

Nintendo Wi-Fi Connection

TWL DWC Programming Manual

Version 2.0.10

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

Confidential

These coded instructions, statements, and computer programs contain proprietary information of Nintendo and/or its licensed developers and are protected by national and international copyright laws. They may not be disclosed to third parties or copied or duplicated in any form, in whole or in part, without the prior written consent of Nintendo.

Table of Contents

1	Introduction	8
2	User Management Under TWL DWC	9
2.1	Managing Wi-Fi User Information	9
2.1.1	User ID and Player ID	9
2.1.2	Difference Between a User ID and Player ID	10
2.1.3	Player Information by Game: Login ID	10
2.1.4	Information for Nintendo Wi-Fi Connection Authentication Saved by Games	11
2.2	Friend Management Overview	12
2.2.1	Building Friend Relationships	12
2.2.2	Building Friendships Using DS Wireless Communications	13
2.2.3	Building Friendships Using Friend Registration Keys	13
2.2.4	Friend Information Saved by Games	14
2.3	Exception Handling	14
2.3.1	Removing the Association Between a Nintendo DS System and a DS Card	14
3	TWL DWC Initialization	16
4	Creating User Data	18
5	Connection Process	20
5.1	Connecting to the Internet	20
5.2	Disconnecting from the Internet	21
5.3	Connecting to the Nintendo Wi-Fi Connection Server	21
6	Creating Friend Rosters and Information	24
6.1	Exchanging Friend Information via DS Wireless Communications	24
6.2	Exchanging Friend Registration Keys	25
6.3	Synchronizing Friend Rosters	26
6.4	Getting Friend Information Types	29
6.5	Getting Friend Status	30
7	Matchmaking	32
7.1	Peer Matchmaking with Friend Unspecified	32
7.2	Peer Matchmaking with Friend Specified	34
7.3	Evaluating Candidate Players for Matchmaking	36
7.4	Server-Client Matchmaking	37
7.5	Reconnection Using the Group ID	39
7.6	Closing Participation	41
7.7	ConnectAttemptCallback	42
7.7.1	Differences Between DWCEvalPlayerCallback and DWCCConnectAttemptCallback	44

7.8	Server Migration	45
7.9	Increasing Matchmaking Speed	46
7.10	Names That Cannot Be Used for Matchmaking Index Keys	47
8	Sending and Receiving Data	48
8.1	Peer-to-Peer Data Exchange	48
8.2	Connection Configurations	51
8.3	Closing Connections.....	52
8.4	Estimated Buffer Sizes to Specify with DWC_InitFriendsMatch	52
8.5	Emulating Delays and Packet Loss	53
8.6	Amount of Data Sent and Received	54
9	Communication Errors.....	56
9.1	Error Handling	56
9.2	List of Error Codes.....	57
10	Network Storage Support	59
11	Differences Between Versions Before NITRO DWC 3.X.....	63
11.1	Matchmaking	63

Code

Code 3-1	DWC Initialization	16
Code 4-1	Creating User Data.....	18
Code 4-2	Saving User Data	19
Code 5-1	Connecting to the Internet	20
Code 5-2	Disconnecting from the Internet	21
Code 5-3	Connecting to the Nintendo Wi-Fi Connection Server	22
Code 6-1	Exchanging Friend Information Using DS Wireless Communications	24
Code 6-2	Exchanging Friend Registration Keys	25
Code 6-3	Friend Roster Synchronization Process.....	27
Code 6-4	Getting Friend Information Types	29
Code 6-5	Getting a Friend's Status	30
Code 7-1	Peer Matchmaking with Friend Unspecified	33
Code 7-2	Peer Matchmaking with Friend Specified	35
Code 7-3	Evaluating Candidate Players for Matchmaking	36
Code 7-4	Server/Client Matchmaking	38
Code 7-5	Getting the Group ID	39
Code 7-6	Reconnecting from the Group ID.....	40
Code 7-7	Close-Participation Process	42
Code 7-8	ConnectAttemptCallback Example.....	43
Code 8-1	Setup for Data Exchange	48

Code 8-2 Sending Data	50
Code 8-3 Emulating Delays and Packet Loss	53
Code 9-1 Error Handling Process	56
Code 10-1 Accessing the Storage Server	59

Tables

Table 7-1 Key Names That Cannot Be Used for Matchmaking Index Keys	47
Table 8-1 Estimated Buffer Sizes	53
Table 8-2 Communication Data Breakdown	54

Figures

Figure 2-1 Save State of the User ID on the Nintendo DS System and DS Card	9
Figure 2-2 Using Multiple Nintendo DS Systems and DS Cards	9
Figure 2-3 How Data Is Stored on the Internet	10
Figure 2-4 Configuration of a Login ID	11
Figure 2-5 Comprehensive Diagram of Terminology for Nintendo Wi-Fi Connection Authentication	12
Figure 2-6 Creating Friendships Using DS Wireless Communications	13
Figure 2-7 Creating Friendships Using Friend Registration Keys	14

Revision History

Version	Revision Date	Description
2.0.10	2009/12/21	Revised sections 7.7 ConnectAttemptCallback and 7.8 Server Migration to state that server migration occurs during server-client matchmaking.
2.0.9	2009/11/25	In Chapter 11 Differences Between Versions Before NITRO DWC 3.X, explained automatic disconnection of a host in the absence of communications after a fixed period of time.
2.0.8	2009/09/01	Revised Chapter 3 TWL DWC Initialization, Chapter 4 Creating User Data, and section 5.3 Connecting to the Nintendo Wi-Fi Connection Server. Explained splitting the <code>DWC_Init</code> function into a development and production version, and the corresponding changes to the arguments of the <code>DWC_CreateUserData</code> and <code>DWC_InitFriendsMatch</code> functions. Corrected mistakes in the handling of the <code>s_userdata</code> variable in Code 6-1 Exchanging Friend Information Using DS Wireless Communications and Code 6-2 Exchanging Friend Registration Keys.
2.0.7	2009/06/30	Fixed a formula for calculating the estimated buffer size for reliable communications in section 8.4 Estimated Buffer Sizes to Specify with <code>DWC_InitFriendsMatch</code> .
2.0.6	2009/05/25	Added a note to section 8.1 Peer-to-Peer Data Exchange, stating that debug output may cause packets to appear to be delayed.
2.0.5	2009/04/08	Fixed cross-reference to section 5.1 Connection to the Internet (Japanese version only).
2.0.4	2009/03/23	Changed the term, "profile ID" to "GS profile ID" for consistency.
2.0.3	2009/02/02	Added description to sections 7.7 ConnectAttemptCallback and 7.8 Server Replacement that there is no server replacement in server-client matchmaking. Corrected Code 7-8 ConnectAttemptCallback Example along with a change in the <code>DWC_GetAIDList</code> function specification.
2.0.2	2008/12/18	Corrected a typo in the Japanese version of the manual.
2.0.1	2008/11/28	Deleted Chapter 9 HTTP Communication along with making the GHTTP library private.
2.0.0	2008/10/10	Transitioned from NITRO DWC to TWL DWC. Made specifications changes specific to matchmaking with TWL DWC 5.0.
1.4.3	2007/07/21	Corrected an error in Code 7.3 Evaluating Candidate Players for Matchmaking: Changed <code>s_int_key</code> to <code>&s_int_key</code> .
1.4.2a	2007/04/27	Corrected typos and changed dates to international format.
1.4.2	2007/02/15	Revised text in Code 6-1 Exchanging Friend Information Using DS Wireless Communications: Changed <code>s_friendData</code> to <code>ownFriendData</code> .
1.4.1	2006/08/09	Revised text in section 8.4 Yardstick for Buffer Size Specified by <code>DWC_InitFriendsMatch</code> and Table 8-2 Communication Data Breakdown.
1.4.0	2006/06/19	Changed the conditions for displaying error codes in section 9.1 Error Handling.
1.3.0	2006/06/06	Revised the section "Examples of When a Temporary login ID May be Duplicated" in section 2.1.3 Player Information by Game: login ID.

Version	Revision Date	Description
		Changed the memory size to 230 KB from 200 KB in Chapter 3 Initializing NITRO-DWC. Added section 7.10 Names That Cannot Be Used for Matchmaking Index Keys. Miscellaneous changes (unified terminology, made corrections, and so on).
1.2.0	2006/03/10	Added Chapter 2 User Management Under NITRO-DWC. Added section 7.9 Increasing Matchmaking Speed. Added section 8.6 Amount of Data Sent/Received. Miscellaneous changes (review of text, changes in terminology, and so on).
1.1.0	2006/01/30	Updated Code 6-3 Synchronizing Friend Rosters. Corrected error in Code 6-4 Friend Information Types" (from "stablished" to "established"). Corrected error in Code 7-3 Evaluating Candidate Players for Matchmaking" (from "anymatch" to "anymatch test"). Changed data load function in Chapter 10 Accessing the Storage Server to a newly added function.
1.0.0	2005/12/28	Initial version.

1 Introduction

The TWL DWC library (DWC library) is designed with the goal of making Nintendo Wi-Fi Connection easy to use and free of charge. Specific benefits include the following.

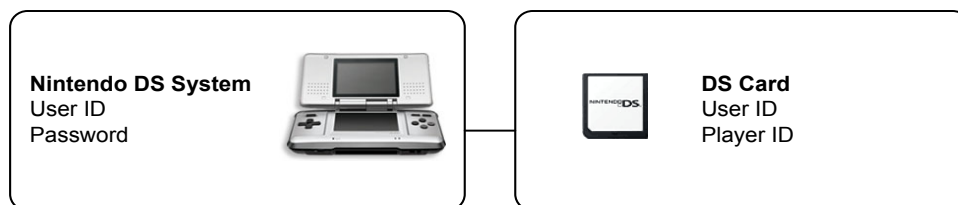
- Making it easy to connect by sheltering users from complicated and detailed Internet settings
- Making it easy to communicate with friends with whom friendships were established by using wireless communications or by exchanging friend registration keys when not connected to the Internet
- Making it easy to remain secure by ensuring that one user cannot easily access another user's Internet-related information when a Nintendo DS system changes hands

2 User Management Under TWL DWC

2.1 Managing Wi-Fi User Information

Information required for Nintendo Wi-Fi Connection authentication includes a user ID, player ID, and password. This information is managed while treating the Nintendo DS system and DS Card as a pair (see Figure 2-1).

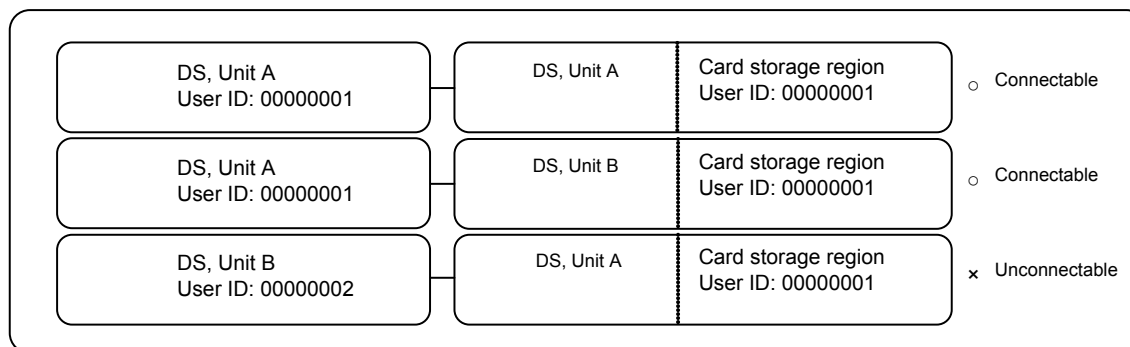
Figure 2-1 Save State of the User ID on the Nintendo DS System and DS Card



- The user ID and password used for Nintendo Wi-Fi Connection authentication are saved on the Nintendo DS system
- The user ID and player ID used for Nintendo Wi-Fi Connection authentication are saved on the DS Card

This information is used by Nintendo Wi-Fi Connection for authentication. If the user ID saved on the DS Card differs from the user ID saved on the Nintendo DS system, data saved on Nintendo Wi-Fi Connection cannot be accessed. This prevents the unauthorized access of data (see Figure 2-2).

Figure 2-2 Using Multiple Nintendo DS Systems and DS Cards



2.1.1 User ID and Player ID

The user ID is generated offline and is designed to be as unique as possible. After it is generated, it becomes the user ID for connecting to the Internet, and authenticating and registering with the system. If the ID is found to already be in use during authentication, a new, unique user ID is assigned.

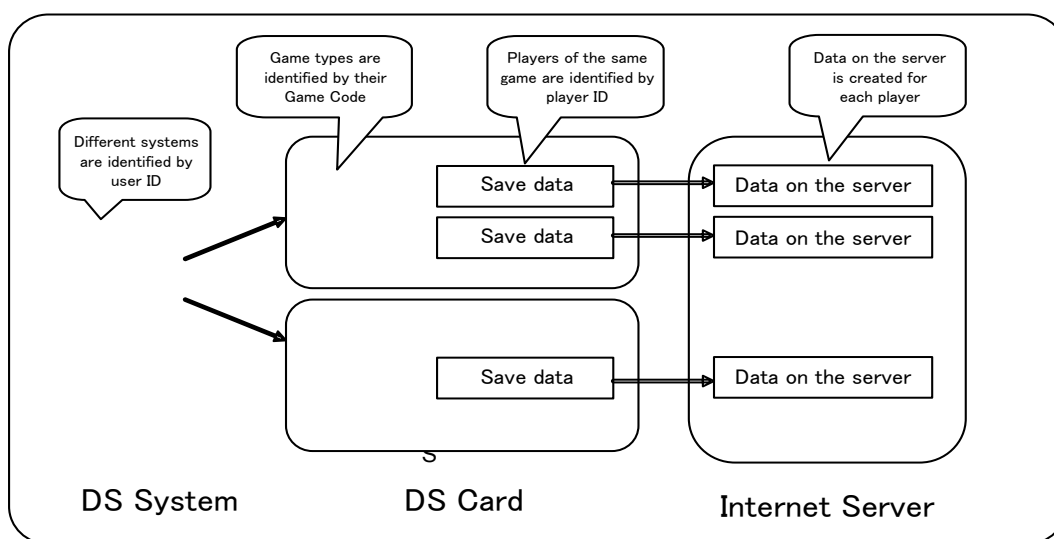
Note: To ensure that the user ID is unique, part of the Nintendo DS system's MAC address is used. Although this prevents the same user ID from being used on different Nintendo DS systems, duplication might occur when a user ID is moved or regenerated.

The player ID is a random 32-bit ID. Because data on the Internet server is managed using the combined user ID, player ID, and Game Code, a player ID only needs to be unique with respect to the user ID and Game Code. If the player ID is duplicated, a unique player ID will be assigned during authentication.

2.1.2 Difference Between a User ID and Player ID

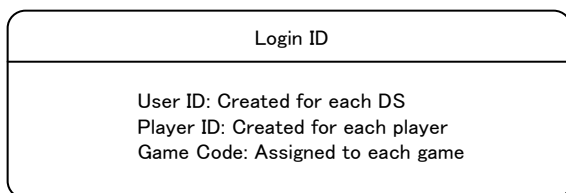
Because a user ID is issued to each Nintendo DS system, a user that uses the same system must use a single user ID for all games. Since player IDs are issued to DS Cards, you can use different player IDs when using the same Nintendo DS system (user ID) and the same Game Code (see Figure 2-3).

Figure 2-3 How Data Is Stored on the Internet



2.1.3 Player Information by Game: Login ID

The combined user ID + player ID + Game Code are called the “login ID” (see Figure 2-4). User information saved on the Internet server is called a “profile,” and the ID used to manage profiles on the server is called a “GS profile ID.”

Figure 2-4 Configuration of a Login ID

Inside the DWC library, the login ID or GS profile ID is used to search for the profiles of other users on the Internet server.

The login ID is generated when not connected to the Internet and becomes a temporary login ID. Although a user is likely to use this login ID as is, it might not be available. In this case, a unique, approved login ID (authenticated login ID) is generated. There is a one-to-one correspondence between authenticated login IDs and assigned GS profile IDs.

A temporary login ID may be duplicated under the following circumstances.

- The login ID is created with a user ID that was not authenticated, the same user ID already is registered in the Authentication server by another person, and the login ID was created with the same player ID for the same game
- Multiple Nintendo DS systems created login IDs with the same player ID for the same game using the same unauthenticated user ID

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

Games must save this information for Nintendo Wi-Fi Connection authentication as backup on the DS Card.

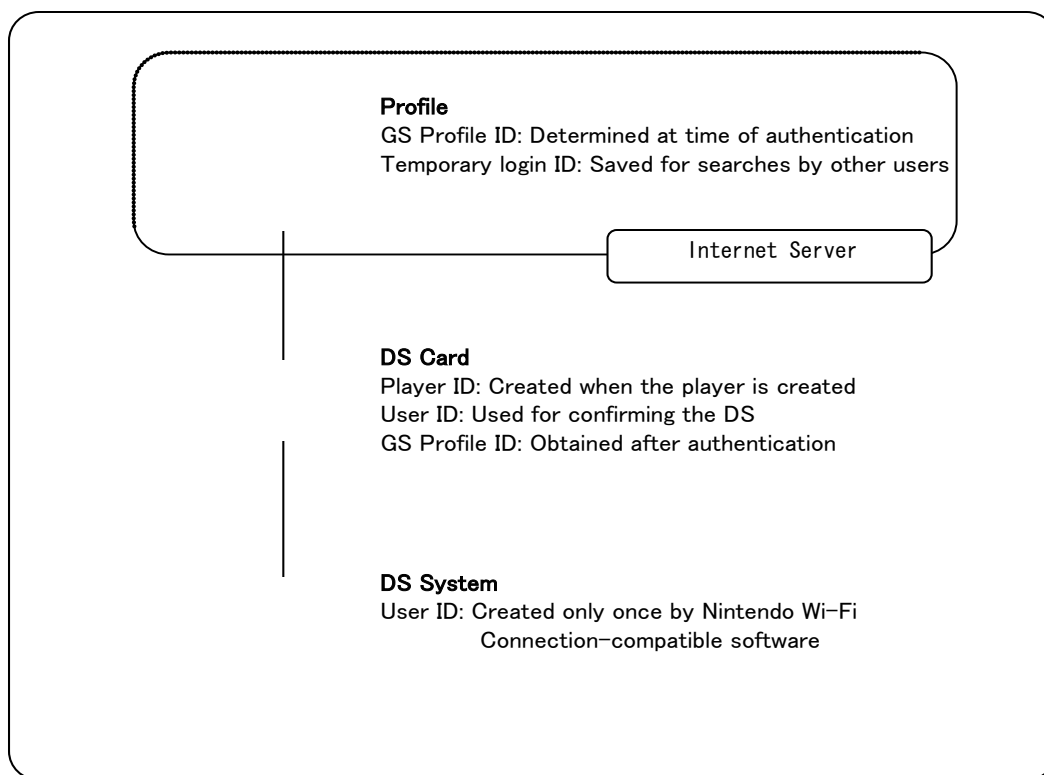
The size of the information used for authentication is 64 bytes.

The Nintendo Wi-Fi Connection authentication information includes the temporary login ID, the authenticated login ID, and the GS profile ID. Developers do not need to fully understand the details of this because this information is created and updated by the DWC library.

Information for Nintendo Wi-Fi Connection authentication must also be saved for each player when multiple players can use the same DS Card.

Figure 2-5 shows the Nintendo Wi-Fi Connection authentication terminology covered so far.

Figure 2-5 Comprehensive Diagram of Terminology for Nintendo Wi-Fi Connection Authentication



2.2 Friend Management Overview

2.2.1 Building Friend Relationships

To be able to easily start communication with friends using DWC, friend relationships are built by an Internet server. Friendships are built by exchanging user information. Established friendships are saved in the profile of each user.

There are two ways to exchange the user information used to create a friendship.

- Using DS Wireless Communication

Using this method, the players exchange login or GS profile IDs. The login ID is used if the player in question has never logged in before. Even though each of these was created locally, it is highly likely that they are unique, but not guaranteed. However, because the probability of duplication is less than 2^{-75} , no special countermeasure against duplication is required. The GS profile ID is used for players who have logged in at least once before. This creates friendships with certainty, because a particular party can always be specifically identified.

- Exchanging friend registration keys

Using this method, the players exchange friend registration keys, included in the GS profile ID, as information used for error checking. A player must have connected to the Internet at least once to use a GS profile ID. You must create an interface that allows input to be confirmed and re-entered in case it is incorrectly input.

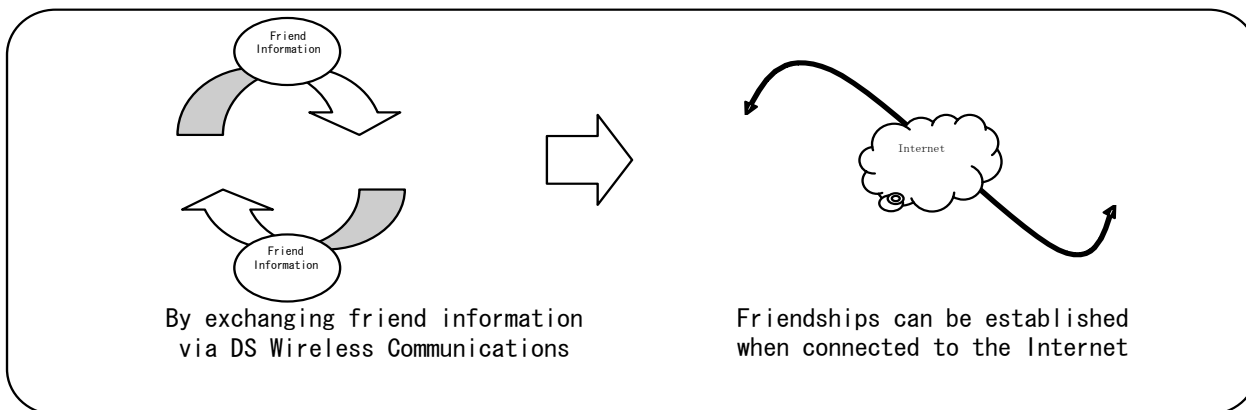
The information exchanged can be created using DWC. DWC includes functions for automatically creating the most applicable information possible based on information used for Nintendo Wi-Fi Connection authentication saved on the DS Card.

2.2.2 Building Friendships Using DS Wireless Communications

A mechanism is provided to allow friendships to be automatically established later on the Internet when information is exchanged with another party during DS Wireless Communications. The information exchanged is created from the login ID or GS profile ID included in user data.

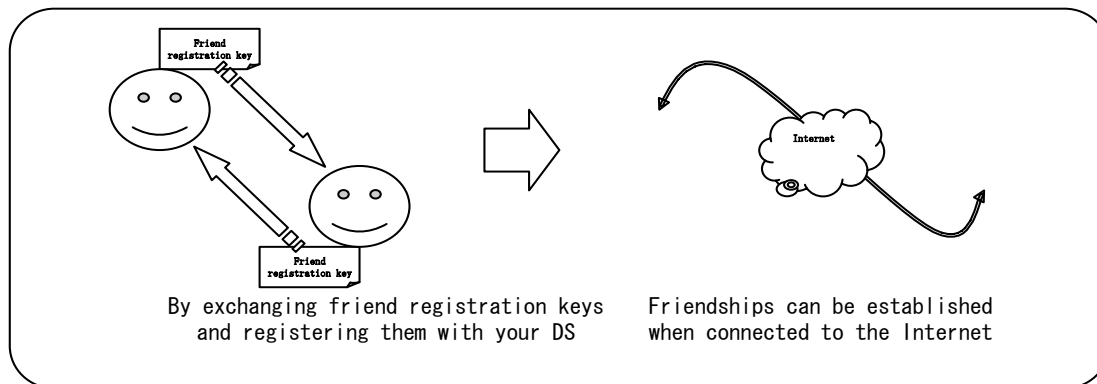
Note: The exchange of this information through DS Wireless Communications is not supported by DWC. Be sure that applications handle the exchange of created information.

Figure 2-6 Creating Friendships Using DS Wireless Communications



2.2.3 Building Friendships Using Friend Registration Keys

The term “friend registration key” refers to information that can be used to specifically identify another user when establishing a friendship. A mechanism is provided that allows friendships to be created by exchanging this friend registration key (see Figure 2-7). Because the friend registration key is manually entered by users, it should not be unnecessarily long. It is created using the GS profile ID obtained by connecting at least once to the Internet rather than using the login ID.

Figure 2-7 Creating Friendships Using Friend Registration Keys

The friend registration key is a 12-digit number.

Pay attention to the following points when developing games.

- You must create a user interface for issuing friend registration keys because a key cannot be issued unless a player connects to the Internet at least once. A message to this effect must be displayed.
- You must create a user interface for entering the friend registration key. The user interface must allow the user to correct an incorrectly input friend registration key. It should also allow users to save and edit the entered data as many times as necessary.

2.2.4 Friend Information Saved by Games

Games must save exchanged friend information for the maximum number of players to be managed as friends in a backup area. This is required so users can edit friendships when they are not connected to the Internet. Friend-related information used by the actual game (such as nicknames and win-loss record) must also be saved. DWC treats all of this as friend information without regard to the type of data (login ID, GS profile ID, and friend registration key).

Twelve bytes per player are required to store friend information used by DWC.

2.3 Exception Handling

2.3.1 Removing the Association Between a Nintendo DS System and a DS Card

For security reasons, Nintendo Wi-Fi Connection treats the Nintendo DS system and DS Card as a set. This can be inconvenient for a user if the Nintendo DS system is resold or broken, as the ability to connect to Nintendo Wi-Fi Connection is lost.

To solve this problem, there is a DWC mechanism that allows the user to delete the data that associates a DS Card with a given Nintendo DS system by destroying information stored in the profile. Because this deletes all Internet friendships, you must create an interface to warn the user before deleting the data.

Even if Internet friendships are deleted, friend information for other parties remains on the DS Card of the deleted user. This allows friendships to be restored by using this information and sending a new friend registration key to the other party. Because it is necessary in these cases to prompt the user to register the deleted user as a friend again, each application needs to include a message for notifying the user of the required procedure.

With regard to specific processing, the currently saved association on the DS Card is deleted. If a user wants to create a new association, it must be handled by creating new user data and destroying the previous user data. Even if user data is updated, friendships on the friend roster saved on the DS Card remain established. If a specification where the friend roster remains intact is used, be sure to clear the friendship-established flag included in the friend information when letting the user know that friendships remain established.

Note: See the flowchart in the *Nintendo Wi-Fi Connection Programming Guidelines*.

3 TWL DWC Initialization

You must initialize the DWC library before calling any of its functions, except

DWC_Debug_DWCInitError or DWC_SetReportLevel.

The `DWC_InitForDevelopment` and `DWC_InitForProduction` functions initialize the DWC library as follows.

- Configure either the development or production server to be used
- Set the memory allocation function used by the DWC library
- Generate information for user authentication stored in the Nintendo DS system
- Check if the connection target information stored in the Nintendo DS system's backup memory is valid

Code 3-1 DWC Initialization

```
void init dwc( void )
{
    int ret;
    ret = DWC_InitForDevelopment("dwctest", 'NTRJ', AllocFunc, FreeFunc);
    //ret = DWC_InitForProduction("dwctest", 'NTRJ', AllocFunc, FreeFunc);

    // Initialize the DWC library
    if ( ret == DWC_INIT_RESULT_DESTROY_OTHER_SETTING )
        disp_init_warning_msg(); // Display warning message
}

// Function for allocating memory
void* AllocFunc( DWCAAllocType name, u32 size, int align )
{
    void * ptr;
    OSIntrMode old;
    (void)name;
    (void)align;

    old = OS_DisableInterrupts();

    ptr = OS_AllocFromMain( size );

    OS_RestoreInterrupts( old );

    return ptr;
}
```



```
// Function for freeing memory
void FreeFunc( DWCAAllocType name, void* ptr, u32 size )
{
    OSIntrMode old;
    (void)name;
    (void)size;

    if ( !ptr ) return;

    old = OS_DisableInterrupts();

    OS_FreeToMain( ptr );
    OS_RestoreInterrupts( old );
}
```

Initialize the library with `DWC_InitForDevelopment` or `DWC_InitForProduction` to use the development or production authentication server, respectively. The authentication server that you use determines the server that is used by the download library and other functions. For more details, see the Function Reference Manual for the DWC initialization functions.

If you implement a sequence to delete DS Card backup data, we recommend calling the initialization function at the same time as that sequence.

The DWC library requires approximately 230 KB of memory for four-player matchmaking. The required memory size decreases by approximately 20 KB each time the maximum matchmaking count is reduced by one. This is a valid approximation as long as the `sendBufSize` and `recvBufSize` arguments to the `DWC_InitFriendsMatch` function are each set to their default values of 8 KB.

4 Creating User Data

The DWC library performs typical processes based on user data.

- Authenticating users
- Creating friend relationships

Even when the Nintendo DS system is not connected to the Internet, it requires user data to create the friend information that is exchanged to create friend relationships via DS Wireless Communications.

If user data is not yet created or the user data is damaged, create the user data with the `DWC_CreateUserData` function and store the user data in the DS Card backup memory.

Be sure the application allocates memory for saving the `DWCUserData` structure. User data for several people is required when a single DS Card supports multiple players.

If player data is already created, be sure to check its validity using the `DWC_CheckUserData` function after loading it from backup into memory (see Code 4-1).

Code 4-1 Creating User Data

```
BOOL create_userdata( void )
{
    // If there is backup data and user data in that backup data, load all and
    // return TRUE.
    if ( DTUDs CheckBackup() )
    {
        (void)DTUD_LoadBackup( 0, &s_PlayerInfo, sizeof(DTUDPlayerInfo) );

        OS_TPrintf("Load From Backup\n");

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

    // If valid user data has not been saved
    OS_TPrintf("no Backup UserData\n");

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

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

    return FALSE;
}
```

Use the `DWC_CheckDirtyFlag` function to check whether it is necessary to save user data to the DS Card. Always use the `DWC_ClearDirtyFlag` function to clear the `DirtyFlag` before saving the user data to backup memory as shown in 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 );
        DTUD_SaveBackup( 0, &s_PlayerInfo.userData, sizeof(DWCUserData) );
    }
}
```

Before connecting to the Internet, be sure to check user data according to the following procedure.

- Use the `DWC_CheckHasProfile` function to check whether the user has already connected to the Internet and obtained a profile in the user data. If there is no profile, the user data is updated and the Nintendo DS system and DS card are treated as a set.
- Check whether the Nintendo DS system and DS Card are being used correctly using the `DWC_CheckValidConsole` function. It is impossible to connect to the Internet if the Nintendo DS system and DS Card are not correct because authentication will fail.

Note: Be sure to check the flowcharts included in *Nintendo Wi-Fi Connection Programming Guidelines*.

5 Connection Process

The DWC library performs a two-phase process when connecting to the Internet.

- Connects to the Internet (making a Nintendo Wi-Fi Connection to get an IP address)
- Connects to the Nintendo Wi-Fi Connection server (referred to as "server")

When a Nintendo DS system connects to the Internet for the first time, the Nintendo authentication server issues a user ID for that system. This user ID is stored in the Nintendo DS system backup memory.

After this initial connection is established, the DWC library stores this user ID and the player ID in the previously created user data to generate a profile. The GS profile ID that corresponds to this generated profile is stored in the user data.

5.1 Connecting to the Internet

When the Nintendo DS system first connects to the Internet to get the IP address, Nintendo's authentication server issues a user ID to that system. Tests are also performed to confirm that the Nintendo DS system can connect to the connection test server using TCP communication and that the Internet connection is functioning normally.

All these processes are performed automatically by calling the `DWC_*Inet` functions, as shown in Code 5-1.

Code 5-1 Connecting to the Internet

```
static DWCInetControl s_ConnCtrl; // Retain until the Internet connection is
disconnected
BOOL connect to inet( void )
{
    // Initialization process for Internet connection
    DWC InitInet( &s_ConnCtrl );

    // Start establishing connection
    DWC ConnectInetAsync();

    // The connection process
    while ( !DWC CheckInet() )
    {
        DWC ProcessInet();

        // V-Blank wait process
        // During the connection process you need to pass the
        // process time to threads that have lower priority than
        // the main thread. Use the OS_WaitIrq function for this.
        GameWaitVBlankIntr();
    }
}
```

```
// Confirm the connection result
if ( DWC_GetInetStatus() != DWC_CONNECTINET_STATE_CONNECTED )
{
    handle_error();
    return FALSE;
}
// Connected
:
```

5.2 Disconnecting from the Internet

Call the `DWC_CleanupInet*` functions as shown in Code 5-2 to disconnect the Nintendo DS system from the Internet.

Even if a communication error occurs and the Nintendo DS system is disconnected automatically, you must call this function because the library memory needs to be freed.

Code 5-2 Disconnecting from the Internet

```
void disconnect_func( void )
{
    while ( !DWC_CleanupInetAsync() )
    {
        GameWaitVBlankIntr();
    }
    :
}
```

5.3 Connecting to the Nintendo Wi-Fi Connection Server

To connect to the Nintendo Wi-Fi Connection server, use the `DWC_InitFriendsMatch` function shown in Code 5-3 to initialize matchmaking and friend relationship features.

The following arguments are given to this function.

- Pointer to user data
- Product ID provided by GameSpy
- Secret key provided by GameSpy
- Send and receive buffer sizes used for communication between Nintendo DS systems
- Pointer to the friend roster
- Maximum number of friends in the friend roster

The specified control objects are used in the DWC library until the `DWC_ShutdownFriendsMatch` function is called.

Chapter 8 Sending and Receiving Data describes the sizes of the Send and Receiver buffers in detail. When 0 is specified, as is the case in the sample program below, the buffers use 8 KB by default.

The friend roster is an array of friend information in the `DWCFriendData` structure. Chapter 6 Creating Friend Rosters and Information discusses friend rosters and friend information in detail.

Next, call the `DWC_LoginAsync` function to make the connection to the server (see Code 5-3).

The first argument of this function is the player's screen name. If players use names in your game application, you must specify the screen name in this argument. The screen name used in the game is sent to the authentication server to confirm and check for inappropriate names.

You can check the results of this function by calling the `DWC_GetIngamesnCheckResult` function (see Code 5-3).

The second argument of the `DWC_LoginAsync` function is not currently used. Pass `NULL` for this argument. The remaining arguments represent the callback to use after login completes and the parameters of the callback.

After calling this function, call the `DWC_ProcessFriendsMatch` function repeatedly to advance the login process, approximately once per game frame.

Next, the `DWC_ProcessFriendsMatch` function executes all matchmaking and friend-related processing until the `DWC_ShutdownFriendsMatch` function is called. After login completes, be sure to call `DWC_ProcessFriendsMatch` function to make sure that network processes (for example, updating the friend roster) do not start while the Nintendo DS system is connected to another client.

Code 5-3 Connecting to the Nintendo Wi-Fi Connection Server

```
static BOOL s_logged = FALSE;

void connect_to_wifi_connection( void )
{
    DWC_InitFriendsMatch( DTUD_GetUserData(),
                        GAME_PRODUCTID, GAME_SECRET_KEY,
                        0, 0,
                        DTUD_GetFriendList(), FRIEND_LIST_LEN );

    // Login using function for authentication
    s_logged = FALSE;
    if ( !DWC_LoginAsync( L"name", NULL, cb_login, NULL ) )
    {
        // Connection process fails to start.
        return;
    }

    // Polling to see if connected
    while ( !s_logged )
    {
        DWC_ProcessFriendsMatch();

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

```
        GameWaitVBlankIntr();
    }

    // Connection process completed
    if ( DWC_GetIngamesnCheckResult() == DWC_INGAMESN_INVALID )
    {
        // Special process performed when inappropriate in-game screenname was detected
        disp_ingamesn_warning();
    }
    :
}

// Callback when logged in
void cb_login( void )
{
    if (error == DWC_ERROR_NONE)
    {
        check_and_save_userdata();

        s_logged = TRUE;
    }
}
```

The `DWC_ShutdownFriendsMatch` function ends the matchmaking and friend relationship features and frees the memory reserved internally by the library.

When the Nintendo DS system connects to the server for the first time using the user data specified by the `DWC_InitFriendsMatch` function, the Nintendo DS system and the DS Card are treated as a pair. When they are treated as a pair, the DS Card that stores the specified user data cannot be used with another Nintendo DS system.

The user data is always updated when the first connection is made. Once the login completes, the application should call a login callback and the `DWC_CheckDirtyFlag` function to check the updated user data. If necessary, save the updated data to the DS Card.

6 Creating Friend Rosters and Information

The DWC library has two procedures for establishing friend relationships among players.

- Exchanging friend information using DS Wireless Communications
- Exchanging friend registration keys

6.1 Exchanging Friend Information via DS Wireless Communications

During DS Wireless Communications, the `DWC_CreateExchangeToken` function is used to create friend information based on the local user data for exchange with other players (see Code 6-1).

Friend information that the Nintendo DS system receives should be saved in the friend roster using the application.

Code 6-1 Exchanging Friend Information Using DS Wireless Communications

```
DWCUserData    s userData;
DWCFriendData s_friendList[ FRIEND_LIST_LEN ];

// Exchange friend information
void exchange_friend_data( void )
{
    int i, j;

    DWCFriendData ownFriendData;
    DWCFriendData recvFriendList[ FRIEND_LIST_LEN ];

    // Create friend information from local user data to send
    DWC_CreateExchangeToken( &s userData, &ownFriendData );

    // Send and receive friend information via MP communication
    MP_start( (ul6 *)&ownFriendData, (ul6 *)&recvFriendList );
    :

    // Save the received friend information in an open slot in the friend roster.
    // Do not save if the same friend information already exists.
    for ( i = 0; i < num recv data; ++i )
    {
        int index;
        for ( j = 0, index = -1; j < FRIEND_LIST_LEN; ++j )
        {
            if ( DWC_IsValidFriendData( &s_friendList[ j ] ) )
            {
                // If the friend roster has valid data, check if it is the same as
                // the received friend information and do not save if it is the same.
                if ( DWC_IsEqualFriendData( &recvFriendList[ i ],
                                            &s_friendList[ j ] ) )
                {
                    break;
                }
            }
            else
            {
                // Records an available friend roster index
                if ( index == -1 ) index = j;
            }
        }
    }
}
```



```

    }

    // Save valid friend information that does not overlap in friend roster
    if ( j >= FRIEND_LIST_LEN && index >= 0 )
    {
        s_friendList[ index ] = recvFriendList[ i ];
    }
}
:
}

```

6.2 Exchanging Friend Registration Keys

A player that has connected at least once to Nintendo Wi-Fi Connection is assigned a GS profile ID, which is saved in the user data. Any player that has a GS profile ID can create a friend registration key that adds special error-checking information to the GS profile ID. This friend registration key is a 12-digit decimal number that players can exchange. Once this friend registration key has been entered, friend data can be exchanged.

After the friend registration key is entered, the `DWC_CreateFriendKeyToken` function is called to convert the key into friend information and save the friend information to the friend roster (see Code 6-2).

Use the `DWC_CheckFriendKey` function to check if the entered friend registration key is valid as shown in Code 6-2. Even if this function is called, the error does not correct itself, so prepare a user interface so that the user can enter the key until the key information is correct.

Code 6-2 Exchanging Friend Registration Keys

```

DWCUserData  s userData;
DWCFriendData s friendList[ FRIEND_LIST_LEN ];
// Display Friend Registration Key
void disp_friend_key( void )
{
    u64 friend_key;
    // Create friend registration key from local user data
    if ( ( friend_key = DWC_CreateFriendKey( &s userData ) ) != 0 )
    {
        // Display friend registration key
        disp_message( "FRIEND CODE : %lld", friend_key );
    }
    else
    {
        // Display message that there is no friend registration key
        disp_message( "FRIEND CODE : not available" );
    }
    :
}

/* Create friend information from friend registration key and register in friend roster */
BOOL register_friend_key( void )
{
    u64 friend_key;
    DWCFriendData friendData;

```

```

while ( 1 )
{
    char friend_key_string[ 13 ];

    // Get user to manually enter friend registration key
    input_friend_key( friend_key_string );

    /* Convert entered friend registration key string into u64 numerical value */
    friend_key = charToU64( friend_key_string );

    // Check that friend registration key is correct and proceed if OK.
    // If there is a problem, display message and have it entered again
    if ( DWC_CheckFriendKey( &s_userData, friend_key ) ) break;
    else disp_warning_message();
}

// Create Friend information from correct Friend Registration Key
DWC_CreateFriendKeyToken( &friendData, friend_key );

{
    int index;
    /* Using same method as MP communication, search for open slot and
       overlaps in friend roster and register friend information. */
    :
    s_friendList[ index ] = friendData;
    :
}
}

```

6.3 Synchronizing Friend Rosters

For a friend roster stored in the application (local friend roster) to be valid on the Internet, you need to call the `DWC_UpdateServersAsync` function and update the friend roster stored on the GameSpy server (server friend roster) as shown in Code 6-3.

To synchronize the friend rosters, you must first complete the login process with the `DWC_LoginAsync` function.

Specify the following function arguments: the player name (the old specification; specify `NULL`), the callback and its parameters when the friend roster completes synchronization, the callback and its parameters for a change notification in friend status (discussed later), and the callback and its parameters when the friend roster is deleted.

The friend roster synchronization process involves two main tasks: sending requests to establish friend relationships for friends that are on the local but not the server friend roster, and deleting friends that are on the server but not the local friend roster.

If a request to establish a friend relationship is sent while the other party is offline, call the `DWC_LoginAsync` function to save the request on the server and immediately deliver the request the next time the contacted partner logs in. The friend relationship is only established after the information is saved in the local friend roster of the other party.

Note that this process only registers the other party as your friend. When the other party receives the request to establish a friend relationship, the contacted partner follows the same process to register the initiating partner as a friend.

After the friend roster synchronization process completes, the callback is called after the local and server friend rosters are checked, needed requests to establish friend relationships are sent, and unneeded friend information is deleted. Be aware that even if the callback has returned, this state does not indicate that all friend relationships are established. If the `isChanged` argument of the callback is set to `TRUE`, this indicates that the friend information in the local friend roster is updated and needs to be saved. If a friend relationship is established at a time other than during the friend roster synchronization process, the callback for an established friend relationship specified by the `DWC_SetBuddyFriendCallback` function is called.

If multiple sets of friend information for the same friend are discovered during the friend roster synchronization process, all but one set are automatically deleted. A callback is called for each deleted set by comparing the friend roster index of the deleted friend information and the friend roster index of the matching friend.

Code 6-3 Friend Roster Synchronization Process

```

BOOL s_update          = FALSE;
BOOL s_updateFriendList= FALSE;

void sync_friend_list( void )
{
    // Set the callback for establishment of friend relationship
    DWC_SetBuddyFriendCallback( cb_buddyFriend, NULL );

    // Synchronize local friend roster and server friend roster
    if ( !DWC_UpdateServersAsync( NULL,
                                cb_updateServers, NULL,
                                NULL, NULL,
                                cb_deleteFriend, NULL ) )
    {
        // Synchronization process fails to start
        return;
    }

    while ( !s_update )
    {
        DWC_ProcessFriendsMatch();

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

        GameWaitVBlankIntr();
    }
    :

    while ( 1 )

```

```

{
    DWC_ProcessFriendsMatch();

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

    // To update the friend roster asynchronously, perform the following
    // processing when appropriate and collect the updated local friend roster
    // and save
    if ( s_updateFriendList )
    {
        // Save the friend roster if it has been updated
        s_updateFriendList = FALSE;
        save_friendList();
    }

    game_loop();

    GameWaitVBlankIntr();
}
:
}

// Callback for when friend roster synchronization has completed
void cb_updateServers( DWCErr error, BOOL isChanged, void* param )
{
    if ( error == DWC_ERROR_NONE )
    {
        // Friend roster synchronization successful
        s_update = TRUE;

        // Must be saved if friend roster has been changed
        if ( isChanged ) s_updateFriendList = TRUE;
    }
}

// Callback for when there is a friend roster deletion
void cb_deleteFriend( int deletedIndex, int srcIndex, void* param )
{
    OS_TPrintf( "friend[%d] was deleted (equal friend[%d]).\n",
                deletedIndex, srcIndex );
    s_updateFriendList = TRUE;
}

// Callback for when friend relationship has been established
void cb_buddyFriend( int index, void* param )
{
    OS_TPrintf( "Got friendship with friend[%d].\n", index );
    s_updateFriendList = TRUE;
}

```

6.4 Getting Friend Information Types

Code 6-4 shows how you can get the data type set in the friend information using the `DWC_GetFriendDataType` function.

The possible data types are listed below.

- `DWC_FRIENDDATA_NODATA` No stored friend information
- `DWC_FRIENDDATA_LOGIN_ID` ID for the state when a connection to Nintendo Wi-Fi Connection has never been made
- `DWC_FRIENDDATA_FRIEND_KEY` Friend registration key
- `DWC_FRIENDDATA_GS_PROFILE_ID` GS profile ID

When the contacted partner has not yet obtained a GS profile ID, the data type `DWC_FRIENDDATA_LOGIN_ID` indicates that friend information was downloaded via DS Wireless Communications.

Once the contacted partner has obtained a GS profile ID and initiating partner has completed the friend roster synchronization process, the data type changes to `DWC_FRIENDDATA_GS_PROFILE_ID`.

The data type `DWC_FRIENDDATA_FRIEND_KEY` indicates that the friend relationship is not yet established for the GS profile ID registered using the friend registration key. Once the friend relationship is established, the data type changes to `DWC_FRIENDDATA_GS_PROFILE_ID`.

You can use the `DWC_IsBuddyFriendData` function to determine whether a friend relationship has been established from the friend information.

Code 6-4 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 ] );
        OS_TPrintf( "friend[%d] type %d.\n", type );

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

6.5 Getting Friend Status

All players maintain their own status when using Nintendo Wi-Fi Connection. Nintendo Wi-Fi Connection is managed by a server operated by GameSpy.

There are two player states that the application can reference.

- The communication state
- A status string or binary data

The communication state is defined by the `DWC_STATUS_*` constants, which are set automatically by the DWC library. The application sets the status string with the `DWC_SetOwnStatusString` function and the binary data with the `DWC_SetOwnStatusData` function as shown in Code 6-5.

Status strings must be null-terminated and can be up to 256 text characters long, including the null terminator. Binary data are converted inside the function into a string, and the approximate number of text characters will be data size x 1.5. The string should not include '/' or '\\' because these text characters are used by the library as identifiers.

The current status of a friend can be obtained if a friend relationship has been established. Specify a friend status change callback as the argument in the `DWC_UpdateServersAsync` function to enable a user to receive notices whenever friend status changes.

To get friend status, use the `DWC_GetFriendStatus*` function group. For this group of functions, communication doesn't occur while accessing the friend status list maintained by the DWC library. However, processing these functions takes several hundred microseconds, so take care when calling the functions frequently over a short period of time.

If there is a sudden loss of power during communication, the player's status will remain in the previous state for a few minutes.

Code 6-5 Getting a Friend's Status

```
void sync friend list( void )
{
    int i;

    // Synchronize local friend roster and server friend roster
    if ( !DWC_UpdateServersAsync( NULL,
                                cb_updateServers, NULL,
                                cb_friendStatus, NULL,
                                NULL, NULL ) )
    {
        // Synchronization process fails to start
        return;
    }
    :

    // Friend roster synchronization completed
    :

    // Set local status test 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 friend information is valid, get the status of that friend
        status = DWC_GetFriendStatus( &friendList[ i ], statusString );

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

// Callback notifying change in friend's status
void cb_friendStatus( int index, u8 status, const char* statusString, void* param )
{
    OS_TPrintf( "Friend[%d] status -> %d (statusString : %s).\n",
                index, status, statusString );
}
```

7 Matchmaking

The DWC library provides two matchmaking methods: peer matchmaking and server-client matchmaking.

In peer matchmaking, the Nintendo DS systems are not distinguished as servers and clients. There are two implementation methods.

- Friend unspecified
- Friend specified

7.1 Peer Matchmaking with Friend Unspecified

This method performs matchmaking for players in the general public.

Call the `DWC_ConnectToAnybodyAsync` function as shown in Code 7-1 to begin peer matchmaking without specifying friends. The function is passed these arguments.

- The connection configuration (see section 8.2 Connection Configurations)
- The maximum desired number of connected players including the local player
- A filter string for matchmaking conditions
- A matchmaking completion callback and its parameters
- A newly connected client notification callback and its parameters
- A player evaluation callback and its parameters (described below)
- A matchmaking condition determination callback and its parameters
- The matchmaking condition values

The matchmaking completion callback is called when the local player successfully connects to the server host. It is also called when a new client host is added to the group to which the local player belongs. The newly connected client notification callback is called when a client host starts a new connection to the group to which the local player belongs.

Use the filter string to narrow the search for matchmaking candidates. The matchmaking index keys (in Code 7-1, the key names are `str_key` and `int_key`) need to be registered in advance using the `DWC_AddMatchKey*` functions. The key names are saved inside the library, but only pointers to the key values are stored in the library. Consequently, you should retain key values until matchmaking completes.

Note: There are certain names that cannot be used as matchmaking index keys. For details, see section 7.10 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 );

    // Start matchmaking without specifying friends
    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 );

    // Poll to see if matchmaking has completed
    while ( !s_matched )
    {
        DWC_ProcessFriendsMatch();

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

        GameWaitVBlankIntr();
    }

    // Matchmaking has completed
    :
}
```

```
// Callback for when matchmaking has completed
void cb_sc_match( DWCErrror 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 )
        {
            // Either the local player has cancelled matchmaking, or the local
            // player is the client host and the server host cancelled matchmaking
            s_cancelled = TRUE;
        }
        // Does nothing even if the newly connected client cancels matchmaking
    }
}

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

7.2 Peer Matchmaking with Friend Specified

This method performs matchmaking for friends registered in friend rosters.

Call the `DWC_ConnectToFriendsAsync` function as shown in Code 7-2 to begin peer matchmaking by specifying friends. The function is passed these arguments.

- The connection configuration (see section 8.2 Connection Configurations)
- The friend roster index array (index list) of friends to perform matchmaking
- The number of elements in the index list
- The maximum desired number of connected players including the local player
- A matchmaking completion callback and its parameters
- A newly connected client notification callback and its parameters
- A player evaluation callback and its parameters (described below)
- A matchmaking condition determination callback and its parameters
- The matchmaking condition values

The matchmaking completion callback is called when the local player successfully connects to the server host and is also called when a new client host is added to the group to which the local player belongs. The newly connected client notification callback is called when a client host starts a new connection to the group to which the local player belongs.

If `NULL` is specified for the index list, all friends in a friend roster are treated as matchmaking candidates. Peer matchmaking with friend specified uses the `DWC_InitFriendsMatch` function to specify the friend roster.

Code 7-2 Peer Matchmaking with Friend Specified

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

void do_friend_match( void )
{
    // Start matchmaking with specifying friends
    DWC_ConnectToFriendsAsync(DWC_TOPOLOGY_TYPE_FULLMESH, NULL, 0, 4,
                             cb_sc_match, NULL,
                             cb_sc_new, NULL,
                             NULL, NULL, NULL );

    // Poll to see if matchmaking has completed
    while ( !s_matched )
    {
        DWC_ProcessFriendsMatch();

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

        GameWaitVBlankIntr();
    }

    // Matchmaking has completed
    :
}

// Callback for when matchmaking has completed
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 )
        {
            // Either the local player has cancelled matchmaking, or the local
            // player is the client host and the server host cancelled matchmaking
            s_canceled = TRUE;
        }
        // Does nothing even if the newly connected client cancels matchmaking
    }
}
```

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

7.3 Evaluating Candidate Players for Matchmaking

During peer matchmaking, players who have been identified as matchmaking candidates can be evaluated using game-specific criteria listed in order of preference. When an evaluation callback is set as an argument of the function that starts peer matchmaking, that callback is called every time a player is identified as a possible matchmaking candidate during matchmaking.

Use the `DWC_GetMatch*Value` functions inside this callback to reference the matchmaking index keys that were registered by the `DWC_AddMatchKey*` functions as shown in Code 7-3. Evaluate each player based on these values and use the evaluated value as the return value. Players whose evaluated value is less than zero are removed as matchmaking candidates.

Note that this method is designed to make selecting players with the highest evaluated values easier, but this method does not guarantee that players with the highest evaluated values will be selected for matchmaking.

Code 7-3 Evaluating Candidate Players for Matchmaking

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

void do_anybody_match( void )
{
    // Set matchmaking index keys
    DWC_AddMatchKeyString( 0, "str key", s_str_key );
    DWC_AddMatchKeyInt( 0, "int key", &s_int_key );

    // Start matchmaking by specifying friends
    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 )
    {
        // Sees which are close to local value and takes it as evaluated value
        return MATH_ABS( s_int_key - eval_int ) + 1;
    }
}
```

```
else
{
    // Does not match make players that do not have the int_key key
    return 0;
}
```

Even though matchmaking is not established with players that have different connection configurations (see section 8.2 Connection Configurations), the evaluation callback is called. If this happens, the return values of the evaluation callback do not influence the matchmaking results.

7.4 Server-Client Matchmaking

In server-client matchmaking among friends, each host takes on a clearly defined role as a server or client. The server specifies the following:

- The connection configuration (see section 8.2 Connection Configurations)
- The number of players desired for connection (this number includes the server host)
- A matchmaking completion callback and its parameters
- A newly connected client notification callback and its parameters
- A matchmaking condition determination callback (described later) and its parameters
- The matchmaking condition values

The server calls the `DWC_SetupGameServer` function and then waits for the client to connect. The code for this process is shown in Code 7-4.

The clients specify the following:

- The connection configuration
- An index list of friends desiring connection
- A matchmaking completion callback and its parameters
- A newly connected client notification callback and its parameters
- A matchmaking condition determination callback (described later) and its parameters
- The matchmaking condition values

The client calls the `DWC_ConnectToGameServerAsync` function. With this function configuration, the clients will try to connect if matchmaking has started with the friend established as the server host.

When server-client matchmaking completes, the server has a friend relationship with every connected client. However, the clients may have friend relationships through their friends via their connection to the server.

The matchmaking completion callback is called when the client successfully connects to the server, and also when a new client is added to the group to which it belongs.

The newly connected client notification callback is called when a new client starts the connection to the group to which it belongs.

Code 7-4 Server/Client Matchmaking

```

static BOOL s_matched = FALSE;

void do_server_match( void )
{
    // Start matchmaking as 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 ) )
        {
            // Error generation.
            handle_error();
            return;
        }

        if ( s_matched )
        {
            // If connection has been made with new client
            init_new_connection();
            s_matched = FALSE;
        }

        GameWaitVBlankIntr();
    }
    :
}

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

    // Poll to see if matchmaking completed
    while ( !s_matched )
    {
        DWC_ProcessFriendsMatch();

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

        GameWaitVBlankIntr();
    }

    // Matchmaking completed

```

```

:
}

// Callback for when matchmaking completed
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 )
        {
            // The local system cancelled matchmaking, or the local system is a
            // client host and the server host cancelled matchmaking
            s_cancelled = TRUE;
        }
        /* Do nothing even if some other newly connecting client cancels matchmaking */
    }
}

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

```

7.5 Reconnection Using the Group ID

Both peer matchmaking and server-client matchmaking use a feature that allows a local player to rejoin a group. The player may have dropped out of the group because of a line disconnection, for example.

After the matchmaking completion callback is called successfully, the group ID that identifies the group in which the local player is participating can be acquired with the `DWC_GetGroupID` function as shown in Code 7-5.

Code 7-5 Getting the Group ID

```

static u32 s_groupID = 0 // ID of the group last participated in

// 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;

            // Get the group ID at this time

```

```

        s_groupID = DWC_GetGroupID();
    }
    else if ( self || isServer )
    {
        // Either the local player has cancelled matchmaking, or the local
        // player is the client host and the server host cancelled matchmaking
        s_cancelld = TRUE;
    }
    // Does nothing even if the newly connected client cancels matchmaking
}
}

```

The `DWC_ConnectToGameServerByGroupID` function is used for a local player to reconnect to a group using the group ID retrieved with the `DWC_GetGroupID` function. The function `DWC_ConnectToGameServerByGroupID` is called as shown in Code 7-6 with the following arguments.

- The connection configuration (see section 8.2 Connection Configurations)
- Group ID
- Matchmaking completion callback and its parameters
- Newly connected client notification callback and its parameters
- Matchmaking condition determination callback and its parameters (described below)
- The matchmaking condition values

The matchmaking completion callback is called when the local player successfully connects to the server and is also called when a new client is added to the group to which the local player belongs.

The newly connected client notification callback is called when a client starts a new connection to the group to which the local player belongs.

Code 7-6 Reconnecting from the Group ID

```

static u32 s_groupID = 0 // ID of the group last participated in
static BOOL s_matched = FALSE;

void do_groupID_match( void )
{
    // Start matchmaking by reconnecting from the 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 occurred.
            handle_error();
            return;
        }
        GameWaitVBlankIntr();
    }
}

```



```

    // Matchmaking completed
    :
}

// 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 )
        {
            // Either the local player has cancelled matchmaking, or the local
            // player is the client host and the server host cancelled matchmaking
            s_cancelled = TRUE;
        }
        // Does nothing even if the newly connected client cancels matchmaking
    }
}

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

```

Reconnecting from the group ID fails when the specified group has no participants or when the specified group is full.

7.6 Closing Participation

When the specified number for matchmaking is reached, no more new clients can participate. However, participation can be closed to new clients even when the specified number is not reached. Also, it is possible to return to accept-participation status from close-participation status.

To change the close-participation status, all participating hosts specify the close-participation status to configure and the close-participation status-change callback and its parameters, and then call the `DWC_RequestSuspendMatchAsync` function at the same time, as shown in Code 7-7.

Only when all participants set the same close-participation status and call the `DWC_RequestSuspendMatchAsync` function at the same time will the close-participation process complete.

For example, a flow must be established so that all participants call the `DWC_RequestSuspendMatchAsync` function with the same close-participation status arguments to guarantee close-participation processing when transitioning from the game setting screen to the actual game screen.

Use the `DWC_GetSuspendMatch` function to get the existing close-participation status.

Code 7-7 Close-Participation Process

```
static BOOL s_suspendCompleted = FALSE;

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

    // Close participation status change completion polling
    while ( !s_suspendCompleted )
    {
        DWC_ProcessFriendsMatch();
        if ( DWC_GetLastErrorEx( NULL, NULL ) )
        {
            // Error occurred.
            handle_error();
            return;
        }
        GameWaitVBlankIntr();
    }
    // Close participation status change completion
}

// Callback called when close participation completes
static void suspendCallback(DWCSuspendResult result, BOOL suspend,
                           void* data)
{
    if(result == DWC_SUSPEND_SUCCESS)
    {
        DWCDemoPrintf("The Close Participation Status is changed to [%s].\n",
                      suspend ? "TRUE" : "FALSE");
    }
    else
    {
        // Error
    }
    s_suspendCompleted = TRUE;
}
```

7.7 ConnectAttemptCallback

The application can set any value in the `connectionUserData` argument (the size is `DWC_CONNECTION_USERDATA_LEN`) for the following functions that start matchmaking, as shown in Code 7-8.

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

This value is sent to the server host that performs the role of accepting new client hosts to determine whether the local player can participate in matchmaking.

For matchmaking that specifies friends or does not specify friends, the application does not need to be aware of who becomes the server host. It is fine to think that all participants may be server hosts. The host that calls the `DWC_SetupGameServer` function during server-client matchmaking becomes the first server host; if it disconnects later, server migration occurs.

The server host receives a `DWCCConnectAttemptCallback` call in conjunction with the new client host's `connectionUserData`.

Whether to accept the new client is not determined at this time. The host that receives `DWCCConnectAttemptCallback` performs the determination based on the `connectionUserData` of the new client host passed in the callback argument. When accepted, `TRUE` is returned.

When the callback terminates with `TRUE`, the newly connected client host is accepted, and the process to connect to the other connected client hosts is started. When the connection process occurs, notification of the `connectionUserData` of the newly connected client host is sent to and shared with all client hosts.

The `connectionUserData` of already connected client hosts (hosts with a determined AID, including the local host) can be retrieved with the `DWC_GetConnectionUserData` function. It is assumed that this function will be used as material for determination received inside `DWCCConnectAttemptCallback`.

Code 7-8 ConnectAttemptCallback Example

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

:
{
    u8 userData[DWC_CONNECTION_USERDATA_LEN];
    :

    // Start matchmaking without specifying friends
    userData[0] = MALE;
    DWC_ConnectToAnybodyAsync( DWC_TOPOLOGY_TYPE_FULLMESH, 4,
                               NULL,
                               cb_sc_match, NULL,
                               cb_sc_new, NULL,
                               NULL, NULL,
                               ConnectAttemptCallback, NULL, userData );
    :
}

/* When a value is set to indicate male or female in the [0] of connectionUserData
 * of DWC_ConnectToXXX and the maximum number for both males and females is
 * limited.
 */
static BOOL ConnectAttemptCallback(u8* newClientUserData, void* param)
{
    int numAid;
    u8* aidList;
    u8* userData; [DWC_CONNECTION_USERDATA_LEN]
    int i;
```

```

int maleCount = 0;    // Number of males
int femaleCount = 0; // Number of females

(void) param;

numAid = DWC_GetAIDList(&aidList);

// If numAid was obtained, you are also in the list, so
// process in the loop.
// If numAid was not obtained, get your own data separately.
if(numAid == 0)
{
    if(male == USER_TYPE_MALE) ++maleCount;
    else ++femaleCount;
}
else
{
    for (i = 0; i < numAid; i++)
    {
        DWC_GetConnectionUserData(aidList[i], userData);
        if (userData[0] == USER_TYPE_MALE)
            maleCount++;
        else if (userData[0] == USER_TYPE_FEMALE)
            femaleCount++;
    }
}

DWCdemoPrintf("male:%d female:%d\n", maleCount, femaleCount);

if (data[0] == USER_TYPE_MALE && maleCount < USER_MAX_MALE)
    return TRUE;
else if (data[0] == USER_TYPE_FEMALE && femaleCount < USER_MAX_FEMALE)
    return TRUE;
else
    return FALSE;
}

```

7.7.1 Differences Between DWCEvalPlayerCallback and DWCCConnectAttemptCallback

Both `DWCEvalPlayerCallback` and `DWCCConnectAttemptCallback` can be used to narrow the matchmaking targets. Note the following differences between the two callbacks.

- `DWCEvalPlayerCallback` cannot be used for server-client matchmaking. Conversely, `DWCCConnectAttemptCallback` can be used for all matchmaking.
- Servers and clients narrow targets differently.
 - `DWCEvalPlayerCallback`

Called by the newly participating client host and then performs determination.
 - `DWCCConnectAttemptCallback`

Called by the server host accepting the newly participating client host, then performs determination.

- Available information may determine the narrowing method.

- DWCEvalPlayerCallback

Called by the new client host to retrieve information about the server host to which a connection is being made. The `DWC_GetMatchIntValue` and `DWC_GetMatchStringValue` functions are called using the arguments passed by the callback. Information for other hosts already connected to the server to which connection is being made cannot be retrieved.

- DWCCConnectAttemptCallback

Called by the server host that accepts the newly participating client host. At this time, the newly participating client host and all hosts that are already participating are passed to the `DWC_SetupGameServer` function and the `DWC_ConnectToXxx` function, respectively. `connectionUserData` can be retrieved. (Uses the `DWC_GetConnectionUserData` and `DWCCConnectAttemptCallback` arguments.)

- Narrowing targets occurs at different times.

- DWCEvalPlayerCallback

Called when a new client host is about to select an available server host. Connection to partners who are excluded by the narrowing conditions is not performed.

- DWCCConnectAttemptCallback

Called when the new client host tries to connect to the server host. Client hosts that are excluded by the narrowing conditions do not participate in the matchmaking of the server hosts that performed the narrowing.

The following is a simplified summary.

- DWCEvalPlayerCallback
 - Determine whether to allow participation using only the information of the server host to which to connect
 - Can be used only for matchmaking that is not server-client matchmaking
- DWCCConnectAttemptCallback
 - Determine dynamically whether to allow participation according to information of all participating hosts
 - Can be used for all matchmaking types

7.8 Server Migration

When the server host disconnects from a connection group, one of the remaining hosts continues in the server host role. By doing this, even if the person who first filled the role of server host drops out, new hosts can still participate in that group.

Server migration occurs only when the connection configuration (see section 8.2 Connection Configurations) uses either hybrid or full-mesh models. When a star model server host disconnects, all other client hosts terminate in an error.

In addition, when a server migration is generated, only the hosts that are already directly connected to the new server host continue. All other hosts are disconnected, even for the hybrid or full-mesh models.

Server migration occurs during all types of matchmaking.

Due to this server migration feature, there is a possibility that when matchmaking by specifying friends, the friend relationship between the server host after migration and the other client hosts may become more than “friend of a friend of a friend.”

For matchmaking by specifying friends, the following process is performed to ensure that all participants fit within the “friend of a friend” range.

1. For the first server host, new client hosts can participate if they are friends with the server host.
2. After server migration is performed one or more times, new client hosts cannot participate in that group. However, if the group includes only you, new client hosts can participate.

Server hosts that are in the second state have return value of `DWC_STATUS_PLAYING`, obtained with a `DWC_GetFriendStatus` function.

New client hosts are unable to connect after a server migration occurs during server-client matchmaking.

7.9 Increasing Matchmaking Speed

During peer matchmaking without specifying friends, you can increase the speed of matchmaking using filters when getting a list of matchmaking candidates from the matchmaking server (see Code 7-1). The matchmaking candidate list stored on the matching server has various conditions attached.

Matchmaking is more likely to fail when this list is obtained unconditionally and matchmaking candidates are filtered inside the evaluation callback. This also takes more time by repeatedly getting the list and performing matchmaking. You can reduce matchmaking failures and increase matchmaking speed using a filter function to form the obtained matchmaking candidate list into a list of acceptable matchmaking candidates.

Conversely, matchmaking efficiency can drop and time may be lost if excessive filtering is performed inside the evaluation callback in situations where the number of candidates is likely to be low (such as when seeking players of the same skill level or in the same geographical region).

Consider the following when seeking to increase the matchmaking speed.

- Use a filter function to form a list of available candidates from the obtained matchmaking candidate list
- Adopt a specification where matches are made aggressively without too much filtering inside the evaluation callback

7.10 Names That Cannot Be Used for Matchmaking Index Keys

There are certain key names that cannot be registered as matchmaking index keys by the `DWC_AddMatchKey*` function because the key names are used by the library and the server. Do not use any of the names listed in Table 7-1.

Table 7-1 Key Names That Cannot Be Used for Matchmaking Index Keys

country	region	hostname	gamename	gamever	hostport
mapname	gametype	gamevariant	numplayers	numteams	maxplayers
gamemode	teampplay	fraglimit	teamfraglimit	timeelapsed	timelimit
roundtime	roundelapsd	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 Exchange

Once matchmaking completes, the connections between participating hosts are established. Depending on the connection configuration settings that are set when matchmaking begins, the direct connections between hosts differ.

There are three formats: a format where an interconnection exists among all hosts if one host completes participation with matchmaking; a format where interconnections exist only between the server and each client; and an intermediate format (at first only server-client, then in the background becoming a complete interconnection).

There are several preparations that are required for direct communication with each host in this group created through matchmaking.

First, set up a receive buffer so each host can receive data. Call the `DWC_SetRecvBuffer` function. For the `aid` argument, specify the AID that serves as the ID number of each host.

The AID accepts values between 0 and N , where N is one less than the number of devices on the network. Therefore, if matchmaking for four players completes, the four are assigned AID numbers 0, 1, 2, and 3. If the device assigned AID = 1 leaves the network, the remaining devices maintain the assigned AID numbers 0, 2, and 3. Any data that arrives before setting up the receive buffer is deleted.

Next, configure the send and receive callbacks using the `DWC_SetUserSendCallback` and `DWC_SetUserRecvCallback` functions. Call the receive callback when a host receives data from another host. Call the send callback immediately after transmission of specified data completes.

In this context, note that *transmission completes* means that the data has been passed to the lower layer transmission function. It does not indicate that the partner device has received the data.

To configure the connection close callback, call the `DWC_SetConnectionClosedCallback` function when the local or partner host leaves the network by the procedure to officially disconnect (see Code 8-1).

Code 8-1 Setup for Data Exchange

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

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



```

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

    // Set the receive buffer for AIDs other than local AID
    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 close callback
DWC_SetConnectionClosedCallback( cb_closed, NULL );
}

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

// Callback for received data
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( DWCErr 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 );
        }
    }
}

```

There are two kinds of data transmission: *reliable transfer* and *unreliable transfer*. Both use UDP communication, but as with TCP communication, reliable transfer does not experience packet loss

and maintains packet order. However, the tradeoff is that reliable transfer takes longer to complete because every sent packet is checked upon receipt.

Because unreliable transfer uses UDP communication, problems with packet loss and packet order can occur. However, transmission is very fast because no packets are checked or resent.

If data transmission occurs at a layer lower than the DWC library, the data accumulates in the send buffer that has a size specified by the `DWC_InitFriendsMatch` function. If the send buffer does not have enough free space when *reliable transfers* are attempted, any unsent data are retained as is. They are sent from inside the `DWC_ProcessFriendsMatch` function as soon as space is freed in the send buffer.

Note that the default maximum amount of data that can be sent at once is 1465 bytes. If you try to send more than this maximum amount of data, the data is divided up and the send is suspended. You can change the maximum size of the send buffer using the `DWC_SetSendSplitMax` function. However, because communication devices with various settings need to be accommodated, do not set a maximum size larger than the default maximum.

Do not delete the send buffer if data for transmission is retained and suspended in this way. Also be aware that the next data set cannot be sent while data is retained and suspended.

Use the `DWC_IsSendableReliable` function to check if space is available in the send buffer, the send target AID is valid, and reliable transfer is possible (see Code 8-2).

If you attempt to send more than the maximum amount of data using unreliable transfer, the transmission will fail, and `FALSE` will be returned.

Code 8-2 Sending Data

```
static u8 s_SendBuffer[ SIZE_SEND_BUFFER ];

void send_data( void )
{
    // Send data using unreliable transfer to all connected hosts.
    // Ignore local AID if passed.
    DWC_SendUnreliableBitmap( DWC_GetAIDBitmap(),
                             s_SendBuffer, SIZE_SEND_BUFFER );
    :

    // Check whether reliable transfer is possible for host with AID=0
    if ( !DWC_IsSendableReliable( 0 ) ) return;

    // Send data using reliable transfer to a specific host
    DWC_SendReliableBitmap( 0, s_SendBuffer, SIZE_SEND_BUFFER );
    :
}
```

The DWC library sends debug output to the debugger on demand. Debug output sometimes takes a long time, so if it occurs at the same time that packets are sent or received, the transmission will appear to be delayed. To avoid this, use the `DWC_SetReportLevel` function to suppress debug output.

Alternatively, you can reduce the cost of processing log output by defining an independent `OS_TVPrintf` function, which is compiled as a weak symbol and called from debug print statements.

8.2 Connection Configurations

There are three connection configurations for the ways in which reliable and unreliable communication can occur after new participant matchmaking completes: a hybrid configuration, a star configuration, and a full-mesh configuration.

Unreliable communication can occur among all hosts. Reliable communication limits the hosts to which data can be sent based on the connection configuration.

The connection configuration can be set with an argument in a function that starts matchmaking:

`DWC_ConnectToAnybodysync`, `DWC_ConnectToFriendsAsync`, `DWC_SetupGameServer`, `DWC_ConnectToGameServerAsync`, and `DWC_ConnectToGameServerByGroupID`.

When starting matchmaking with these functions, set the desired connection configuration.

- Full-mesh configuration (`DWC_TOPOLOGY_TYPE_FULLMESH`)

At the time the matchmaking of the new participant host completes, an interconnection between all hosts exists, and reliable and unreliable communication can occur among all hosts.

- Star configuration (`DWC_TOPOLOGY_TYPE_STAR`)

An interconnection exists only between the server host and client host. Reliable communication can occur between the server host and client host, but reliable communication cannot occur between two client hosts.

Unreliable communication can occur among all hosts.

Unreliable communication between client hosts is automatically relayed by the server host.

- Hybrid configuration (`DWC_TOPOLOGY_TYPE_HYBRID`)

At the time the matchmaking of the new participant host completes, an interconnection exists only between the server host and newly connected client host. After matchmaking completes, an interconnection between client hosts is established in the background.

Reliable communication can occur with hosts for which there is a direct interconnection.

The library does not directly notify whether a connection between client hosts has been established.

It is necessary to check whether reliable communication is possible with another host using the method described below.

(Reliable communication can occur between server hosts and client hosts because they always have an interconnection.)

Unreliable communication can occur among hosts.

Unreliable communication among client hosts is automatically relayed by the server host when there is no direct connection.

A list of the AIDs of the hosts that can perform reliable communication can be retrieved with the `DWC_GetDirectConnectedAIDBitmap` function. `DWC_IsSendableReliable` returns `FALSE` for partner hosts for which there is no direct interconnection.

The full-mesh configuration takes the most time for matchmaking because it is necessary to establish interconnections between all the hosts when matchmaking completes. In addition, if there are more participants, the time until matchmaking completes also increases.

The star and hybrid configurations complete matchmaking more quickly than the full-mesh model because matchmaking completes when the connection between the new client host and server host is established.

8.3 Closing Connections

Call the `DWC_CloseAllConnectionsHard` function to close the connection with all hosts in the group. When the close process is executed, the connection-close callback set by the `DWC_SetConnectionClosedCallback` function is called before exiting this function.

The close notification also notifies other hosts that were connected and calls the connection-close callback.

This `DWC_CloseAllConnectionsHard` function is called even if there are no other connected hosts at the time. This function call deallocates any remaining regions of memory that were used for matchmaking and restores the communication state to the online state. Calling this function does not close the connection with the Nintendo Wi-Fi Connection server.

8.4 Estimated Buffer Sizes to Specify with `DWC_InitFriendsMatch`

The DWC library uses the buffer sizes specified by the `DWC_InitFriendsMatch` function. When data is sent using reliable communication, the Send buffer stores data for which ACK is not returned. The Receive buffer stores data that did not reach the Receive buffer in the correct order.

With reliable communication, you need as much capacity as possible to deal with instantaneous network interruptions. The Send and Receive buffers both need to be large enough to handle as much interruption time as the game's specifications permit.

Although the Send and Receive buffers are generally not used with unreliable communication, you still need a Send buffer of at least 1 KB and a Receive buffer of at least 128 bytes because DWC uses reliable communication internally when connecting peer-to-peer.

Table 8-1 Estimated Buffer Sizes

Kind of Communication		Estimated Buffer Sizes	Comments
Reliable Communication	Send buffer size	Compute buffer size as (allowable duration in seconds of instantaneous interruption as per the game specs) x (amount of reliable data per second) x (total size of reliable data).	Minimum of 1 KB
	Receive buffer size	Total size of reliable data = 7 x (number of divisions in the data being sent) + (size of data being sent) + 15	Minimum of 128 bytes
Unreliable Communication	Send buffer size	(Max. data size for unreliable communication) + 2 bytes	Minimum of 1 KB
	Receive buffer size	Minimum of 128 bytes	

Note: The number of divisions in the data sent indicates the number into which the data is divided when the total data size exceeds the maximum amount of data that can be sent at any one time. This is specified by the `DWC_SetSendSplitMax` function (default size: 1465 bytes).

The following example shows how to calculate the required size of the Send and Receive buffers. Assume that:

- The game spec allows an instantaneous interruption to last for as long as 1 second
- Communication is performed once every 3 frames
- The maximum amount of data that can be sent at one time is 64 bytes
- The game is sending 100 bytes of data using reliable communication

In this case, the required size of the Send and Receive buffers is:

$$1 \text{ (second)} \times (60 \text{ (frames)} \div 3) \times (7 \times 2 \text{ (divisions)} + 100 \text{ (bytes)} + 15) = 2580 \text{ (bytes)}$$

8.5 Emulating Delays and Packet Loss

The DWC library can emulate delays and packet loss for sending and receiving data. For send delays, the send data is copied to another buffer and kept for a specified amount of time. This data will not be sent to the partner because the data is deleted when the connection is closed. For this reason, using only the receive delay is recommended.

The packet loss rate (in units of percent), the delay time (in units of milliseconds), and the AID of the receiving Nintendo DS system are specified in Code 8-3.

Code 8-3 Emulating Delays and Packet Loss

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

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

8.6 Amount of Data Sent and Received

Table 8-2 shows the amount of data transmitted during reliable and unreliable communication.

Table 8-2 Communication Data Breakdown

Send Data Items	Send 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 send	Data send	Receive check	Data send
	15 bytes	7 + XXX bytes	5 bytes	XXX bytes
FCS	4 bytes			
B (random time for avoiding packet collision)	MAX 600 μ s (microseconds)			

Note: The header send and receive check are sent before and after the “data send” event during reliable communication.

Although you can find the data send time for each transmission based on the formula $\text{Preamble} + (\text{MAC} + \text{LLC} + \text{IP} + \text{UDP} + \text{DATA} + \text{FCS}) \times 4 + \text{B} [\mu\text{s}]$, it is difficult to accurately calculate the amount of data sent and received. This is due to the fact that the transmission time varies depending on factors such as the number of retries required due to bandwidth conditions, the number of sent packets, and the amount of transmission standby to avoid packet collisions.

This section provides figures obtained in experiments for the amount of data sent/received.

Experiments were conducted by measuring throughput, CPU load, and the packet loss ratio while varying conditions such as the use of reliable or unreliable communication, the AP model and manufacturer, the amount of radio usage, the send size, and the send frequency. As a result, the following became clear.

- Send frequency (the number of packets issued) is greatly affected by the presence of back-off time (including empty intervals between communication and random time for avoiding packet collisions) of the header part and wireless communication
- In a four-unit mesh network where data is sent at a rate of once every three frames and the radio noise is under 10%, the upper limit of send size is in the range of 120-150 bytes

- In a four-unit mesh network where data is sent at a rate of once every three frames and the radio noise is around 50%, the upper limit of send size is in the range of 100-120 bytes
- When using reliable communication, traffic congestion occurs easily because congestion is exacerbated by the need to repeatedly resend data when the network is busy. Once this occurs, recovery time is extended.

Note: Radio noise is generated by using `WMTTestTool` from another Nintendo DS system.

Based on the experimental results above, Nintendo titles communicate as listed below.

- Four-unit mesh network, unreliable communication

Nth frame: Send to Party 1

(N+1)th frame: Do not send

(N+2)th frame: Send to Party 2

(N+3)th frame: Do not send

(N+4)th frame: Send to Party 3

(N+5)th frame: Do not send

(Repeats from this point on)

Communication every 60 to 104 bytes

- Four-unit server-client type connection, reliable communication

Send frequency is three frames with a usual send size of 1 to 40 bytes (up to 256 bytes).

When you are developing a game, be sure to consider the following design considerations.

- Network Environment

Expect Internet-based transmission delays and packet losses. Transmission delays are generally constant for domestic connections, but tend to vary widely for international communications.

- Wireless Environment Congestion

Compared to wired environments, a large number of packets often cause congestion.

When you are fine-tuning a communication environment, adjust both the size and number of packets to send or receive at a particular time. To guarantee game content when restricting communication partners is unavoidable due to line quality, see the *Nintendo Wi-Fi Connection Programming Guidelines*.

9 Communication Errors

The DWC library provides an error handling system for all DWC modules. In this system, DWC errors are treated like application errors.

9.1 Error Handling

You can get the error status in the DWC library using `DWC_GetLastErrorEx` function, as shown in Code 9-1. The error classification is the return value. The arguments are the error code and the pointer to the storage location for the error handling type.

The error code is 0 or a negative number. If you are going to show the error code, be sure to invert the sign so that the value is shown as a positive number. However, if it is a recoverable error and the Nintendo DS system was not disconnected from Nintendo Wi-Fi Connection, you do not need to display the error code.

The error process type indicates the recovery process required after the error occurs, and a routine error process can be created for each value.

Once the error state has been entered, the DWC library rejects most functions. To return from the error state, call the `DWC_ClearError` function.

Code 9-1 Error Handling Process

```
void main loop( void )
{
    while ( 1 )
    {
        DWC_ProcessFriendsMatch();

        handle_error(); // Error-handling process

        GameWaitVBlankIntr();
    }
    :
}

// Error-handling process
void handle_error( void )
{
    int dwcError, gameError;

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

int handle_dwc_error( void )
{
    int errcode;
    DWCErrors err;
    DWCErrorsType errtype;
```



```

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

// If there is no error, return without doing anything
if ( err == DWC ERROR NONE ) return 0;

// Clear the error
DWC ClearError();

// Display an error message
disp_error_message( err );
// If error code is -10000 or lower, display the code as a positive number
if ( errcode <= -10000 ) disp_message( "%d", -1*errcode );

if ( errtype == DWC_ETYPE_SHUTDOWN_FM )
{
    // End the FriendsMatch process
    DWC_ShutdownFriendsMatch();
}
else if ( errtype == DWC_ETYPE_DISCONNECT )
{
    /* End the FriendsMatch process and perform cleanup on Internet connection */
    DWC_ShutdownFriendsMatch();
    disconnect_func();
}
else if ( errtype == DWC_ETYPE_FATAL )
{
    // Fatal Error, so nothing can be done after prompting to turn power off
    while (1) ;
}
/* If only a minor error, you can just clear the error and resume the FriendsMatch process */
return err;
}

```

9.2 List of Error Codes

This list provides the main error codes that occur during the matchmaking and friend relationship process.

If the last three digits of an error code are 010 or 020, these errors are likely to occur if the GameSpy server is in an unstable state (for example, during maintenance).

- 61010 A communication error occurred with the GameSpy GP server during GP server login.
- 61020 A communication error occurred with the GameSpy GP server during GP server login.
- 61070 A login timeout error occurred during GP server login.
- 71010 A communication error occurred with the GameSpy GP server while synchronizing friend rosters.
- 80430 Connection to the client DS failed for server-client matchmaking because the server DS that the client DS was attempting to connect with or the client DS connected to the server DS was powered off.
- 81010 A communication error occurred with the GameSpy GP server during matchmaking.
- 81020 A communication error occurred with the GameSpy master server during matchmaking.

- 84020 Communication from the GameSpy master server was interrupted during matchmaking. Either the master server is down or the firewall is blocking UDP.
- 85020 A communication error occurred with the GameSpy master server during matchmaking.
- 85030 The GameSpy master server DNS failed during matchmaking. All error codes with 030 as the last three digits indicate DNS errors.
- 86420 NAT negotiations failed the set number of times during one matchmaking session. There may be a problem with the router. In server-client matchmaking, this error only occurs when the client DS that has started connecting and NAT negotiation has failed one time.
- 97003 A socket error has occurred in a lower layer than the DWC library after matchmaking completes.

Error codes with 1010 or 1020 as the last four digits and error code 85020 are known to occur frequently in the NITRO Wi-Fi library for NitroWiFi, version 1.0 RC2 and earlier, when TCP transfers with the GameSpy server are delayed.

10 Network Storage Support

The DWC library can store data onto the network storage server provided by GameSpy. Code 10-1 shows you how to use this feature.

To access this storage server, complete the process up to the login using the `DWC_LoginAsync` function. Next, log in to the storage server using the `DWC_LoginToStorageServerAsync` function.

The data to save on the storage server can have public or private attributes. If the data is saved using the `DWC_SavePublicDataAsync` function, the data attributes are public and other players can reference the data.

If the data is saved using the `DWC_SavePrivateDataAsync` function, the data attributes are private and other players cannot reference the data.

To load data from the storage server, call the `DWC_LoadOwnPublicDataAsync` function to load your own public data, `DWC_LoadOwnPrivateDataAsync` function to load your own private data, and the `DWC_LoadOthersDataAsync` function to load the friend data saved in your friend roster. Friends are specified by the friend roster index.

When saving or loading data completes, the appropriate callback set by the `DWC_SetStorageServerCallback` function is called. These callbacks are always called in the order that the save and load functions were called.

A string that combines key and value can be specified as saved data. The key/value combinations are repeated by delimiting with `\\`, as in `\\name\\mario\\stage\\3`. If this example data is specified, `mario` is registered in the key value name, and `3` is registered in the key value stage as a string.

To load data saved on the storage server, specify the keys that you want to retrieve as `\\name\\stage`, separating the name and stage with `\\`.

In this case, the string that you can get with a load callback would be in the format of `\\name\\mario\\stage\\3`.

If you attempt to load a key that does not exist on the storage server or a key that was saved by a friend who used the private attribute, the `success` argument of the callback function will be `FALSE`. If you specify multiple keys to load and only some of the keys fall into these two categories, the `success` argument will be `TRUE`, but these keys will not be included with the loaded data.

After storage server processing completes, call the `DWC_LogoutFromStorageServer` function to log out from the storage server (as in Code 10-1).

Code 10-1 Accessing the Storage Server

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

void access_net_storage( void )
{
    // Login to the storage server
    if ( !DWC_LoginToStorageServerAsync( cb_storage_login, NULL ) )
```

```
{
    OS_TPrintf( "DWC LoginToStorageServerAsync() failed.\n" );
    return;
}

// Wait for login to storage server to complete
while ( !s_storage_logged )
{
    DWC_ProcessFriendsMatch();

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

    GameWaitVBlankIntr();
}

// Set callbacks for the time when saving and loading complete
DWC_SetStorageServerCallback( cb_save_storage, cb_load_storage );

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

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

// Wait for saving to complete
while ( s_cb_level > 0 )
{
    DWC_ProcessFriendsMatch();

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

    GameWaitVBlankIntr();
}

// Load local saved data
s_cb_level++;
if ( !DWC_LoadOwnDataAsync( "\\id\\stage", NULL ) )
{
```

```
    OS_TPrintf( "DWC_LoadOwnDataAsync() failed.\n" );
    return;
}
// Load one's own private save data
s_cb_level++;
if ( !DWC_LoadOwnPrivateDataAsync( "\\id", NULL ) )
{
    OS_TPrintf( "DWC_LoadOwnPrivateDataAsync() failed.\n" );
    return;
}
// Load another player's saved data
s_cb_level++;
if ( !DWC_LoadOthersDataAsync( "\\name", 0, NULL ) )
{
    OS_TPrintf( "DWC_LoadOthersDataAsync() failed.\n" );
    return;
}

// Wait for loading to complete
while ( s_cb_level > 0 )
{
    DWC_ProcessFriendsMatch();

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

    GameWaitVBlankIntr();
}

// Log out from storage server
DWC_LogoutFromStorageServer();
:
}

// Callback for the tune when logged in to storage server
void cb_storage_login( DWCErr error, void* param )
{
    if ( error == DWC_ERROR_NONE )
    {
        s_storage_logged = TRUE;
        s_cb_level       = 0;
    }
}

// Callback for when data is saved to storage server
void cb_save_storage( BOOL success, BOOL isPublic, void* param )
{
    OS_TPrintf( "result %d, isPublic %d.\n", success, isPublic );
    s_cb_level--;
}
```

```
// Callback for the time when data loaded from storage server
void cb_load_storage( BOOL success, int index, char* data, int len, void* param )
{
    OS_TPrintf( "result %d, index %d, data '%s', len %d\n",
                success, index, data, len );
    s_cb_level--;
}
```

11 Differences Between Versions Before NITRO DWC 3.X

This chapter describes and notes the differences between TWL DWC 5.0 and later, and NITRO DWC 3.X and earlier.

11.1 Matchmaking

- Changed the callback for matchmaking by specifying friends and without specifying friends to `DWCMatchedSCCallback` and `DWCNewClientCallback`, the same as for server-client matchmaking.

Matchmaking by specifying friends and without specifying friends did not call the matchmaking completion callback until the specified number had gathered for DWC 3.X and earlier. However, this version of the DWC calls the matchmaking completion callback each time a person participates, the same as with server-client matchmaking.

- By using the reconnection feature that uses the group ID, you can participate in the same group after dropping out.
- The `distantFriend` argument for matchmaking by specifying friends was deleted. The behavior is the same as in DWC 3.X for `distantFriend=TRUE` while accepting participants.
- Added the connection configuration setting argument to the matchmaking start functions.
 - `DWC_ConnectToAnybodyAsync`
 - `DWC_ConnectToFriendsAsync`
 - `DWC_SetupGameServer`
 - `DWC_ConnectToGameServerAsync`
 - `DWC_ConnectToGameServerByGroupID`

When setting to either the star or hybrid configuration, matchmaking can complete more quickly than DWC 3.X or earlier when there are a large number of people.

There are cases when mutual reliable communication does not exist, depending on the connection configuration settings.

- Added `DWCConnectAttemptCallback` to all matchmaking types to determine accepting matchmaking.
- Deleted functions related to matchmaking options.
 - `DWC_SetMatchOption`
 - `DWC_GetMatchOption`
 - `DWC_GetMOMinCompState`
 - `DWC_GetMOSCCConnectBlockState`
 - `DWC_ClearMOSCCConnectBlock`

- `DWC_StopSCMatchAsync`

For the previous function, consider substitute specifications for the application with the `RequestSuspendMatchAsync` function to close participation. The following is an example of a method to substitute with the `RequestSuspendMatchAsync` function to close participation.

- `DWC_MATCH_OPTION_MIN_COMPLETE`

Matchmaking completion notification is made with a callback every time a person connects. When a specified time has passed without reaching the maximum number of participants, participation is closed with the `RequestSuspendMatchAsync` function, and matchmaking can be completed with less than the maximum number of participants.

- `DWC_MATCH_OPTION_SC_CONNECT_BLOCK`

The feature to close participation with the `RequestSuspendMatchAsync` function does not perform the exact same operation as `DWC_MATCH_OPTION_SC_CONNECT_BLOCK`.

- `DWC_StopSCMatchAsync`

This can be replaced with the feature to close participation with the `DWC_RequestSuspendMatchAsync` function.

- Added `(void* param)` to `DWCUserSendCallback`, `DWCUserRecvCallback`, `DWCUserRecvTimeoutCallback`, and `DWCUserPingCallback` so that parameters can be passed from the application.
- Hosts are now automatically disconnected if communication is absent for a fixed period of time. This is 20 seconds by default but can be set by the `DWC_SetConnectionKeepAliveTime` function. Applications must send some type of reliable or unreliable communication to all connected hosts within the timeout interval set by this function. This is because there is no automatic communication within the DWC library to maintain a connection.

Microsoft, Windows, Internet Explorer, and Visual Studio are registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.

Metrowerks and CodeWarrior are registered trademarks or trademarks of Metrowerks Inc. in the United States and other countries.

Avid, Softimage, SOFTIMAGE|3D and SOFTIMAGE|XSI are registered trademarks or trademarks of Avid Technology Inc.

Maya, Discreet, and 3ds Max are registered trademarks or trademarks of Autodesk Inc./Autodesk Canada Inc. in the United States and other countries.

Adobe, Photoshop, Acrobat, and Acrobat Reader are registered trademarks or trademarks of Adobe Systems Incorporated.

OPTiX web technology and iMageStudio are registered trademarks or trademarks of Web Technology Corp.

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

© 2005-2010 Nintendo

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