Nintendo Wi-Fi Connection
Revolution DWC Programming Manual

Version 2.0.0

RVL-06-0199-001-Q
Released: August 8, 2008

Table of Contents

Code

Tables

Figures

Revision History

Version	Revision Date	Description
2.0.0	2008/06/05	• Revised to support DWC 2.0
1.3.10	2008/04/17	• Added a note to section 10.3.9.3 Delete Entry Dialog Box, stating that data cannot be deleted with “all” specified on the production server.
1.3.9	2008/01/31	• Added information to section 8.5 Data Send and Receive Volumes (when Wireless Communications are Used) regarding the maximum number of communication packets per second. • Slightly revised the notation in section 10.2.4.3 Getting Communication Results: Ranking List Get Mode (Top, Nearby, Friend) to make it easier to understand. • Removed a description from Chapter 4 Creating User Data regarding the use of the `DWC_CheckHasProfile` function at connection time (this description was DS-specific and unrelated to Wii).
1.3.8	2008/01/30	• Added information on how to connect to the general-purpose ranking server for development from the production general-purpose ranking server. • Added an explanation to section 11.4.1 Initialization for a Callback that Was Added to the Download Library.
1.3.7	2008/01/07	• Added support for the revision that implemented ranking timeouts in the library.
1.3.6	2007/12/13	• Deleted the description in section 6.4 Getting Friend Status that stated that several hundred microseconds are required. • Added a caution about cancellation in Chapter 9 Network Storage Support. • Deleted descriptions that pertained to GHTTP, which is now prohibited.
1.3.5	2007/10/03	• Standardized notation and corrected typos. • Added section 3.2.1 DWC Memory Usage Estimates.
1.3.4	2007/08/30	Corrected an error in which the maximum number of ranking categories was listed as 300.
1.3.3	2007/07/24	Changed the maximum number of ranking categories from 100 to 1000 and the number of categories that can be obtained at once from 10 to 30.
1.3.2	2007/06/29	• Added section 3.1 Items to Be Considered when Using 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 section 5.1.

Version	Revision Date	Description
1.3.0	2007/04/27	• Added a description about the console friend link feature to sections 2.2.1 and 2.2.3. • Added support for argument changes for `DWC_InitFriendsMatch` and `DWC_InitLanMatch`. • Added to Table 11.3 Parameters Set with the Order Get Mode. • Added to section 11.2.4.3 Getting Communication Results: Ranking List Get Mode (Top, Nearby, Friend). • Changed the maximum size of the user-defined data in section 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 section 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 section 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	Release 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 section 8.3 Targets for the Buffer Size Specified by `DWC_InitFriendsMatch`.
0.0.2	2006/10/30	Revised 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 connect Wii 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 omits the functions associated with GameSpy servers and allows the use of only the download function.

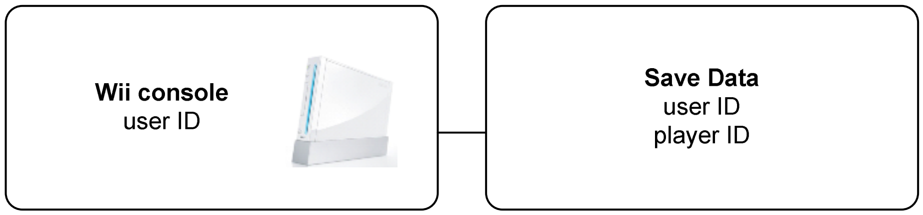
This document is shared by the RVL DWC and RVL DWC-DL packages and may include information not applicable to developers who use the RVL DWC-DL package.

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 in 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 for a Wii Nintendo Wi-Fi Connection compatible title to another Wii console. It is also not possible to recreate or delete user IDs saved in the Wii console. Accordingly, no discrepancies arise between the user ID saved in 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.

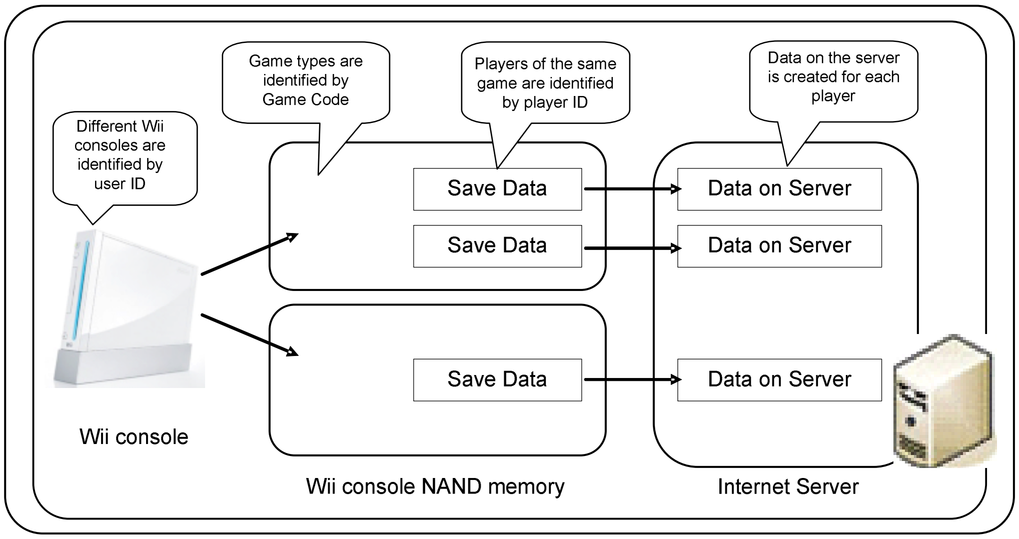
The player ID is a 32-bit random number. Because the data on the Internet server is managed in units of “user ID + player ID + Game Code,” no problems will arise as long as the player ID is unique for the same user ID and the same Game Code. If it is duplicated, a player ID that is not a duplicate will 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 players 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).

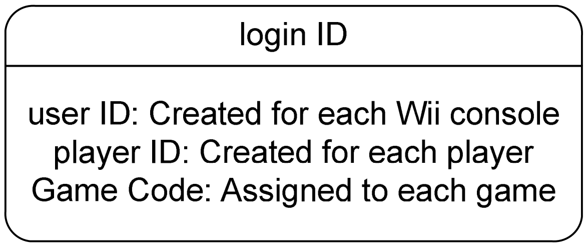
Figure 2-2 How Data Is Maintained on the Internet


Different Wii consoles are identified by user ID
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). User information saved on the Internet server is called a *GS profile*, and the ID used to manage GS profiles on the server is called the *GS 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 GS profile ID is used to search for other user GS profiles on the Internet server from within DWC. GS 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.

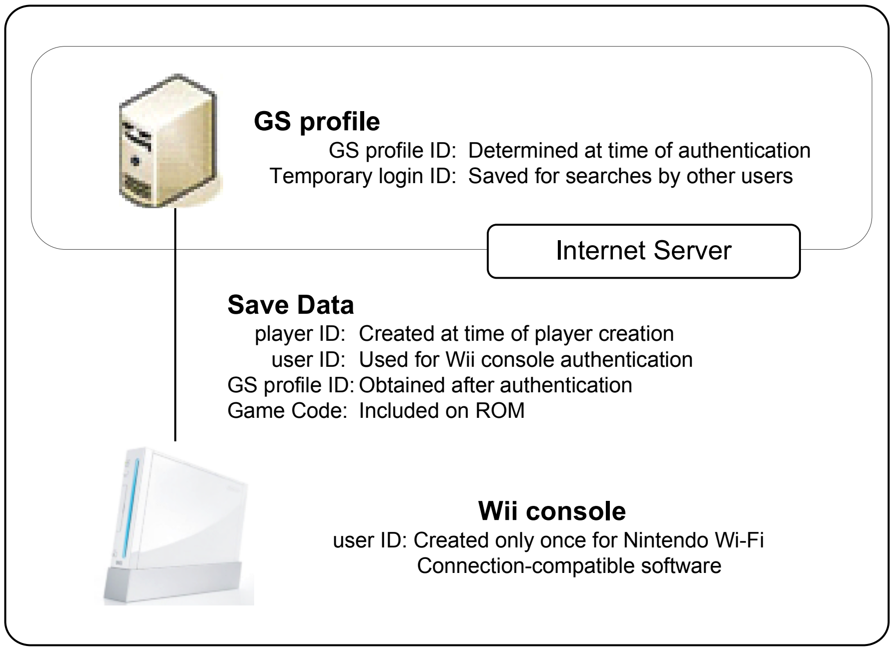
Note: The size of the information used for authentication is 64 bytes.

A temporary login ID, post-authentication login ID, and GS 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 its contents.

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.

The terminology related to Nintendo Wi-Fi Connection authentication used up to this point is shown in Figure 2-4.

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


GS profile
GS 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
GS 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

With DWC, friendships are formed on the Internet server so that communications with friends can be started easily. Friendships are formed by exchanging user information. Established friendships are saved in each user's GS profile.

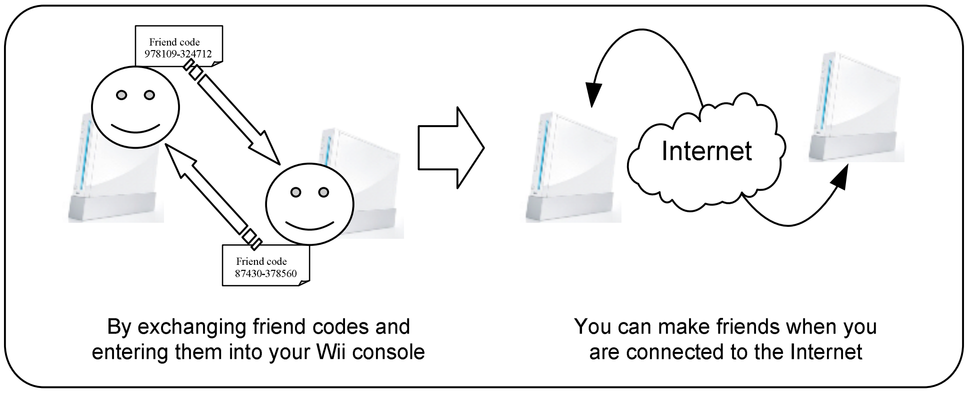
The information that can identify a partner user for entering into a friend relationship is called a "friend code." The exchange of these friend codes provides a mechanism for creating friend relationships (see Figure 2-5):

- 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 from the Nintendo Wi-Fi Connection authentication information 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 GS profile ID obtained when logging onto 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 friend codes and entering them into 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. In addition, the actual game must save the managed information (nicknames and versus scores) related to friends. DWC handles this as friend information without any awareness of the login ID, GS profile ID, or type of friend code.

Note: Twelve bytes are required per person 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, developers should have a good grasp of both types of friendships 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.

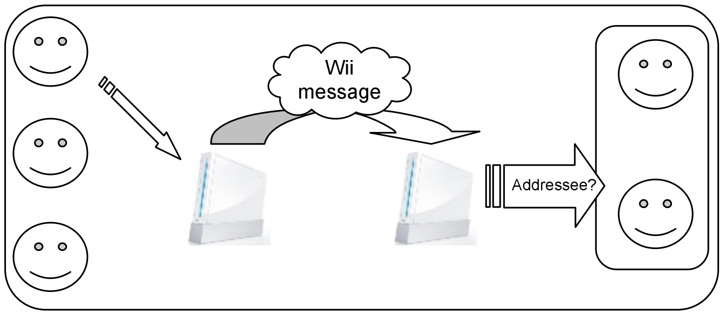
  Some method must be provided, such as including “Hi [friend name], it’s [user name]. Would you like to play a Nintendo Wi-Fi Connection game with me?” in the message, so the user can determine the Wii user to whom the friend request is being made.

  Recipients cannot be specified even for Wii messages that cannot be posted on the Wii Message Board and can be seen only from the game program.

- 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 are not applied right away. Thus, the friend relationship is not established immediately, like it is when 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 issues 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 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. Therefore, it is 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 (the general-purpose ranking and data storage servers are also linked)
- Initialization of 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',           // game 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 region (NULL pointer)
     if( ptr != NULL )
     {
          OSLockMutex( &s_mutex );
          MEMFreeToExpHeap( s_hmem2heap, ptr);
          OSUnlockMutex( &s_mutex );
     }
}
```

3.2.1 DWC Memory Usage Estimates

Table 3-1 shows the amount of memory used by DWC.

The following figures are the peak values before library usage ends. The actual amount of memory that is necessary depends on the allocator design, so a size larger than the one in the table must be specified.

Table 3-1 DWC Memory Usage Estimates (Measured Values)

	Two-person matchmaking	Three-person matchmaking	Four-person matchmaking
Anybody	120,877 bytes	139,563 bytes	158,267 bytes
Friends	115,593 bytes	139,209 bytes	169,347 bytes
Server-Client (Server)	115,593 bytes	150,591 bytes	169,444 bytes
Server-Client (Client)	120,498 bytes	139,209 bytes	157,977 bytes

3.2.2 Using a DNS Server to Connect to a Different Server

The connection target for the authentication server specified by the `DWC_Init` function also affects the connection target for the general-purpose ranking and data storage server.

When creating a version of a released title in another language (or in other situations), 125.206.241.190 can be specified as the DNS server to forward communications to the development server for programs that connect to the production server. This is only effective for general-purpose ranking and data storage features, however, and does not affect authentication, downloads, and matchmaking.

3.3 Shutdown

To shut down the use of DWC, call the `DWC_Shutdown` function. Calling `DWC_Shutdown` deallocates the memory used for the DNS cache.

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

To use DWC again, it must be called from `DWC_Init`.

Code 3-2 Shutting Down DWC

```
void shutdown_dwc( void )
{
     // DWC Initialization
     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 in the save 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;
   }

   // When valid user data is not saved
   printf("no Backup UserData\n");

   // Create user data
   DWC_CreateUserData( &s_PlayerInfo.userData );

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

   return FALSE;
}
```

The `DWC_CheckDirtyFlag` function can be used to determine whether it is necessary to save the user data to save 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) );
    }
}
```

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. When connecting to the server after this, the user ID and player ID are saved to the user data created by DWC, and a GS profile is created. The GS profile ID that corresponds to the created GS profile is also stored in the user data.

5.1 Connecting to the Nintendo Wi-Fi Connection Server

When connecting to the Nintendo Wi-Fi Connection 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
- GS product ID provided by GameSpy
- Game name provided by GameSpy
- Secret key provided by GameSpy
- Size of the send/receive buffer for peer-to-peer communications
- 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, the default of 8 KB is used.

The friend roster is an array of friend information in the `DWCFriendData` structure. For details about the friend roster and friend information, see Chapter 6 Creating Friend Rosters and Friend Information.

Next, 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 in 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 appropriateness. The results of that check can be obtained 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_LoginAsync` function, call the `DWC_ProcessFriendsMatch` function in 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 intentional network processes by the application (such as during connection to another host) be sure to call this function because communication processes (such as updating the friend roster) may be issued.

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 game screen name 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 two procedures are available in DWC to create friend relationships among players.

- Using friend codes

  Friend codes, which are GS profile IDs that include error-checking information, are exchanged.

  To use a GS profile ID, the user must connect 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 information.

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

The validity of the input friend code 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 to re-enter
       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 friend roster maintained by the application (hereafter, the local friend roster) online, call the `DWC_UpdateServersAsync` function to update the friend roster stored on the GameSpy server (hereafter, the server 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 friend roster synchronization is complete) with its parameters
- Callback for the friend status change notification (described below) with its parameters
- Callback for the friend roster deletion and its parameters

Synchronization of the friend rosters works primarily by:

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

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

The friend roster synchronization completion callback is called when both the local and the server 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 all friend relationships have been established.

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

In addition, if duplicate friend information for the same Wii Friend is found during friend roster synchronization, duplicate information is automatically deleted, leaving a single entry. Whenever a deletion occurs, the index of the friend information in the friend roster that was deleted and the index of the Wii 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 );

     // Synchronize the local friend roster and server 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 friend roster is updated irregularly, perform the following processes
      // when appropriate, and save the updated local friend roster all at once.
      if ( s_updateFriendList )
```

```
        {
                // Save if the friend roster has been updated
                s_updateFriendList = FALSE;
                save_friendList();
        }

        game_loop();

        GameWaitVBlankIntr();
    }
    :
}

// Friend roster synchronization completion callback
void cb_updateServers( DWCError error, BOOL isChanged, void* param )
{
    if ( error == DWC_ERROR_NONE )
    {
            // Wii friend roster synchronization successful
            s_update = TRUE;

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

// Friend roster 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 can get the data type set in 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 for the state when no connection to Nintendo Wi-Fi Connection has been made
• `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 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 data is the GS profile ID registered with the friend code (indicating that the friend relationship has not yet 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 information 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 maintain 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 with 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 that has a character count of roughly 1.5 times the data size.)

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

Once a friend relationship is established, a friend's current status can be obtained. 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 the user 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.

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 friend rosters
     if ( !DWC_UpdateServersAsync( NULL,
                                    cb_updateServers, NULL,
                                    cb_friendStatus, NULL,
                                    NULL, NULL ) )
    {
            // Synchronization failed to start
            return;
    }
    :
```

```
    // Friend roster 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;

                  // If the friend information is valid, get that friend’s status
                  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

An exchange of friend codes in an application can be performed by embedding them in WiiConnect24 messages. This enables the exchange of friend codes between users who have formed a Wii console friend relationship. For details about the methods of sending and receiving messages, see the *WiiConnect24 Overview*.

To embed one's friend code in a WiiConnect24 message, use the `DWC_CfSetAppFriendKeyToNWC24Msg` function (see Code 6-5). Calling this function will dynamically allocate the memory internally, but when the `DWC_Shutdown` function is called, this memory will be deallocated automatically. Call the `DWC_CfReset` function to explicitly deallocate the memory. However, the memory may only be deallocated after the `NWC24CommitMsg` function is called for the `NWC24MsgObj` that contains 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 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 server-client matchmaking.

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

- Peer matchmaking with friend unspecified (with the general public)
- Peer matchmaking with friend specified (with person registered in friend roster)

7.1 Peer Matchmaking with Friend Unspecified

Matchmaking is performed against an unspecified number of players.

To begin peer matchmaking with friend unspecified, use the `DWC_ConnectToAnybodyAsync` function (see Code 7-1).

The following items are passed as the arguments of this function.

- The connection topology (for details, see section 8.2 Connection Topology)
- The maximum number of desired player connections, including the local host
- A filter string that appends conditions for matchmaking
- The matchmaking completion callback and its parameters
- The new connection client notification callback and its parameters
- The player evaluation callback and its parameters (described later)
- The matchmaking condition determination callback and its parameters
- The matchmaking condition values

The matchmaking completion callback is called not only when the local host has successfully connected to the server host, but also when a new client host has been added to the local host's group.

The new connection client notification callback is called when a new client host has begun the process of connecting to the group to which the local host belongs.

The filter string is used to filter players to be considered as matchmaking candidates. To use the matchmaking index keys (the keys named `str_key` and `int_key` in the example below) in this filter string, they must be registered in advance using the `DWC_AddMatchKey*` function (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 section 7.10 Key Names That Cannot Be Used for Matchmaking Index Keys.

Code 7-1 Peer Matchmaking with Friend Unspecified

```
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 index keys
    DWC_AddMatchKeyString( 0, "str_key", s_str_key );
    DWC_AddMatchKeyInt( 0, "int_key", &s_int_key );

    // Begin matchmaking with friend unspecified
    DWC_ConnectToAnybodyAsync( DWC_TOPOLOGY_TYPE_FULLMESH, 4,
                               "str_key = 'anymatch_test' and int_key = 10",
                               cb_sc_match, NULL,
                               cb_sc_new, NULL,
                               NULL, NULL,
                               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_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 the local host cancelled matchmaking or if the local host
         // is the client host and the server host cancelled matchmaking
         s_canceld = TRUE;
      }
      // Do nothing even if newly connected client cancels matchmaking
   }
}
// New connection client notification callback
void cb_sc_new( int index, void* param )
{
   printf( "Newcomer : friend[%d].\n", index );
}
```

7.2 Peer Matchmaking with Friend Specified

Matchmaking is performed with friends registered in the friend roster.

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

The following items are passed as the arguments of this function.

- The connection topology (for details, see section 8.2 Connection Topology)
- The friend roster index array (or the index list) of friends you want to matchmake with
- The number of elements in this index list
- The maximum number of desired player connections, including the local host
- The matchmaking completion callback and its parameters
- The new client connection notification callback and its parameters
- The player evaluation callback and its parameters (described later)
- The matchmaking condition determination callback and its parameters
- The matchmaking condition values

The matchmaking completion callback is called not only when the local host has successfully connected to the server host, but also when a new client host has been added to the local host's group.

The new client connection notification callback is called when a new client host has begun the process of connecting to the group to which the local host belongs.

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

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

Code 7-2 Peer Matchmaking with Friend Specified

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

void do_friend_match( void )
{
     // Begin matchmaking with friend specification
     DWC_ConnectToFriendsAsync( DWC_TOPOLOGY_TYPE_FULLMESH, NULL, 0, 4,
                                cb_sc_match, NULL,
                                cb_sc_new, NULL
                                NULL, NULL
                                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_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 the local host cancelled matchmaking or if the local host
            // is the client host and the server host cancelled matchmaking
            s_canceld = TRUE;
        }
        // Do nothing even if newly connected client cancels matchmaking
    }
}

// Callback to inform of newly connected client
void cb_sc_new( int index, void* param)
{
    printf( "Newcomer : friend[%d].\n", index );
}
```

7.3 Evaluating Matchmaking Candidates

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

When an evaluation callback is specified as an argument for the peer matchmaking start function, the specified evaluation callback is called each time a matchmaking candidate is found. In this callback, the matchmaking index 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 an evaluation value as the return value.

A player with an evaluation value of zero or less is removed as a matchmaking candidate. This does not mean that the player with the highest evaluation value is *always* selected, but the player with the higher evaluation value is more *likely* to 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 index keys
     DWC_AddMatchKeyString( 0, "str_key", s_str_key );
     DWC_AddMatchKeyInt( 0, "int_key", &s_int_key );

     // Begin matchmaking with friend unspecified
     DWC_ConnectToAnybodyAsync( DWC_TOPOLOGY_TYPE_FULLMESH, 4,
                                "str_key = 'anymatch_test'",
                                cb_sc_match, NULL,
                                cb_sc_new, NULL
                                cb_eval, NULL
                                NULL, NULL,NULL);
     :
}

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

     // Get the value for the matchmaking index 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;
     }
}
```

Even though matches are not made with peers that have a different connection topology (for details, see section 8.2 Connection Topology), the evaluation callback gets called anyway. In such cases, the value returned by the evaluation callback has no effect on the result of matchmaking.

7.4 Server-Client Matchmaking

Server-client matchmaking is performed among friends by clearly classifying each host as either a server or a client.

The server host specifies the following.

- The connection topology (for details, see section 8.2 Connection Topology)
- The maximum number of desired player connections, including the local host
- The matchmaking completion callback and its parameters
- The new connection client notification callback and its parameters
- The matchmaking condition determination callback and its parameters (described later)
- The matchmaking condition values

With these items specified, the server host calls the `DWC_SetupGameServer` function and then waits for the connection from the client host (see Code 7-4).

The client host specifies the following.

- The connection topology (for details, see section 8.2 Connection Topology)
- The friend roster index of friends you want to connect with
- The matchmaking completion callback and its parameters
- The new connection client notification callback and its parameters
- The matchmaking condition determination callback and its parameters (described later)
- The matchmaking condition values

If a client host calls the `DWC_ConnectToGameServerAsync` function and specifies these items, it can proceed to connect once one of its friends starts the matchmaking process as a server 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 the user's connection to the server host is successful, but also when another client host is added to the group to which the user belongs.

The new connection client notification callback is called when a client host starts a new connection to the group to which the user already belongs.

Code 7-4 Server-Client Matchmaking

```
static BOOL s_matched = FALSE;

void do_server_match( void )
{
     // Begin matchmaking as the server host
     DWC_SetupGameServer( DWC_TOPOLOGY_TYPE_FULLMESH, 4,
                          cb_sc_match, (void *)CB_CONNECT_SERVER,
                          cb_sc_new, NULL
                          NULL, NULL, 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( DWC_TOPOLOGY_TYPE_FULLMESH, 0,
                                   cb_sc_match, (void *)CB_CONNECT_CLIENT,
                                   cb_sc_new, NULL,
                                   NULL, NULL, 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 the local host cancels matchmaking or if the local host
                   // is a client host and the server host cancelled 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 );
}
```

7.5 Reconnecting Using Group ID

A feature for both peer matchmaking and server-client matchmaking enables you to rejoin a group in which you were a participant after having left the group, for example because you were unintentionally disconnected.

Once the local host has successfully called the matchmaking completion callback, the local host can use the `DWC_GetGroupID` function to get the group ID that identifies the group in which it was a participant (see.Code 7-5).

Code 7-5 Getting the Group ID

```
static u32  s_groupID = 0 // Group ID for group in which the local host most recently participated

// 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;

            // At this time, get group ID
            s_groupID = DWC_GetGroupID();
        }
        else if ( self || isServer )
        {
            // If the local host cancelled matchmaking or if the local host
            // is a client host and the server host cancelled matchmaking
            s_canceld = TRUE;
        }
        // Do nothing even if a newly connected client cancels matchmaking
    }
}
```

After the local host gets the group ID with `DWC_GetGroupID`, it can reconnect to a group in which it once was a participant using the `DWC_ConnectToGameServerByGroupID` function.

Call the `DWC_ConnectToGameServerByGroupID` function with the following values specified for its arguments (see Code 7-6**Error! Reference source not found.**).

- The connection topology (for details, see section 8.2 Connection Topology)
- The group ID
- The matchmaking completion callback and its parameters
- The new connection client notification callback and its parameters
- The matchmaking condition determination callback and its parameters (described later)
- The matchmaking condition values

The matchmaking completion callback is called not only when the local host has successfully connected to the server host, but also when a new client host has been added to the local host's group.

The new connection client notification callback is called when a new client host has begun the process of connecting to the group to which the local host belongs.

Code 7-6 Reconnecting Through the Group ID

```
static u32  s_groupID = 0 // Group ID for group in which the local host most recently participated
static BOOL s_matched = FALSE;

void do_groupID_match( void )
{
    // Start matchmaking as part of reconnection via group ID
    DWC_ConnectToGameServerByGroupID(DWC_TOPOLOGY_TYPE_FULLMESH,
                         s_groupID,
                         cb_sc_match, NULL,
                         cb_sc_new, NULL,
                         NULL, NULL, NULL );

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

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

        GameWaitVBlankIntr();
    }

    // Matchmaking complete
    :
}


// 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 the local host cancelled matchmaking or if the local host
            // is the client host and the server host cancelled matchmaking
            s_canceld = TRUE;
        }
        // Do nothing even if a newly connected client cancels matchmaking
    }
}

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

Reconnecting through a group ID will fail if there is no longer anyone in the specified group or if the specified group is filled to capacity.

7.6 Suspension

When matchmaking has attained the predefined number of people, no other new clients can participate. However, it is also possible to suspend the process so that no other new clients can participate even if the predefined number of people has not been reached. Further, the suspended process can be reopened to accept participants again.

To suspend or recommence matchmaking, all participating hosts must call `DWC_RequestSuspendMatchAsync` at the same time, specifying the suspension state to set, the suspension state change callback, and the callback parameters (see Code 7-7).

The suspension process will complete successfully only if all participants call `DWC_RequestSuspendMatchAsync` at the same time, specifying the same suspension state.

For example, when transitioning from the game settings screen to the real game screen, all participants must always perform the suspension process by using the same suspension state argument to call `DWC_RequestSuspendMatchAsync`.

To obtain the current setting for the suspension state, use the `DWC_GetSuspendMatch` function.

Code 7-7 The Suspend Process

```
static BOOL s_suspendCompleted = FALSE;

void do_suspend(BOOL suspend)
{
    DWC_RequestSuspendMatchAsync(suspend, suspendCallback, NULL);

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

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

        GameWaitVBlankIntr();
    }

    // Suspension state change has completed
}


// Callback to call on completion of the suspend process
static void suspendCallback(DWCSuspendResult result, BOOL suspend, void* data)
{
    if(result == DWC_SUSPEND_SUCCESS)
    {
        DWCDemoPrintf("Suspension state has changed to [%s].\n", suspend ? "TRUE" : "FALSE");
    }
    else
    {
        // Error
    }

    s_suspendCompleted = TRUE;
}
```

7.7 ConnectAttemptCallback

For the following functions, the application can set any value for `u8 *connectionUserData`. The size of this parameter is `DWC_CONNECTION_USERDATA_LEN`, currently 4 (see Code 7-8). These functions all commence matchmaking.

- `DWC_ConnectToAnybodyAsync`
- `DWC_ConnectToFriendsAsync`
- `DWC_SetupGameServer`
- `DWC_ConnectToGameServerAsync`
- `DWC_ConnectToGameServerByGroupID`

The value set for `u8 *connectionUserData` is sent to the server host, which oversees the acceptance of new client hosts and is used by that server host to determine whether the local host can participate in matchmaking.

For friend-specified and friend-unspecified matchmaking, the application does not need to be aware of who the server host is. It is best to assume that anyone could be the server host. (The same is true even with server-client matchmaking because anyone can become the server host if the initial server host leaves and another takes on the role as the replacement.)

The server host takes calls for `DWCConnectAttemptCallback` in accordance with the `connectionUserData` of new client hosts.

At this point in time, the acceptance of the new client host is not yet decided. The host that receives `DWCConnectAttemptCallback` makes the determination based on the new client host's `connectionUserData`, which was passed to the callback argument. If it decides to accept the new client host, the callback function returns `TRUE`.

If the callback ends on `TRUE`, the newly connected client host is accepted and it begins connecting to the other client hosts already connected. During connection, the `connectionUserData` of the newly connected client host is reported to and shared with all the other client hosts.

Use the `DWC_GetConnectionUserData` function to get the `connectionUserData` for all client hosts already connected (that is, all hosts with an AID, including the local host). This function will be used for decision-making materials inside `DWCConnectAttemptCallback`.

Code 7-8 Example of ConnectAttemptCallback

```
define MALE   0
#define FEMALE 1
#define MAX_MALE_COUNT   2
#define MAX_FEMALE_COUNT 2


:
{
   u8 userData[DWC_CONNECTION_USERDATA_LEN];
```

```
    :

    // Start friend-unspecified matchmaking
    userData[0] = MALE;
    DWC_ConnectToAnybodyAsync(DWC_TOPOLOGY_TYPE_FULLMESH, 4,
                              NULL,
                              cb_sc_match, NULL,
                              cb_sc_new, NULL,
                              NULL, NULL,
                              ConnectAttemptCallback, NULL,userData );
    :
}

/* This portion demonstrates the case in which you pre-set a value indicating FEMALE
or MALE in [0] of DWC_ConnectToXXX’s connectionUserData, then use that value to set
upper limits on the number of males and females.*/
static BOOL ConnectAttemptCallback(u8* newClientUserData, void* param)
{
  int numAid;
  u8* aidList;
  u8* userData;
  int i;
  int maleCount = 0;   // Number of males
  int femaleCount = 0;  // Number of females

  numAid = DWC_GetAIDList(&aidList);
  for(i = 0; i < numAid; i++)
  {
     DWC_GetConnectionUserData(aidList[i], &userData);
     if(userData [0] == MALE)
       maleCount++;
     else if(userData [0] == FEMALE)
       femaleCount++;
  }

  if(newClientUserData [0] == MALE && maleCount < MAX_MALE_COUNT)
    return TRUE;
  else if(newClientUserData[0] == FEMALE && femaleCount < MAX_FEMALE_COUNT)
    return TRUE;
  else
    return FALSE;
}
```

7.7.1 Difference Between DWCEvalPlayerCallback and DWCConnectAttemptCallback

While the two functions `DWCEvalPlayerCallback` and `DWCConnectAttemptCallback` are both designed to narrow down the targets for matchmaking, they differ in the following ways.

- `DWCEvalPlayerCallback` cannot be used for server-client matchmaking. In contrast, `DWCConnectAttemptCallback` can be used for all types of matchmaking.
- Different hosts judge how to narrow down the matchmaking targets.
  - `DWCEvalPlayerCallback`

    Called by the newly participating client host, which makes the judgments.
  - `DWCConnectAttemptCallback`

    Called by the server host that accepts the newly joined client host. This server host makes the judgments.
- Different information is used to narrow down the targets.
  - `DWCEvalPlayerCallback`

    The callback is called from the new client host. When this happens, use the argument passed to the callback to call the `DWC_GetMatchIntValue` or `DWC_GetMatchStringValue` function and get the connection target server host's information. You cannot get information from the other hosts already connected to that connection target server.
  - `DWCConnectAttemptCallback`

    Called from the server host that accepts the newly participating client host.

    When this happens, to get the `connectionUserData` passed to the `DWC_SetupGameServer/DWC_ConnectToXxx` functions by the newly participating client host and all other hosts that are already participating, use `DWC_GetConnectionUserData` and the arguments for `DWCConnectAttemptCallback`.
- Different timing is used when narrowing down the matchmaking targets.
  - `DWCEvalPlayerCallback`

    Called by a new client host when it is selecting the server host it will subsequently participate with. It will not proceed to connect to peers that are excluded by the narrowing-down criteria.
  - `DWCConnectAttemptCallback`

    Called when a new client host tries to connect to the server host.

    Client hosts that are excluded by the narrowing-down criteria cannot participate in matchmaking with the server host that performed the narrowing down.

To summarize these differences:

- `DWCEvalPlayerCallback`
  - Used when it is possible to judge whether to allow participation and base that judgment solely on the connection target server host's information

- Cannot be used in server-client matchmaking, only in the other types.

- `DWCConnectAttemptCallback`
  - Used when judging whether to allow participation must be done dynamically, based on information about all participating hosts.
  - Can be used with all types of matchmaking.

7.8 Server Exchange

If the server host in a connected group becomes disconnected from that group, one of the remaining hosts can inherit the role of server host. In this way, other hosts will still be able to join the group even if the person who initially acted as the server host leaves. For friend-specified matchmaking and server-client matchmaking, this feature can cause situations where the replacement server host has "friend of a friend of a friend" or even more distant relationships with the other client hosts.

To ensure that all members fall within the scope of "friends of friends," the following processes are conducted in friend-specified matchmaking and server-client matchmaking.

1. With the first server host, newly connected client hosts can join if they are friends of the server host.
2. After the server host has been replaced even once, no new client hosts can join the group.

For server hosts in this second state, the value returned by `DWC_GetFriendStatus`-type functions is `DWC_STATUS_PLAYING`.

7.9 Increasing Matchmaking Speed

If using peer matchmaking with friend unspecified, using filtering can increase matchmaking speed when a matchmaking candidate list is retrieved from the matchmaking server (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 re-obtain 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 and time is lost.

When trying to increase the matchmaking speed, consider the following.

- Using filtering to make the candidate list a list of candidates with whom matchmaking is likely to succeed
- Creating specifications that optimize candidates without using extreme filtering in the evaluation callback

7.10 Key Names That Cannot Be Used for Matchmaking Index Keys

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

Table 7-1 List of Key Names That Cannot Be Used as Matchmaking Index 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 completes, the participating hosts will all have established connections with each other. However, which hosts have direct connections will differ depending on the connection topology configured at the start of matchmaking.

There are three kinds of topology. In the first topology interconnections are made between all hosts every time a player joins through matchmaking. In the second topology a mutual interconnection is formed only between server and client. The third topology is an intermediate type, in which at first the interconnection is formed only between server and client, but later all hosts become interconnected in the background.

Some preparation is needed before communication among hosts on this network can begin.

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

Next, the send/receive callbacks can be set using the `DWC_SetUserSendCallback` and `DWC_SetUserRecvCallback` functions (see Code 8-1). The receive callback is called when the 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, can be set to be called when either the user’s device or that of another host officially 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, so 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;
```

[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.

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

           // 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, NULL);

     // Set the receive callback
     DWC_SetUserRecvCallback( cb_recv, NULL);

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

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

// Data receive callback
void cb_recv( u8 aid, u8* buffer, int size, void* param )
{
     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. Because confirmation is made each time a sent packet arrives instead of being able to rearrange the order in which packets arrive, it takes time for transmission to complete.

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 retransmission.

When data being sent 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 Reliable transmission is attempted 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 set (1,465 bytes by default). If there is an attempt to send more than this amount, the data is partitioned and transmission is suspended. 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 Reliable transmission is possible, including checking whether there is space in the send buffer and whether the recipient AID is valid (see Code 8-2).

If using Unreliable transmission, an attempt to send more than the data size noted above fails and `FALSE` is returned.

Code 8-2 Sending Data

```
static u8  s_SendBuffer[ SIZE_SEND_BUFFER ];

void send_data( void )
{
     // 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 Reliable transmission to the host whose AID = 0 can be done
     if ( !DWC_IsSendableReliable( 0 ) ) return;
```

```
    // Reliable transmission of data to a specific host.
    DWC_SendReliableBitmap( 0, s_SendBuffer, SIZE_SEND_BUFFER );
    :
}
```

8.2 Connection Topology

There are three possible connection topologies that provide a route for Reliable or Unreliable communications after the matchmaking for a new participant has completed: *hybrid*, *star* and *full mesh*.

With Unreliable communications, all hosts can mutually intercommunicate. Reliable communications have restrictions that depend on the connection topology and concern which peers a host can send signals to.

You can set the connection topology as an argument of these functions, all of which commence matchmaking.

- `DWC_ConnectToAnybodyAsync`
- `DWC_ConnectToFriendsAsync`
- `DWC_SetupGameServer`
- `DWC_ConnectToGameServerAsync`
- `DWC_ConnectToGameServerByGroupID`

When you start matchmaking, configure the desired connection topology using these functions.

1. Full mesh (`DWC_TOPOLOGY_TYPE_FULLMESH`)

   At the time when matchmaking has completed for a newly joined host, mutual interconnections exist among all hosts, and they can communicate with one another using both Reliable and Unreliable communications.

2. Star (`DWC_TOPOLOGY_TYPE_STAR`)

   In this mode, mutual connections exist only between the server host and the client hosts. Reliable communications can take place between server host and client host, but not among the client hosts. Unreliable communications can take place among all hosts. The server host automatically relays Unreliable communications between client hosts.

3. Hybrid (`DWC_TOPOLOGY_TYPE_HYBRID`)

   At the time when matchmaking has completed for a newly joined host, an interconnection exists only between the server host and the newly connected client host. After matchmaking, interconnections among client hosts are built in the background.

   Hosts that have a direct connection can use Reliable communications. The library does not provide direct notification of whether a connection has been established between any two given client hosts. You must use the method described below to check whether Reliable communications can be conducted with a given peer. (Because the server host and the client host invariably are interconnected, Reliable communications are possible in this case.)

All hosts can use Unreliable communications with each other. For Unreliable communications between client hosts, the server host automatically acts as a relay when the client hosts are not interconnected.

To get a list of the AIDs of the hosts that can conduct Reliable communications, use the `DWC_GetDirectConnectedAIDBitmap` function. Moreover, `DWC_IsSendableReliable` always returns `FALSE` for peer hosts that are not directly interconnected.

Matchmaking takes the longest time when the topology is set to full mesh because all hosts must have established interconnections at the time that matchmaking completes. In addition, the more people who have joined a group, the longer the matchmaking takes to complete.

To speed up the matchmaking process, use either the star topology or the hybrid topology because in either mode the matchmaking ends as soon as the connection has been established between the new client host and the server host.

8.3 Terminating a Connection

To disconnect from all hosts in a group, call the `DWC_CloseAllConnectionsHard` function. Once the close process has been executed, the connection-closed callback set in the `DWC_SetConnectionClosedCallback` function is called before exiting. At the same time, a close notification is sent to connected hosts and the connection-closed callback is called.

This function can even be called 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 terminated even if this function is called.

Also provided are the `DWC_CloseConnectionHard` function that terminates the connection by specifying the AID, and the `DWC_CloseConnectionHardBitmap` simultaneously closes multiple connections by specifying the AID bitmap.

These functions are expected to be used in abnormal circumstances, such as closing a connection to a host with which it is no longer possible to communicate due to a loss of power.

8.4 Targets for the Buffer Size Specified by DWC_InitFriendsMatch

The buffer size specified by the `DWC_InitFriendsMatch` function is the size of the buffer used internally by DWC. The send buffer is used to store the data sent by Reliable communication for which an acknowledgement (ACK) has not been returned. The receive buffer is used to store data that did not arrive in the correct order.

For Reliable communications, the largest send and receive buffers permitted by the game specifications are needed 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 communication 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 number of partitions into which the transmission data is divided 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 Reliable transmission of 100 bytes of data is calculated as follows:

1 (second) x (60 (frames) ÷ 3) x (7 x 2 (partitions) + 100 (bytes) + 15) = 2580 (bytes)

8.5 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 );
     :
}
```

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

Communication traffic during Reliable and Unreliable transmission is given in Table 8-2.

Table 8-2 Communication Data Breakdown

Transmission Data Item	Transmission Data Size			
Preamble	192 bits (24 Bytes)			
MAC	24 Bytes			
LLC	8 Bytes			
IP	20 Bytes			
UDP	8 Bytes			
DATA	Reliable Communications			Unreliable Communications
	Header Transmission	Data Transmission	Receive Check	Data Transmission
	15 Bytes	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 communication, 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 1,424 bytes per frame were sent to two other players in an environment where three Wii consoles connected by different Internet lines had been matched together, we confirmed that communications without packet loss or dropped frames is possible. (Data was both sent and received at a rate of approximately 170 KB/sec.) The maximum communication speed measured at the socket layer is approximately 480 packets/second for send and receive combined.

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

- Network Environment
  - Internet Transmission Latency and Packet Loss (domestic and international)
    - Transmission latency is fairly uniform domestically, but tends to increase in international communications.
  - Congestion in Wireless Environments
    - With large numbers of packets, congestion occurs more easily in wireless environments than in wired environments.
  - 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.
- Communication delays due to other IOP processing loads
  - When the IO processor is monopolized by disc access or another process, the socket receives 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 connections need to be restricted to ensure the game contents, due to connection quality, refer to Nintendo Wi-Fi Connection Programming Guidelines for Wii for an appropriate course of action.

9 Network Storage Support

DWC can store data in persistent storage on the GameSpy network. To access persistent storage, first complete the login process with the `DWC_LoginAsync` function. Next, use the `DWC_LoginToStorageServerAsync` function to log in to persistent storage (see Code 9-1).

Data that can be saved in persistent storage can take either a Public or a Private attribute. Data saved using the `DWC_SavePublicDataAsync` function takes the Public attribute, and other players can reference this data (see Code 9-1). Conversely, data saved using the `DWC_SavePrivateDataAsync` function takes the Private attribute, and other players cannot reference this data.

When data is loaded from persistent storage, 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 friend roster (see Code 9-1).

When saving and loading is complete, the callback function set with the `DWC_SetStorageServerCallback` function is called (see Code 9-1). 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`.

Specify the keys (separated by the `\\` delimiter) for the values to be obtained when loading data. 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 only 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 fit the above description, `success` is `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 persistent storage (see Code 9-1).

Note: The callback might not be called in some cases, such as when a top-level connection on the router is disconnected during loading. Create a timeout on the application side and then implement a system so the user can cancel the process. Call the `DWC_LogoutFromStorageServer` function for the cancellation.

Code 9-1　Accessing Persistent Storage

```
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 save 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--;
}
```

10 General-Purpose Ranking Library

10.1 Introduction

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

The following capabilities are provided by the general-purpose ranking library.

- Uploading 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-purpose ranking library provides functions to communicate with the general-purpose ranking server, used to implement these capabilities. To communicate with the server, the library uses `DWC_GHTTP` internally. Therefore, the general-purpose ranking library and `DWC_GHTTP` cannot be used at the same time. Because all communication errors occurring during general-purpose ranking are generated by `DWC_GHTTP`, error codes depend on `DWC_GHTTP`. During actual use, be sure to follow the instructions given in *Nintendo Wi-Fi Connection Programming Guidelines for Wii*.

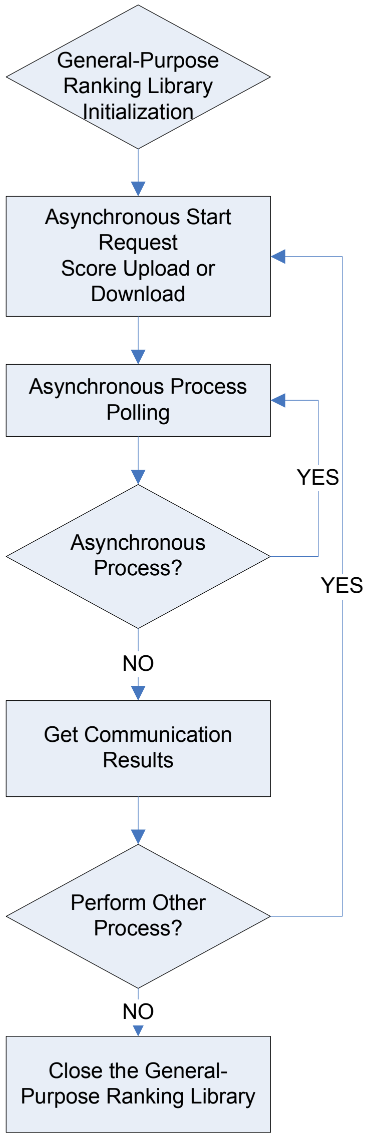
10.2 Using the General-Purpose Ranking Library

This section describes the use of the general-purpose ranking library.

10.2.1 Process Flow

The process flow for the general-purpose ranking library is shown in Figure 10-1.

Figure 10-1 Process Flow


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

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

10.2.2 Initializing the General-Purpose Ranking Library

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

- Initialization Data

  This specifies the general-purpose ranking library initialization data string sent by Nintendo. 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-purpose ranking servers are available: one for development and one for released products. The server that is connected is determined at initialization, based on the authentication server that that is logged into. When `DWC_CONNECTINET_AUTH_RELEASE` is specified in the `DWC_Init` function, the released products server will be connected. When the default state or `DWC_CONNECTINET_AUTH_TEST` is specified, the development server will be connected.

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

Be sure to use the production server for released products.

Code 10-1 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

}
```

10.2.3 Uploading Scores

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

Once the asynchronous process begins, polling continues until the process completes with the `DWC_RnkProcess` function, described in section 10.2.4.6 Using Asynchronous Polling for Getting Communication Result Status.

The category ID, market, score, and user-defined data can be specified in the score to be uploaded.

The category ID is an identifier for game titles that have multiple ranking lists and can be specified in the range of zero to 1000 (`DWC_RNK_CATEGORY_MAX`).

The user can specify `DWC_RNK_REGION_JP` (Japan), `DWC_RNK_REGION_US` (North America), `DWC_RNK_REGION_EU` (Europe), `DWC_RNK_REGION_KR` (Korea), or `DWC_RNK_REGION_ALL` (all countries) for the market. 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.

Any binary up to 764 bytes (`DWC_RNK_DATA_MAX`) in length can be specified in the user-defined data.

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

Code 10-2 Request to Start 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

}
```

10.2.4 Downloading Scores

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

Once the asynchronous process begins, polling continues until the process completes with the `DWC_RnkProcess` function, described in section 10.2.4.6 Using Asynchronous Polling for Getting Communication Result Status.

There are four get modes for downloading scores.

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

Table 10-1 Get Modes for Downloading Scores

Get Modes	Description
Order	Gets the user's own ranking. Gets the order by comparing registered scores. Ascending or descending order is specified with a parameter.
Top Ranking List	Gets the specified number of rankings from the top.
Nearby Ranking List	Gets the specified number of rankings nearest to the user's own.
Friends Ranking List	Gets the ranking for a maximum of 64 friends.

10.2.4.1 Specifying the DWCRnkGetParam Structure

The following is a description of how to specify the `DWCRnkGetParam` structure parameters that are passed as arguments to the `DWC_RnkGetScoreAsync` function. The `DWCRnkGetParam` structure is defined as 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 10-2.

Table 10-2 Size Specified for Each Get Mode's Parameter

Get Mode Parameters	Size Specified
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 fields

  These are specified for the Order Get Mode.

Table 10-3 Parameters Set with Order Get Mode

Parameters	Description
`DWCRnkGetParam.order.sort`	Specifies the sorting order for scores: • `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 the user hasn't uploaded scores within the specified time interval, the ranking within the specified time interval will be obtained for the user's score that was uploaded last.

- DWCRnkGetParam.toplist fields

  These are specified for the Top Ranking List Get Mode.

Table 10-4 Parameters Set with Top Ranking List Mode

Parameters	Description
`DWCRnkGetParam.toplist.sort`	Specifies the sorting order for scores: • `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 30 (`DWC_RNK_GET_MAX`) can be specified.

- DWCRnkGetParam.nearby fields

  These are specified for the Nearby Ranking List Get Mode.

Table 10-5 Parameters Set with Nearby Ranking List Get Mode

Parameters	Description
`DWCRnkGetParam.nearby.sort`	Specifies the sorting order for scores: • `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 30 (`DWC_RNK_GET_MAX`) can be specified. The values start at 2 because the user's score is always listed first.

- DWCRnkGetParam.friends fields

  These are specified for the Friend Ranking List Get Mode.

Table 10-6 Parameters Set with Friend Ranking List Get Mode

Parameters	Description
`DWCRnkGetParam.friends.sort`	Specifies the sorting order for scores: • `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. For example, specifying 180 gets the ranking updated within the last 180 minutes (3 hours).
`DWCRnkGetParam.friends.limit`	Specifies the maximum number of ranking lists to get. A numerical value between 2 and 30 (DWC_RNK_GET_MAX) can be specified. The values start at 2 because the user's 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, it stores from the beginning, and the rest is filled in with zeros.

Code 10-3 Request to Start 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 10-4 Request to Start 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

}
```

10.2.4.2 Getting Communication Results: Order Get Mode

When score downloading in Order Get Mode ends normally, the communication results can be obtained 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 the user's score is not uploaded when the get occurs, zero is returned.

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

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

The `DWCRnkData` structure's order field stores rankings only for the row for the user's own score. 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.) The exception to this is for the Top Ranking list Get Mode, for which zeroes are stored in the order field of all rows (even when the rows contain the user's own score).

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

If the user attempts to get Nearby or Friends Ranking Lists without having registered the user's own score, it is treated as if the user's 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 the user's score, regardless of the interval specified with `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 a number of users have the same score 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 for this, such as limiting the display of scores to the user's alone or increasing the score range to reduce the possibility of duplicate scores or numbering of the scores, even though there is a possibility of error.

10.2.4.4 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 number of scores that match the filtering conditions specified when the `DWC_RnkGetScoreAsync` function is called.

10.2.4.5 Getting Ranking Lists Among Rivals

It is possible to designate the rivals' GS profile IDs in the GS profile ID list specified by the Friends Ranking List Get Mode. By specifying the rival GS profile ID list, the user can get the ranking list among rivals.

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

Code 10-5 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 );
}
```

10.2.4.6 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. In general, call the function once per 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. This return value can be monitored to see when an asynchronous process has ended.

If an error occurs, DWC_RNK_IN_ERROR is returned. Subsequent processing cannot continue once an error occurs, so the general-purpose ranking library must be closed and tried again from the initialization process.

When a timeout (30 seconds) occurs, the DWC_RnkProcess function returns DWC_RNK_PROCESS_TIMEOUT.

Code 10-6 Asynchronous Processing Polling

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

         // V-Blank waiting
         GameWaitVBlankIntr();

}

switch( res ){

case DWC_RNK_PROCESS_NOTASK:     // Asynchronous process completed
      break;

case DWC_RNK_PROCESS_TIMEOUT:    // Timeout (30 seconds)
      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;

}
```

10.2.5 Closing the General-Purpose 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 memory used by the general-purpose ranking library and resets the internal states.

Because the pointer returned by the `DWC_RnkResGetRow` function directly references the receive buffer, it is invalidated when the general-purpose ranking library closes. If necessary, perform a save prior to closing.

10.2.6 Terminating Processes and Error Processing

Any current asynchronous process can be cancelled by calling the `DWC_RnkCancelProcess` function.

When a process is canceled, the general-purpose ranking library enters an error state. Because subsequent processing is not performed once an error occurs, close the general-purpose ranking library, and then retry from the initialization process.

Even when the library is used correctly, the process may be canceled because of a network error occurring due to issues related to connection quality. Therefore, always denote the appropriate error processes.

10.2.7 Error Codes

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

10.2.8 Sample Code (Ranking)

The DWC package includes sample code that uses the general-purpose 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.

10.2.9 Dependency on DWC_GHTTP

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

Note: With the current version, it is not possible for applications to use `DWC_GHTTP` directly.

10.2.10 Memory Consumption

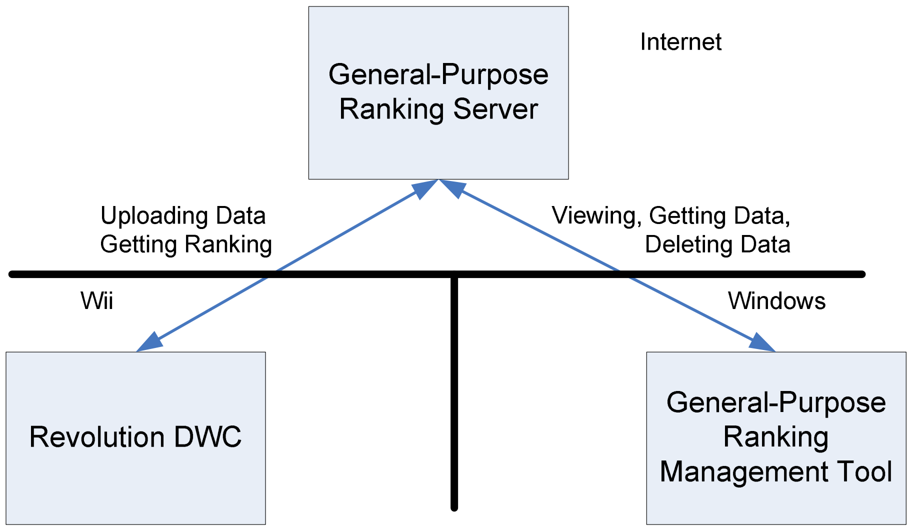
The general-purpose ranking library uses the `DWC_Alloc` function for internally allocating memory. For score uploading, the target amount of memory to be allocated equals 2 x (the size of the user-defined data + approximately 200 bytes). For score downloading, the target is about 200 bytes, in addition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

10.3 General-Purpose Ranking Management Tool

10.3.1 Overview

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

Figure 10-2 Structure Diagram



10.3.2 File Structure

The General-Purpose Ranking Management Tool includes the following files.

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

10.3.3 Run Environment

10.3.3.1 Install the .NET Framework

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

10.3.3.2 Install the Configuration File

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

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

10.3.3.3 Communication Settings

The General-Purpose Ranking Management Tool communicates with the Internet, so run it on a computer that has Internet access. The Internet Explorer settings are used as the communication settings.

10.3.4 Communication Load on the General-Purpose Ranking Server

Because the General-Purpose Ranking Management Tool can get large amounts of data at once, it places a large load on the general-purpose 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.

10.3.5 Time Zones for Acquired Data

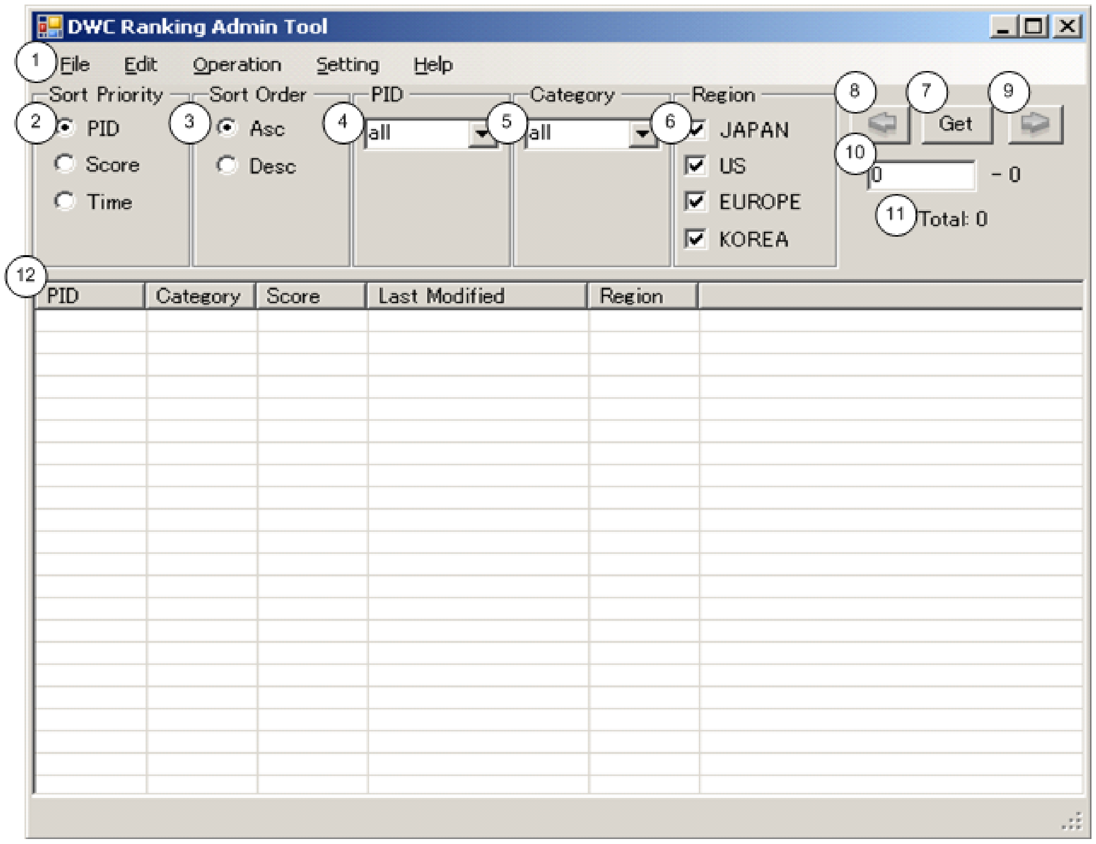
- Time displayed in "Last Modified" in the management tools list:
  - When "use UTC" is checked, displays in UTC.
  - When "use UTC" is not checked, displays in the time zone of the local machine.
- When obtained with **Get CSV** in the management tool:
  - When "use UTC" is checked, sets the time zone 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).

10.3.6 Starting the General-Purpose Ranking Management Tool

Executing `DWCRankingAdmin.exe` starts the General-Purpose Ranking Management Tool.

10.3.7 Screen Composition

Figure 10-3 Main Form



The main form (shown in Figure 10-3) appears when the General-Purpose Ranking Management Tool is started. The main form displays data and specifies the display conditions. The following is a description of the available functionality.

1. Application menus: Menu functions are described in section 10.3.8 Menu Composition.
2. Sorting order 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. Ascending or descending sorting order: The following choices are available.
   - Asc: sort in ascending order.
   - Desc: sort in descending order.
4. PID of the data to be displayed: Enter either a decimal numerical value or “all.” For the latter selection, all PID data is displayed.
5. Category ID for the data to be displayed: Enter either a decimal numerical value or “all.” For the latter selection, all category ID data is displayed.

6. Market (Region) of the data to be displayed: Each of the markets can be toggled on or 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. Display previous button: This button is active when the offset is greater than zero and the data is displayed. Pressing this button displays data with an offset value that is set to 100 less than the current offset.
9. Display next button: This button is active when the displayed data is followed by the ordered data. Pressing the button displays data with an offset value that is set to 100 greater than the current offset.
10. Textbox for entering the offset value for the displayed data: 100 data items from the specified order are displayed.
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.

10.3.8 Menu Composition

10.3.8.1 File Menu

- **Exit**: Closes the application.

10.3.8.2 Edit Menu

- **Copy**: Copies to the clipboard any data selected on the list.
- **Select All**: Selects all of the data displayed on the data list.

10.3.8.3 Operation Menu

- **get CSV**: Opens the Get CSV dialog box (see section 10.3.9.1 Get CSV Dialog Box).
- **get UserData**: Opens the Get User Data dialog box (see section 10.3.9.2 Get UserData Dialog Box).
- **Delete Entry**: Opens the Delete Entry dialog box (see section 10.3.9.3 Delete Entry Dialog Box).

10.3.8.4 Setting Menu

- **use UTC**: When activated, displays time in the Coordinated Universal Time (UTC) format.
- **Proxy Server Settings**: Opens the **ProxyServerSettings** dialog box to perform settings for communication via a Proxy server.
- **Public Server**: When activated, connects to the release product general-purpose ranking server.

10.3.8.5 Help Menu

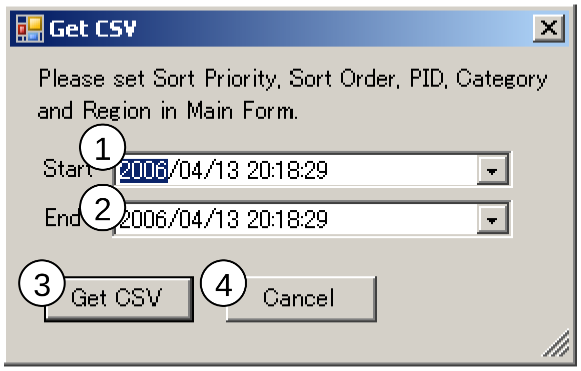
- **Version**: Displays version information.

10.3.9 Dialog Boxes

10.3.9.1 Get CSV Dialog Box

Figure 10-4 shows the dialog box for getting the data in a CSV file format.

Figure 10-4 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

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 and closes the dialog box without getting a CSV file.

Once the data is retrieved, the **File Save** dialog box opens for the save location to be specified.

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 “_”.

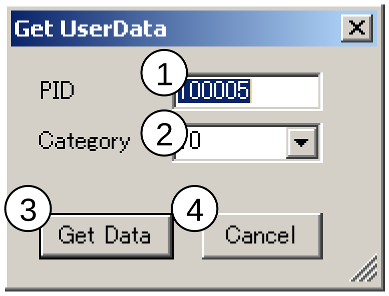
Up to 5,000 entries can be obtained for a single CSV file. A warning appears if that limit is exceeded. If this occurs, shorten the time frame or find some other way to reduce the number of entries and try again.

Getting many entries can take a minute or longer. Because intense processing is required to obtain a CSV, do this only once every 30 minutes, at most.

10.3.9.2 Get UserData Dialog Box

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

Figure 10-5 Get UserData Dialog Box



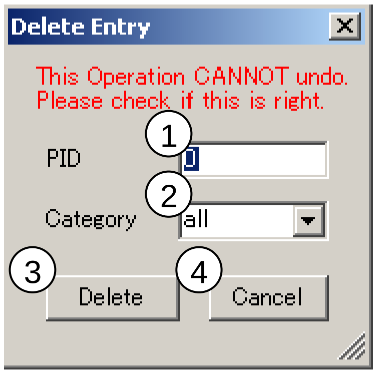
1. Specify the PID for the data to be retrieved. If the data is selected in the data list when this dialog box opens, the PID for the selected data appears here when the dialog box opens.
2. Specify the category ID for the data to be retrieved. If the data is already selected in the data list when this dialog box opens, the category ID for the selected data appears here when the dialog box opens. “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.

10.3.9.3 Delete Entry Dialog Box

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

Figure 10-6 Delete Entry Dialog Box


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

1. Specify the PID for the data that to be deleted. If the data is already selected in the data list when this dialog opens, the PID for the selected data appears here when the dialog box opens.
2. Specify the category ID for the data that to be deleted. If the data is already selected in the data list when this dialog opens, the category ID for the selected data appears here when the dialog box opens. 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.

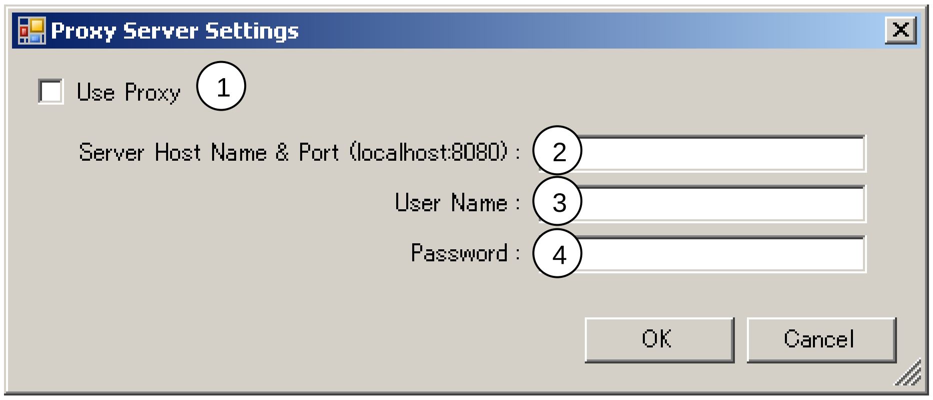
When data is deleted, the data displayed in the list is automatically updated.

Note: On the production server, you cannot delete data when “all” is specified.

10.3.9.4 Proxy Server Settings Dialog Box

This dialog box is used for setting the proxy server.

Figure 10-7 Proxy Server Settings Dialog Box



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.

10.3.10 Shortcut Keys

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

- CTRL+C: Copies the data selected in the data list to the clipboard.
- CTRL+A: Selects all the data displayed in the data 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.

10.4 Web Services Development

10.4.1 Web Services

The general-purpose 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.

10.4.2 Using Web Services

10.4.2.1 Access

To get the general-purpose ranking library data, access the URLs below. A tab-delimited CSV file can then be obtained 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 by Nintendo.

10.4.2.2 Security

For security purposes, there are limitations on the IP addresses that can access the general-purpose ranking library Web services. When using these services, contact support@noa.com with the global IP address for the accessing device.

There are no access limitations for the development server.

10.4.2.3 Data Formats

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

The row elements include, in order: 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 “_”.

10.4.2.4 Load on the Server

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

10.4.2.5 Get Parameters

The get method can be specified by providing parameters in the request to the general-purpose ranking server.

The request parameters take 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 10-7. Not all of the parameters need to be specified. Default values are applied for any omitted parameters.

Table 10-7 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
```

11 Download Service

11.1 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
- Extracting files by adding attributes
- Adding 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

You must follow prescribed procedures to use the Wi-Fi Connection Download Service. As the first step, inform Nintendo at support@noa.com that you plan to use the 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.

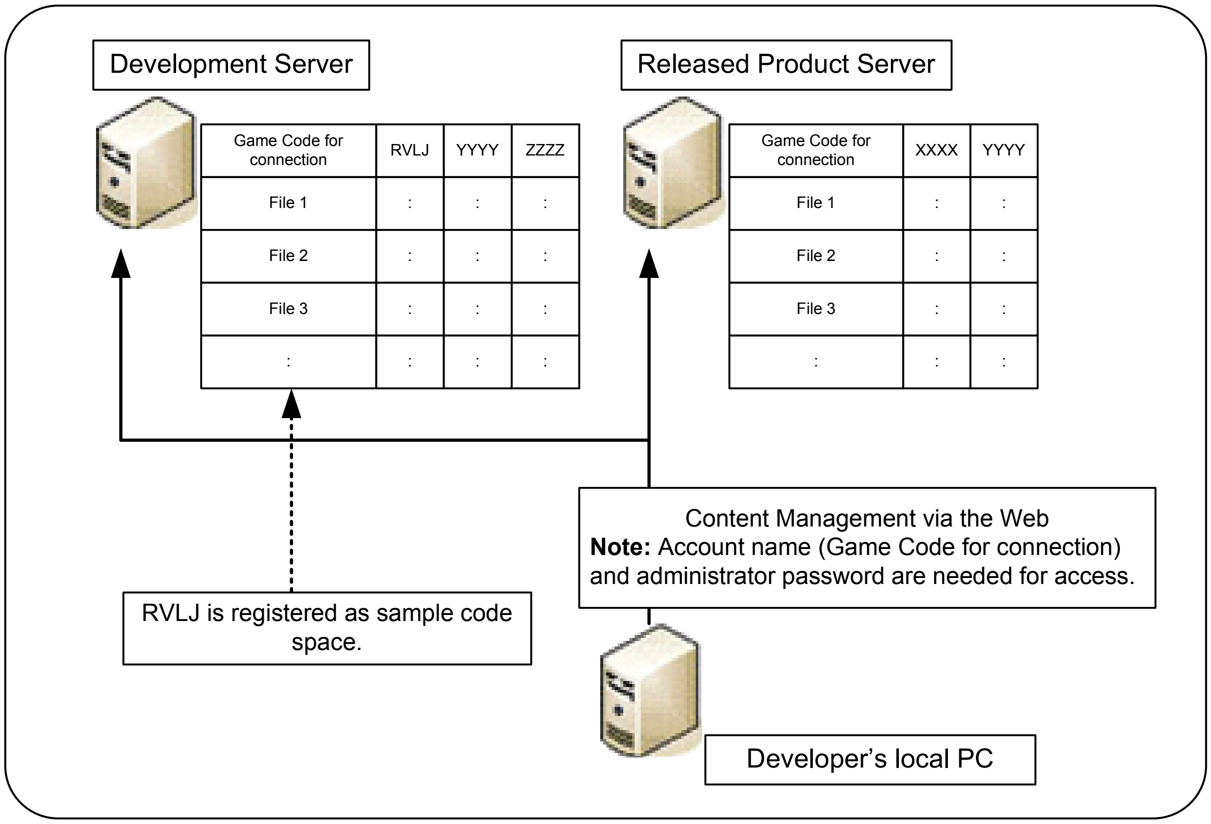
11.2 Overview

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

11.2.1 Structure of the Download Server

As shown in Figure 11-1, two types of download servers are used for content registration: the development and the production servers. Development and debugging takes place on the development server, while a released product’s ROM uses the production server. On the client side, switching of the download server is linked to the settings in the `DWC_Init` function.

Figure 11-1 Structural Diagram of the Server


Development Server
Released Product Server
RVLJ is registered as sample code space.
Content Management via the Web
Note: Account name (Game Code for connection) and administrator password are needed for access.
Developer's local PC

Game Code for connection	RVLJ	YYYY	ZZZZ
File 1	:	:	:
File 2	:	:	:
File 3	:	:	:
:	:	:	:

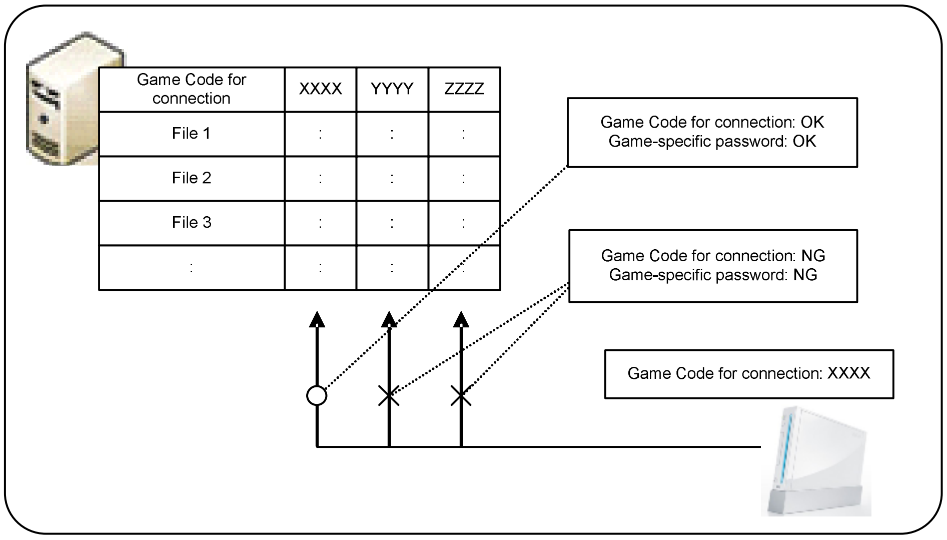
Game Code for connection	XXXX	YYYY
File 1	:	:
File 2	:	:
File 3	:	:
:	:	:

After applying to use Nintendo Wi-Fi Connection Download Service, disk space and an administrative screen are assigned for each game on each 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.

11.2.2 Content Confidentiality

Disk space where the content is registered 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 game-specific password after receiving the application to use Nintendo Wi-Fi Connection Download Service.

Figure 11-2 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.

11.2.3 Content Attributes

Content attributes refer to the maximum of 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 determines if the get operation is possible. As a result, controlling these attributes allows files to be filtered out or get restrictions to be used.

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 blank.

Table 11-1 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 11-1 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	“”	“”	“”	
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”	

11.2.4 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

11.2.5 Demo Programs

The connection Game Code for the demo program is “RVLJ” and content for the demo program is registered on the development server (see Table 11-2). 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. After connecting, the get file attributes are 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 11-2 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		
`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

11.3 Connecting to Nintendo Wi-Fi Connection

If 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.

11.3.1 Initialization

DWC-DL must be initialized with the `DWC_Init function` as would be done for the normal DWC package.

11.3.2 Creating User Data

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

11.3.3 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`.

11.3.4 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 the `DWC_NdProcess` function once each game frame.

11.3.5 Close Process

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

11.4 Downloads

11.4.1 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 11-1).

Because the initialization process will perform HTTP communications in the background, be sure to provide enough processing time to threads with a 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`. Also, if a callback function has been specified with the `DWC_NdInitAsync` function, that callback will be invoked.

Code 11-1 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 callback_nd( DWCNdCallbackReason reason,
                  DWCNdError error,
                  int servererror )
{
  // This callback is optional
}

void init_dwc_nd( void )
{
   if ( !DWC_NdInitAsync( callback_nd, gamecd, passwd ) )
   {
      error();           // error processing
      return;
   }

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

11.4.2 Extracting Files with Attribute Specifications

If the get file attributes are specified with the `DWC_NdSetAttr` function, the files to be downloaded can be extracted (see Code 11-2). Extraction takes place by specifying three attribute strings. However, if no attributes are assigned, extraction does not occur and all files are seen as downloadable. The attribute string is specified as a null-terminated ASCII string of 10 or fewer characters.

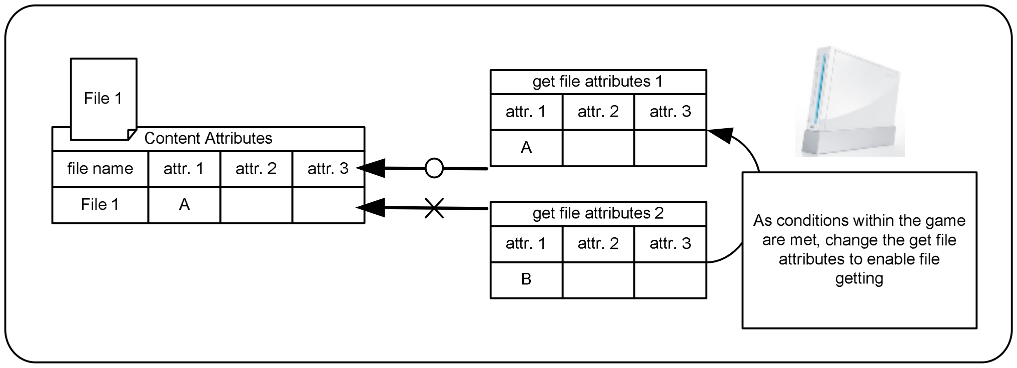
Code 11-2 Specifying Attributes

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

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

This extraction can also be used 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 11-3 Get Limitations Based on Get File Attributes



11.4.3 File Downloads

When downloading files, a call to the `DWC_NdGetFileListNumAsync` function gets the total number of downloadable files. Similarly, a call to the `DWC_NdGetFileListAsync` function gets either a part or all of the downloadable file list. The file name, game description, attributes, and file size are stored in the file list, so the files to be downloaded can clearly be indicated to the user.

Start the file download as follows (see Code 11-3).

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

Code 11-3 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;
}
```

11.4.4 Cancel Processing

File count acquisition, file list acquisition, and file download processes can be cancelled by calling the `DWC_NdCancelAsync` function (see Code 11-4).

Code 11-4 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.

11.4.5 Progress Level Checking

The download progress can be checked by calling the `DWC_NdGetProgress` function during file downloads (see Code 11-5).

Code 11-5 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 ));
    }
}
```

11.4.6 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 11-6).

Code 11-6 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();
}
```

11.5 Contents Management

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

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

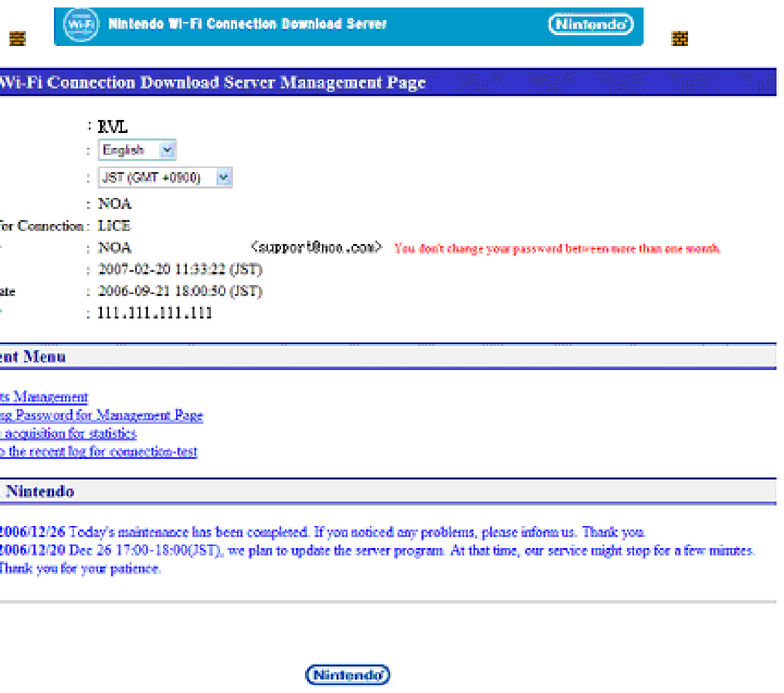
1. Using a Web browser on a PC, go to the URL sent by Nintendo.
2. At the authentication screen, enter the account name and administrator password issued by Nintendo.

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

11.5.2 Download Server Management Screen

The screen in Figure 11-4 is displayed upon logging into the Download Server Management screen.

Figure 11-4 Download Server Management Screen



Nintendo Wi-Fi Connection Download Server

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@noa.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.

11.5.2.1 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 the current time zone to be used for the management screen.
- Game Name: The target game name 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 used for the prior login is displayed.

11.5.2.2 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

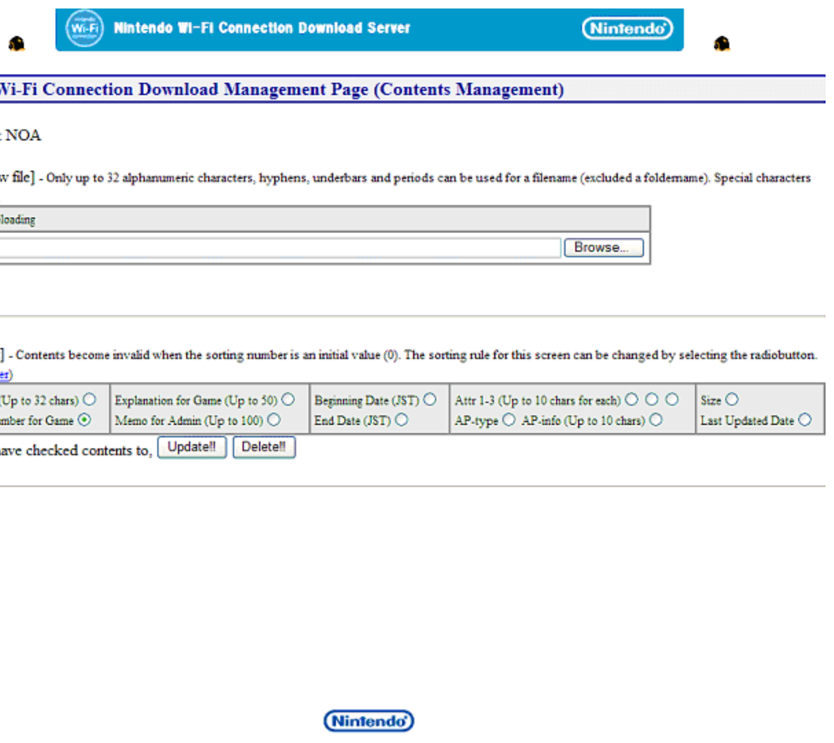
11.5.2.3 News from Nintendo

Notifications from Nintendo are displayed here.

11.5.3 Contents Management Screen

Downloadable content can be registered, confirmed, and configured on the Contents Management screen.

Figure 11-5 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,

11.5.3.1 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 a hard drive. Next, click **Upload** to register the content.

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

11.5.3.2 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 then click **Update** or **Delete**.

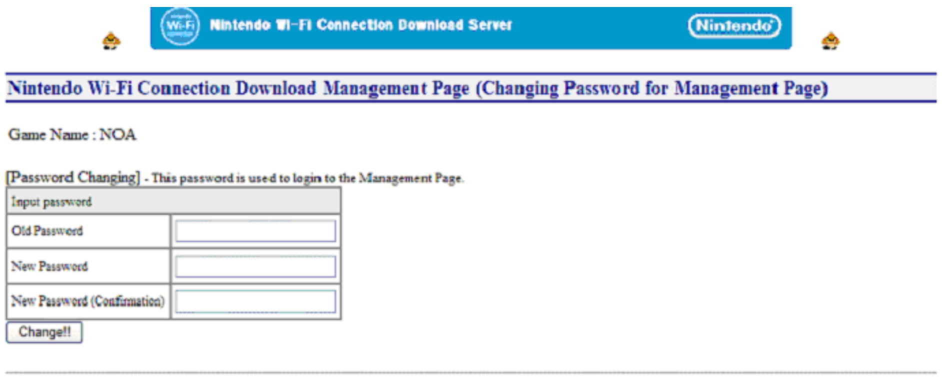
Table 11-3 Description of Fields on the Contents Management Screen

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 is retrieved with the file list. Referenced as a UTF16LE string.
Admin Notes	Space for the administrator to add notes.
Beginning Date End Date	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 was 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 get (see section 11.2.3 Content Attributes).
Size	Size of the registered file.
Last Updated Date	Date and time of the last content update.

11.5.4 Change Administrator Password Screen

The password for the download server administrator can be changed.

Figure 11-6 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!!

11.5.4.1 Changing Passwords

To change passwords, use the following procedure.

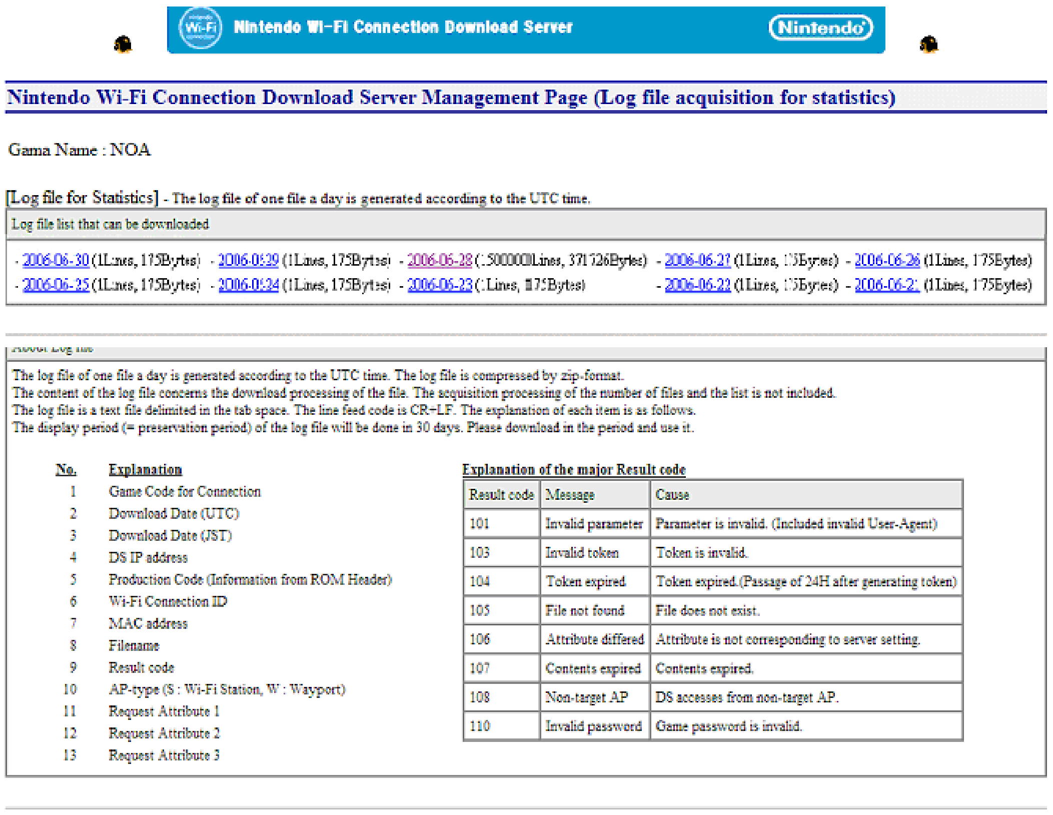
1. In the **Old Password** field, enter the current password.
2. In the **New Password** field, enter the new desired password.
3. In the **New Password (Confirmation)** field, enter the new password one more time to confirm it.
4. Click **Change** to change the password.

11.5.5 Get Statistical Log File Screen

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

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

Figure 11-7 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 (150000Lines, 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)

About Log file

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.

11.5.5.1 Log File for Statistics

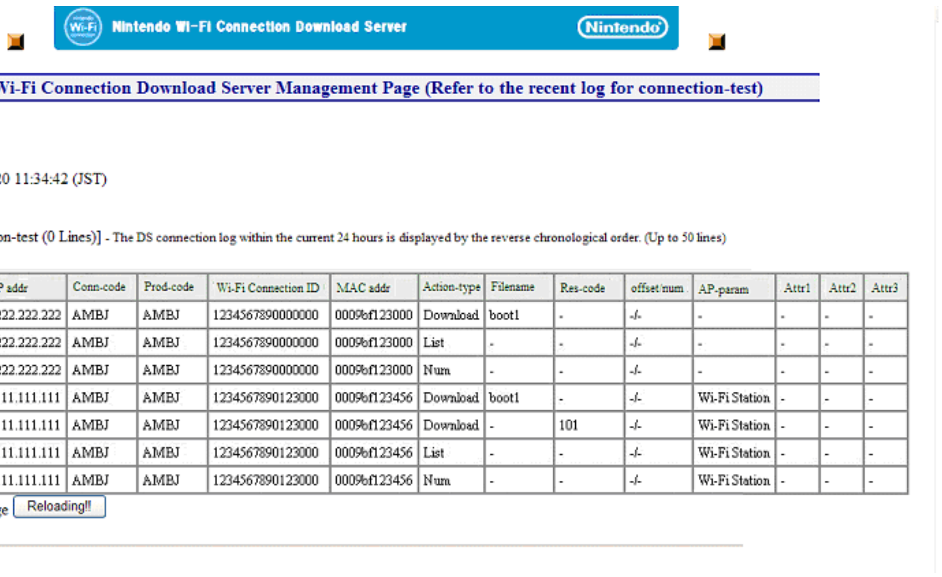
A log file can be obtained by clicking a date displayed in the downloadable log file list.

Note: The content of the log file differs from that described in section 11.5.6 Nearest Log Reference Screen for Connection Tests and is only for download processes. The retrieval of file lists and files is not logged.

11.5.6 Nearest Log Reference Screen for Connection Tests

In the Nearest Log Reference Screen for Connection Tests, the log information for the 50 most recent server accesses can be seen.

Figure 11-8 Nearest Log Reference Screen for Connection Tests



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!!

11.5.6.1 Nearest Log for Connection Tests

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 Game Code for connection to the download server that is provided by Nintendo
- Product Code: Game Code provided by Nintendo
- Wii Number: Displays the Wii Number 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 11-7.
- 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-letter string that follows the ISO3166 standardization.
- Region: The market code for the connected Wii console (00 – JPN, 01 – North America, and 02 – EUR)

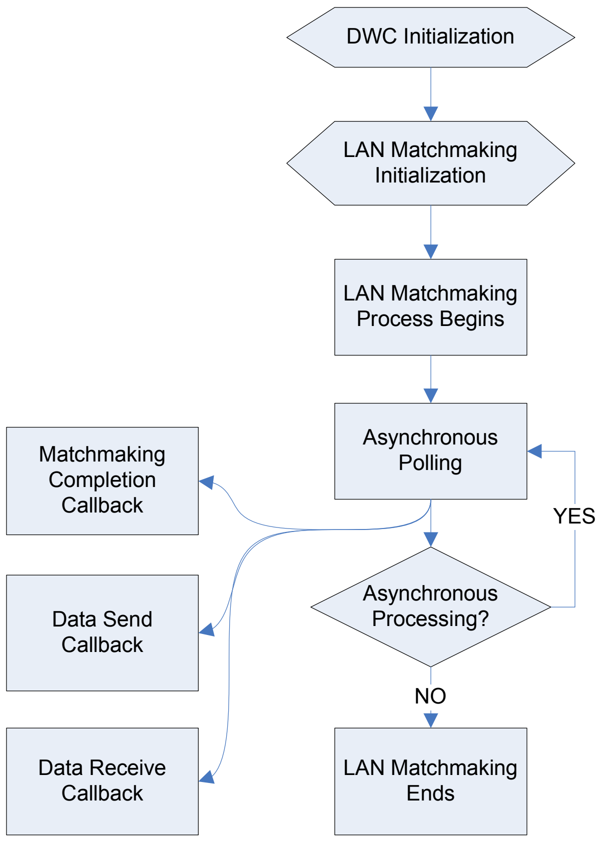
12 Intra-LAN Matchmaking

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, various other communication environments can also be tested.

12.1 Process Flow

Figure 12-1 LAN Matchmaking Process Flow



12.1.1 Initializing LAN Matchmaking

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

Code 12-1 Initializing a LAN Match

```
// LanMatch initialization
DWC_InitLanMatch( NULL );

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

12.1.2 Starting LAN Matchmaking

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

Code 12-2 Starting a LAN Match

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

// main loop
while (1) {

        DWC_ProcessLanMatch();

        :

}
```

12.1.3 Matchmaking Completion

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

Code 12-3 Matchmaking Completion Callback

```
void matchedCallback( DWCLanMatchResult result )
{

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

}
```

12.1.4 Sending and Receiving Data

Once the matchmaking is complete, data can be sent to or received from any peer 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.

An AID can be obtained 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 the user.

Code 12-4 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 12-5 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 );
}
```

12.1.5 Terminating LAN Matchmaking

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

12.2 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.

12.3 Coexistence of LAN Matchmaking and Other DWC Features

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

13 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.

13.1 Error Processing

The status of DWC errors can be obtained with the `DWC_GetLastErrorEx` function (see Code 13-1). The return value is the error type, the arguments are the error codes, and a pointer indicates 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, displaying the error is unnecessary when the error is recoverable and there is no disconnection from Nintendo Wi-Fi Connection.

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 13-1).

Code 13-1 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 or equal to 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 turning off power
           while (1) ;
     }
     // If a minor error, just clear it and you can restart the FriendsMatch process

     return err;
}
```

13.2 Error Code List

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

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 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: A fixed number of 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.

14 Differences from DWC1.x

This chapter lists the differences between DWC1.x and DWC2.x and associated precautions.

14.1 Matchmaking-Related Differences

- The callbacks for friend-specified and friend-unspecified matchmaking have been changed to be the same as for server-client matchmaking: `DWCMatchedSCCallback` and `DWCNewClientCallback`.

  In the case of friend-specified and friend-unspecified matchmaking, the matchmaking completion callback in DWC1.x was not called until the prescribed number of people had gathered. In DWC2.x, this callback is called for every new participant, just as is the case with server-client matchmaking.

- There is now a feature to reconnect using the group ID, which enables participants to rejoin groups after dropping out.

- The `distantFriend` argument has been eliminated from friend-specified matchmaking. The behavior of DWC2.x while new participants are being accepted is the same as when `distantFriend=TRUE` in DWC1.x.

- An argument for the connection topology has been added to the functions that start matchmaking.
  - `DWC_ConnectToAnybodyAsync`
  - `DWC_ConnectToFriendsAsync`,
  - `DWC_SetupGameServer`
  - `DWC_ConnectToGameServerAsync`
  - `DWC_ConnectToGameServerByGroupID`

  If there are many participants, setting the connection topology to star or hybrid will cause matchmaking to complete more quickly than in DWC1.x. Depending on the connection topology, there will be cases where mutual Reliable communications cannot take place.

- The callback function `DWCConnectAttemptCallback` has been added for all matchmaking types as a way to determine whether to accept the match.

- The matchmaking option APIs have been eliminated.
  - `DWC_SetMatchOption`
  - `DWC_GetMatchOption`
  - `DWC_GetMOMinCompState`
  - `DWC_GetMOSCConnectBlockState`
  - `DWC_ClearMOSCConnectBlock`
  - `DWC_StopSCMatchAsync`

  If you used these functions, investigate your application to see whether you can replace them with the suspend features of the `RequestSuspendMatchAsync` function. Below is one example of how to use the suspend features to replace these eliminated functions.

- `DWC_MATCH_OPTION_MIN_COMPLETE`

  Matchmaking completion is reported by a callback every time a person is connected. If the maximum number of people is not reached within the prescribed time, you can complete matchmaking at a number below the maximum by suspending the process using the `RequestSuspendMatchAsync` function.
- `DWC_MATCH_OPTION_SC_CONNECT_BLOCK`

  Note that the suspend features of the `RequestSuspendMatchAsync` function cannot exactly replicate the behavior of `DWC_MATCH_OPTION_SC_CONNECT_BLOCK`.
- `DWC_StopSCMatchAsync`

  For this, you can substitute the suspend features of the `DWC_RequestSuspendMatchAsync` function.

• `(void* param)` has been added to the functions `DWCUserSendCallback`, `DWCUserRecvCallback`, `DWCUserRecvTimeoutCallback` and `DWCUserPingCallback` so optional parameters can be passed from the application.