Chance-Encounter Communications Library

Version 1.2.0

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

TWL-06-0163-001-B
Released: September 16, 2009

Contents

Code

Figures

Revision History

Version	Revision Date	Description
1.2.0	2009/07/29	Added Chapter 3 Shared Chance-Encounter Relay Stations.
1.1.1	2009/05/14	Made changes due to this library's inclusion in the TWL-SDK.
1.1.0	2006/08/10	Changed a function name (`WXC_RegisterData` to `WXC_RegisterCommonData`).
1.0.0	2005/09/22	Initial version.

1 Introduction

This document describes the operational overview of “chance-encounter communications” in the Nintendo DS environment and the details of its associated libraries. The chance-encounter communications library is referred to as the Wireless Xing (Crossing) Communications library, or the WXC library for short.

1.1 Overview

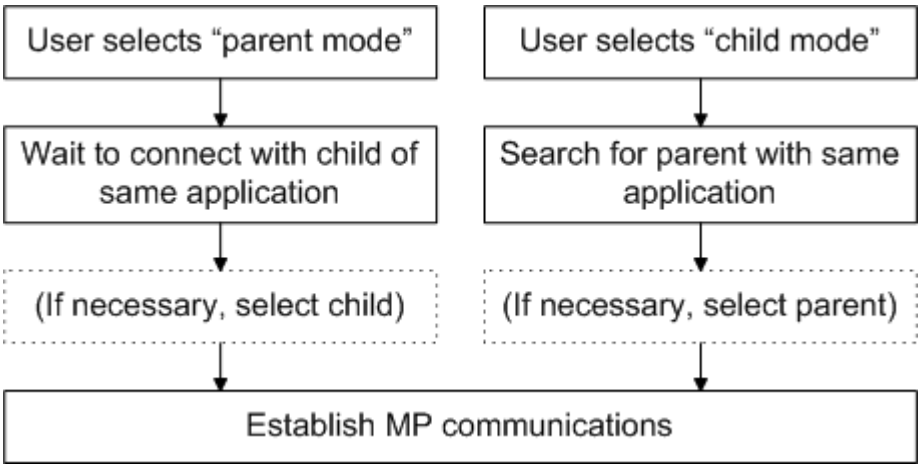
In this document and in the WXC library, the term "chance-encounter communications" refers to a set of functions that automatically execute the following procedures based user application settings.

- Searching for the same kind of application and establishing communications
- Directly exchanging data using block transfers

The WXC library automatically manages these internal procedures, so you only need to be concerned with advanced configuration and the processes that occur after these internal procedures. You do not need to be concerned with wireless controls.

For most Nintendo DS wireless games, each system chooses to act as a parent or child before communicating. Also, the child must select a parent from the list of parents found. Selection and decision are usually manual tasks performed from the user interface. Figure 1-1 shows the connection sequence.

Figure 1-1 Connection Sequence for a Typical Nintendo DS Wireless Play Game

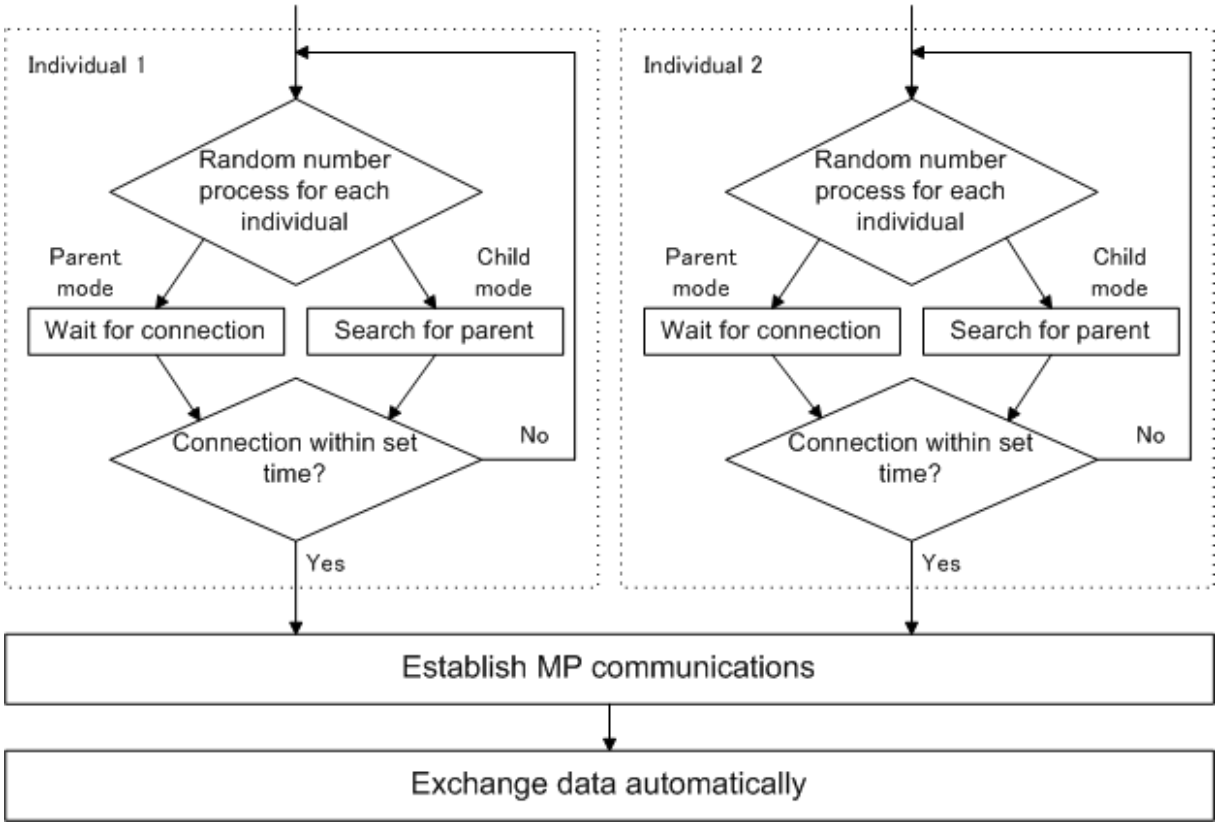


In chance-encounter communications, the library executes the previously described selection process. By randomly alternating the parent and child states, the roles of parent and child get assigned automatically to the individual Nintendo DS systems to establish wireless communications as a pair.

Because chance-encounter communications are based on the notion that a connection can be established with any partner that has the same application, the library also chooses the connection target on the child side by determining the GGID in the reception beacon.

Figure 1-2 shows the connection sequence.

Figure 1-2 Connection Sequence for Chance-Encounter Communications



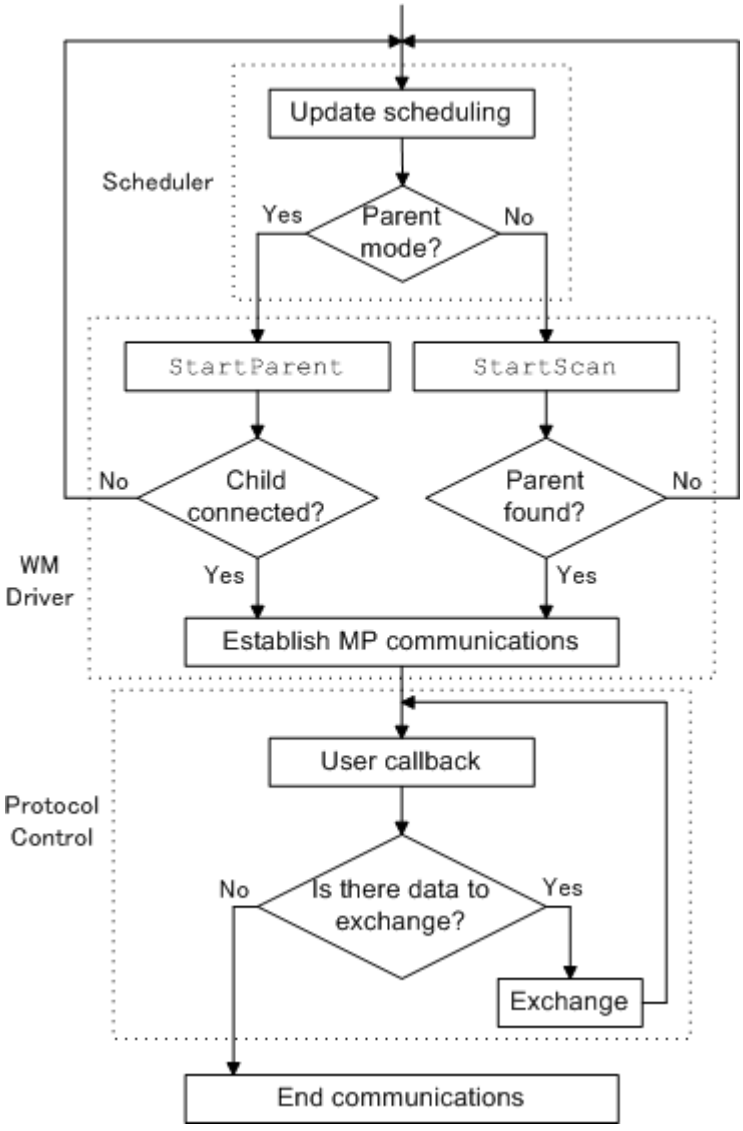
2 WXC Library Operations

This chapter describes the structure of the WXC library, its operations, and how to use it.

2.1 Overall Structure

Figure 2-1 shows the operational flow of the WXC library. The internal processes are broadly divided into three sub-modules: Scheduler, the WM Driver, and Protocol Control.

Figure 2-1 Overall Operation of the WXC Library



2.1.1 WM Driver

The WXC library makes internal use of the WM library.

Calling the `WXC_Start` function starts the activation of WM and automatically shifts the wireless state to either IDLE, MP_PARENT, or MP_CHILD, according to the evaluation by the Scheduler.

Calling the `WXC_End` function ends the activation of WM.

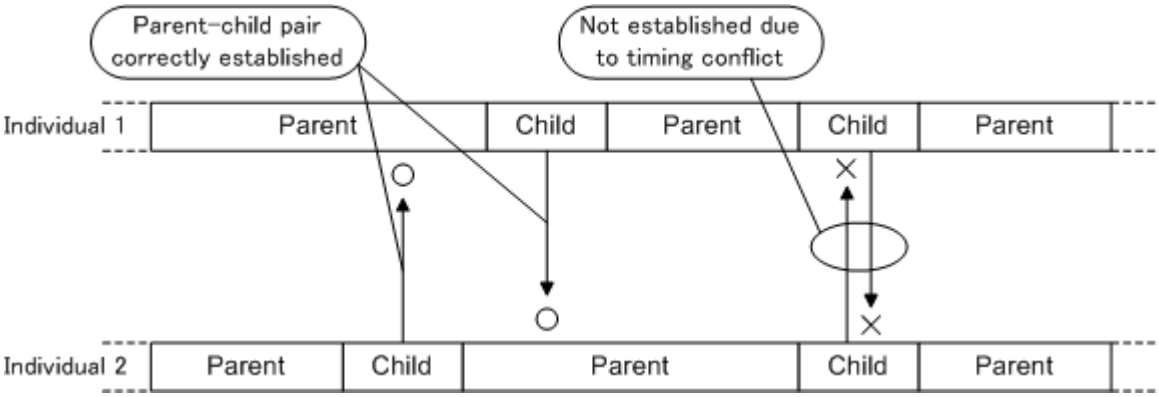
2.1.2 Scheduler

The WXC library repeats the parent and child state transitions, but the parent-child pair may not be established if there are search timing conflicts or if the state transition interval are synchronized or closely matched.

To reduce the chance of this happening, the Scheduler sets a random period for each system.

Figure 2-2 shows the state transition of parent and child performed by the scheduler and the success/failure of the communication between different systems.

Figure 2-2 Parent-Child State Changes and Timing of Establishment of Communication



2.1.3 Protocol Control

MP communications of the WXC library use the sequence of executing the data transfer in both directions at once to exchange data.

Data transfer uses block transfer as the method for handling the received bit set employed by the MB library and the WBT library. The data exchange is deemed successful when both sides confirm completion. The communication process is treated as a symmetric process and there is no distinction between parent and child.

For this sequence, it is a precondition that both the parent and the child know the size of data being sent and received; a sufficient receive buffer for storing the maximum possible data to receive from the other party must be configured in the application. If either parent or child exceeds the receive size, the exchange will not occur, and both the parent and child processes will fail.

2.2 Library Operating Procedures

This section describes how to use the WXC library.

2.2.1 Initializing the Library

Call the `WXC_Init` function first and initialize the WXC library.

The WXC library needs a work memory region of `WXC_WORK_SIZE` bytes and one DMA channel.

This is also where the system callback is specified for notifying changes in internal state and the occurrence of various events.

Code 2-1 Initializing the Library

```
/* Initialize the library internal state */
WXC_Init(OS_Alloc(WXC_WORK_SIZE), system_callback, 2);
```

Note: This function initializes only the internal work memory and the various sub-modules. It does not start chance encounter communications.

Next, call the `WXC_RegisterCommonData` function to associate the GGID with the send/receive buffers and register them in the library.

Code 2-2 Registering the GGID and Data Buffers

```
/* Register data block. Prepare Receive buffer of sufficient size. */
   WXC_RegisterCommonData(APP_GGID, user_callback,
send_buffer, sizeof(send_buffer),
recv_buffer, sizeof(recv_buffer));
```

This sets the library to search for and exchange data with chance-encounter communications applications that have the same GGID.

Note: The user callback registered here is different from the system callback.

2.2.2 Starting

After the GGID and data are registered, call the `WXC_Start` function to begin actual chance-encounter communications.

From this point forward, all notifications about progress are notified of system callbacks and user callbacks. There is no need to be aware of the internal state of the library with the application.

Code 2-3 Starting the Library

```
/* Start wireless of the library */
WXC_Start();
```

2.2.3 Controlling Callbacks

There are two kinds of callbacks: system callbacks specified by the `WXC_Init` function and user callbacks specified by the `WXC_RegisterCommonData` function.

System callbacks are notified about state changes that affect the overall WXC library. The content of a notification can be ignored if is not applicable to the application.

Code 2-4 Example Implementation of a System Callback

```
static void system_callback(WXCStateCode state, void *arg)
{
  switch (state)
  {
  /* The following are notified from inside calls to state-change APIs and are not
specifically used */
  case WXC_STATE_READY:  /* Generated from inside WXC_Init function call */
  case WXC_STATE_ACTIVE: /* Generated from inside WXC_Start function call */
  case WXC_STATE_ENDING: /* Generated from inside WXC_End function call */
          break;
  case WXC_STATE_END:   /* Generated upon completion of WXC_End function's end
process */
          /*
                 * arg is the internal work memory region allocated by
                 * the WXC_Init function. The work memory is released
                 * to the user at this point, so it has been dynamically allocated
                 * it can be destroyed here.
                 */
          {
                  void *system_work = arg;
                  OS_Free(system_work);
          }
          break;
  case WXC_STATE_CONNECTED: /* Generated if new connection has occurred */
/*
* arg is the pointer to the WXCUserInfo structure that
* represents the connection target. Describe the process here
* only when there is some special control to do.
*/
          {
                  const WXCUserInfo * user = (const WXCUserInfo *)arg;
                  OS_TPrintf("connected(%2d):" user->aid);
          }
          break;
  }
}
```

User callbacks are notified when data exchange completes.

The receive buffer specified in the application is passed to the argument as the state in which the data is already stored.

Code 2-5 Example Implementation of a User Callback

```
static void user_callback(WXCStateCode stat, void *arg)
{
  /* At the present time, stat is always WXC_STATE_EXCHANGE_DONE */
#pragma unused(stat)
  /* arg is the pointer to the WXCBlockDataFormat structure where received data has
been stored */
   const WXCBlockDataFormat * block = (const WXCBlockDataFormat *)arg;
   u8     *recv_data = (u8 *)block->buffer;
   u32    recv_length = block->length;
  /*
   * Here is where processes unique to the application are performed.
   * It is also possible to call the WXC_End function here.
   */
}
```

2.2.4 Ending

If it is not necessary to continue with chance-encounter communications, call the `WXC_End` function to instruct the library to end.

When the end process completes, the internal state of the WXC changes to WXC_STATE_END, and notification is given to a system callback.

Code 2-6 Ending the Library

```
/* End wireless of the library */
WXC_End();
/*
* The end process is not completed immediately. The function
* waits for notification of completion with a system callback
* or periodically checks the WXC's internal state.
*/
while (WXC_GetStateCode() != WXC_STATE_END)
{
   OS_Sleep(100);
}
```

3 Shared Chance-Encounter Relay Stations

In some physical locations, shared chance-encounter relay stations (hereafter, chance-encounter relay stations) have been established to increase the opportunities for game titles that use the WXC library to exchange data with other copies of the same game title. A few cautions must be considered to prevent malfunction when a game title using the WXC library exchanges data with a chance-encounter relay station.

This chapter is a general description of the operating principles and cautions for chance-encounter relay stations.

3.1 Basic Operations of Chance-Encounter Relay Stations

A chance-encounter relay station automatically exchanges data when a game title using the WXC library passes nearby and temporarily saves that data, and then uses it for data exchange when another copy of the same game title next passes nearby. By repeating this operation, it acts like a relay station for chance-encounter data.

When you consider this relay station behavior with regards to your game title, be aware of the following two points.

- Because of the nature of the relay feature, chance-encounter communications via relay stations does not amount to a pure 1:1 exchange of data.

  In normal chance-encounter communications, 1:1 data exchanges occur between copies of the same game title, but 1:1 data exchanges do *not* occur in chance-encounter communications via relay stations. Therefore, chance-encounter communications via relay stations cannot be used for exchange of information such as Wi-Fi friend codes that have meaning only if exchanged between a specific pair of systems.

- Chance-encounter communications via relay stations may sometimes duplicate or lose chance-encounter data.

  Chance-encounter data is sometimes duplicated (see section 3.4 Support for Invalid Data (Zero Data)) or lost. (Loss is caused by the relay station internal specification stating that this data is saved once per hour, as well as by other factors.) For these reasons, you must not only confirm that the duplication or loss of chance-encounter data does not disrupt the atmosphere or setting of the game, but also ensure that neither the existence of two sets of exchanged chance-encounter data in the game nor the loss of exchanged chance-encounter data causes a malfunction.

3.2 Support for the WXC Library

You must use the WXC library to support communication with relay stations.

When you develop titles that support relay stations, be aware of the following four precautions required when using the WXC library.

- You must always use the Nintendo-supplied WXC library.
- You must always use the new `WXC_RegisterCommonData` function. Do not use the obsolete `WXC_RegisterData` function.
- You must confirm operations by checking that the relay station test program included with the SDK (`$TwlSDK/bin/ARM9-TS/Rom/RelayStation.srl`) responds correctly.
- [For North American titles only] North American DS Download Stations also function as relay stations for WXC chance-encounter communications. Therefore, for North American titles, you must additionally confirm that your title operates properly with the DS Download Station test program included with the SDK (`$TwlSDK/bin/ARM9-TS/Rom/DSDownloadStation.srl`).

You must observe those points in implementation and, in the design stage, also consider the points in the following sections.

3.3 Capacity Restrictions on Chance-Encounter Data

The maximum length of chance-encounter data is 32 KB. The WXC library restricts chance-encounter data of any larger size. If data exceeds the maximum size, the library is unable to receive it. This applies not only to chance-encounter communications with the relay station, but also to normal chance-encounter communications between copies of a game.

3.4 Support for Invalid Data (Zero Data)

Invalid data (also called zero data) is data sent from the relay station the first time the relay station has a chance encounter with a particular game title. Because the relay station does not yet have data on hand to exchange with copies of that game title, it exchanges chance-encounter data with a size of 0 bytes in the first encounter only. From the game title's viewpoint, this chance-encounter data is invalid, so you must anticipate the state of the game title when it receives this invalid data and incorporate code that allows it to handle this situation.

The following are some specific examples of ways in which the title can support this zero data.

- When the game title receives invalid data via chance-encounter communications, it reads other data from the game ROM and makes it appear that this data was received (in other words, this method makes it seem to the user that normal chance-encounter communications has occurred with another player).
- The game title exchanges its chance-encounter send data (the chance-encounter data configured to be sent to other parties when they are encountered, hereafter called the send data) for the invalid data from the relay station, but mention of this exchange is never displayed in the game to make it seem to the user that no exchange occurred. When this method is used, the same send data is re-set on the game side after the exchange finishes (in other words, this method makes it seem to the user that no chance encounter has occurred).

However, this method will leave send data in the game that is identical to that saved in the relay station. A player could use a procedure like the following to exploit this fact and make multiple identical copies of an in-game item.

1. The player sets Item A as their send data and exchanges it via chance-encounter communications for invalid data from the relay station. The game still has Item A set as its send data, so it appears to the player that no chance-encounter communications has occurred. However, the relay station receives Item A, so the same Item A now exists in both places.
2. The player removes Item A from their send data and instead sets Item B.
3. The player exchanges Item B for Item A via chance-encounter communications with the relay station. Now two instances of Item A exist in the game, because the game has received one from the relay station (sent in Step 1) and still has the one removed from the send data by the player in Step 2.

The phenomenon of this duplication can only occur when a specific game title first encounters and is registered with a specific relay station, so the probability of this actually occurring is thought to be low. If the possibility of item duplication is an issue even after its low probability is taken into account, there are various ways to prevent duplication of items. One way is to embed a unique identifier number in the game's chance-encounter data and thereby prevent the local copy of a game from receiving its own chance-encounter data back again.

3.5 Support for Repeated Chance-Encounter Communications

"Repeated chance-encounter communications" indicates a situation where the same player has two or more consecutive chance encounters with a relay station. When this happens, the player receives the same send data that they had set previously.

For example, some game titles implement a type of chance-encounter communications that allows the reception of items and messages, but does not allow the player to immediately set that unchanged received data as the send data for the next chance encounter. Specifications for this type of chance-encounter communications state that it is not possible for a copy of one of these game titles to receive its own data when it exchanges data with another copy of the game.

However, because the specifications for relay stations state that stations take the chance-encounter data they receive and send it out again, the relay stations make it possible for that "impossible" situation to occur. All developers creating titles with support for relay stations must be aware of this issue and design the title in such a way that it will not malfunction if the player ever receives back the same data that they set as send data.

One possible approach to ensure that the player will not receive their own send data is to embed a unique identifier in the chance-encounter data, as described in section 3.4 Support for Invalid Data (Zero Data), and use this identifier to evaluate whether a given piece of data actually originated in the local copy of the game. We recommend considering this approach while fulfilling the specifications for chance-encounter communications in your game.