Chance Encounter Communication Library

Version 1.1.1

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

TWL-06-0163-001-A
Released: August 7, 2009

Contents

Code

Figures

Revision History

Version	Revision Date	Description
1.1.1	2009/05/14	Made changes due to this library's inclusion in 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 communication library is referred to as the Wireless Xing (Crossing) Communication library, or the WXC library for short.

1.1 Overview

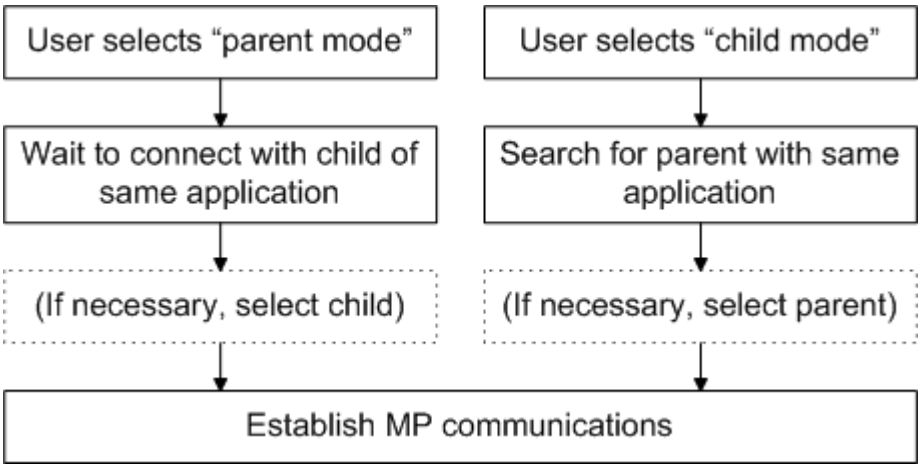
In this document and in the WXC library, the term "chance encounter communications" refers to a set of functions that automatically execute the following procedures based user application settings.

- 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 DS Wireless Play Game

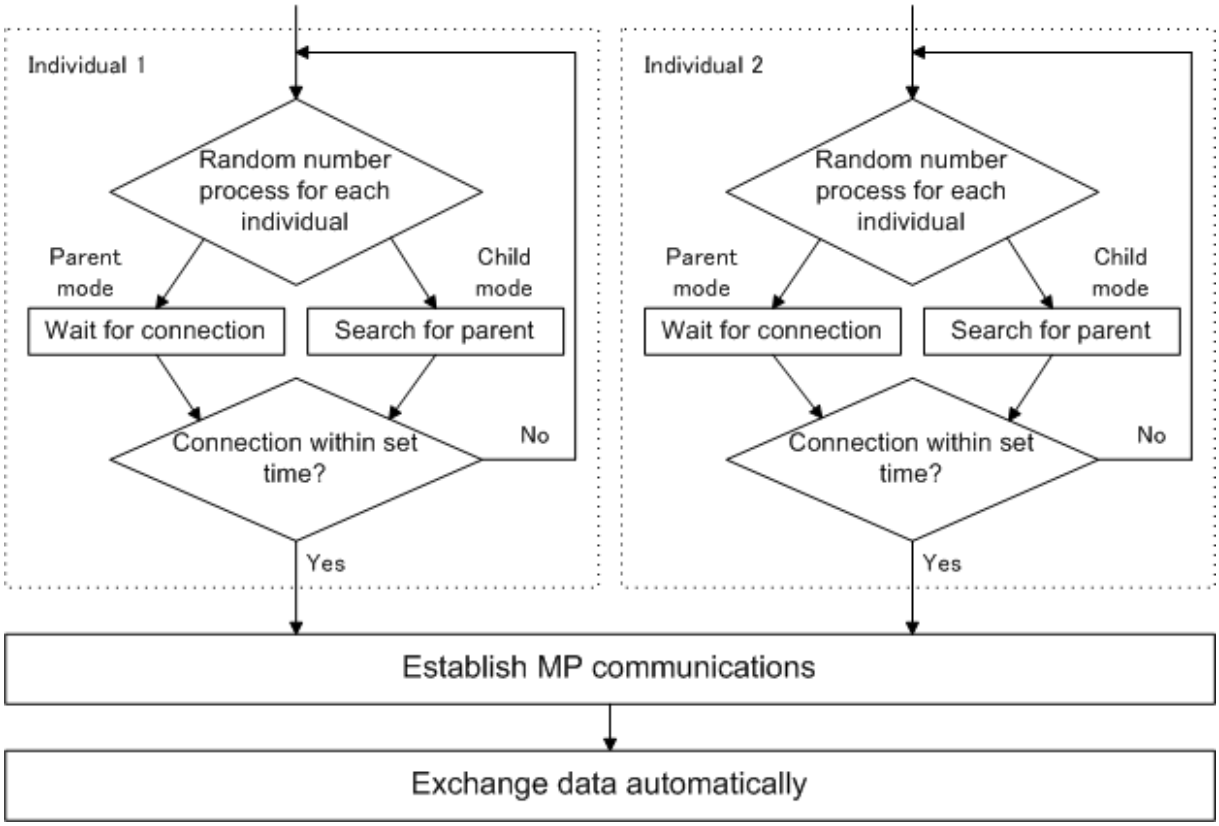


In chance encounter communications, the library executes the previously described selection process. By randomly alternating the parent and child states, the roles of parent and child get assigned automatically to the individual Nintendo DS systems to establish wireless communications as a pair.

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.

The connection sequence is schematized in Figure 1-2.

Figure 1-2 Connection Sequence for Chance Encounter Communication



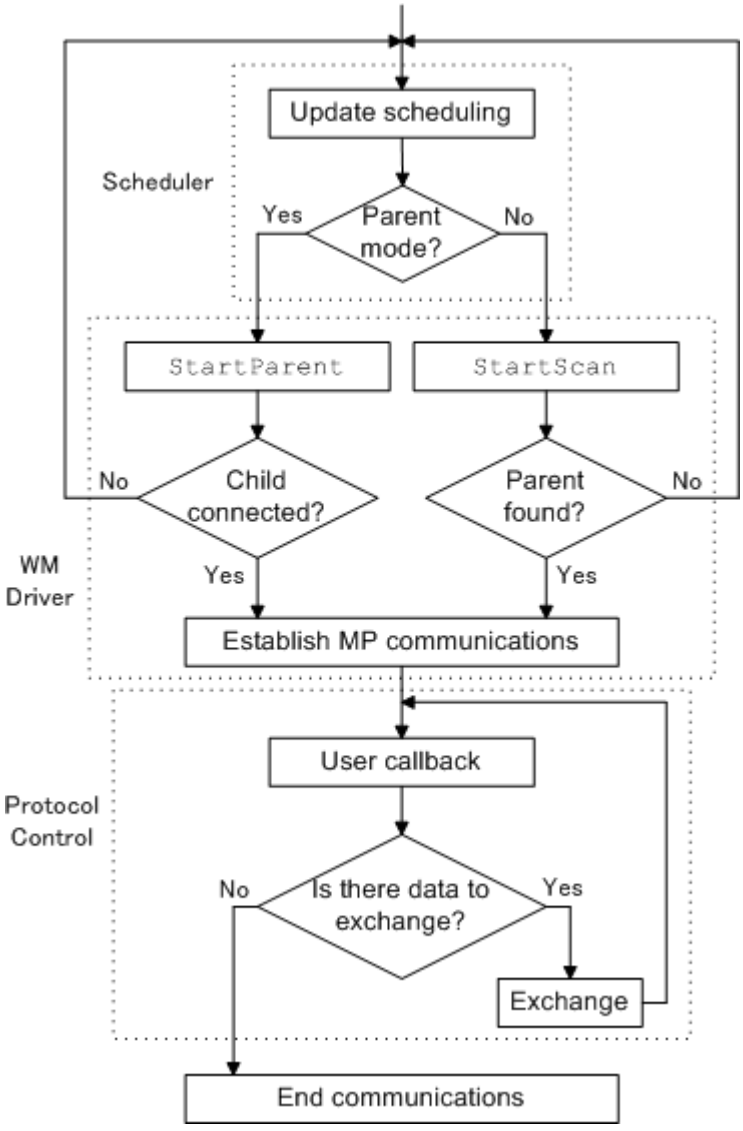
2 WXC Library Operations

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

2.1 Overall Structure

The operational flow of the WXC library is shown in Figure 2-1. 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.

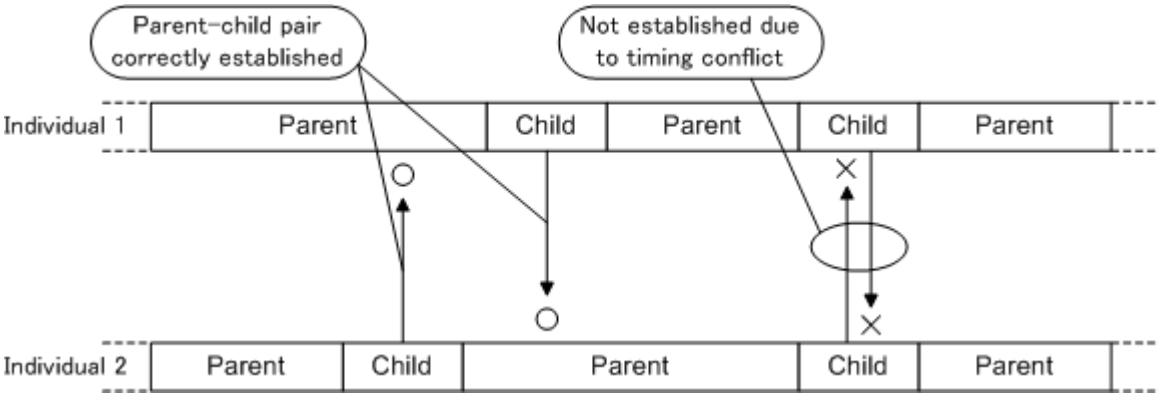
2.1.2 Scheduler

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.

The following figure 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 WX 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);
}
```