmaking.
- 85030: The GameSpy master server had a DNS failure during matchmaking. Errors whose last three digits are 030 are all DNS errors.
- 86420: Several NAT negotiations failed during a single matchmaking. There is a possible problem with the router. This only occurs with client hosts that have begun connections for server-client matchmaking, and this error results after one NAT negotiation failure.
- 97003: A socket error occurred at a layer below the DWC after matchmaking completed.