Wireless Communication Tutorial

NITRO-SDK

Version 1.1.1

The contents in this document are highly confidential and should be handled accordingly.

Contents

TWL-06-0009-001-A
Released: October 20, 2008

Revision History

Version	Revision Date	Description
1.1.1	2008/09/16	Added information on TWL (hardware and SDK name changes).
1.1.0	2005/11/21	3 Updated to reflect current `dataShare-Model` 4.1 Corrected text to reflect changes to WH specifications.
1.0.1	2005/04/18	3 Corrected description (Added description of `wh_config.h` and sample source code.) 3.1 Corrected description (Added section about setting `wh_config.h` Deleted description about internal dynamic memory allocation)
1.0.0	2004/11/24	Initial version.

1 Overview of the Wireless Manager

1.1 Role of the Wireless Manager

The *wireless manager* (WM) is situated between the NITRO-compatible wireless hardware and applications. It receives information and shares that information directly with the hardware. The wireless manager is a library providing relatively low-level parts.

The *wireless manager library* is mainly an implementation of a game-specific wireless communication method. The wireless manager library provides a unique protocol called the *MP communication protocol*. Frameworks are also included, such as a data sharing function that operates on that protocol.

This document explains the fundamentals needed to use the wireless manager. This document also includes explanations, taking sample programs as examples and implementing them in real applications.

1.2 MP Communication Protocol

As described in the programming manual, there are three separate ways to use the NITRO and TWL wireless features depending on your purposes. The three modes are:

- Infrastructure
- DS Wireless Play
- Single-Card Play

This tutorial addresses only the DS Wireless Play mode.

In DS Wireless Play mode, communication occurs wirelessly while each connected device has a game card inserted. In Single-Card Play mode, a game card is inserted in one machine, and the other machines operate by downloading the program from that machine. In Infrastructure mode, communication occurs using the Internet.

When communicating wirelessly after a program downloaded in Single-Card Play mode has started, communication occurs in either the DS Wireless Play or Infrastructure mode.

The protocol normally used in DS Wireless Play mode is called the *MP (Multi Poll) Communication* protocol. This protocol provides the functionality called *sending and receiving data in real-time with multiple machines*, which is necessary in many communication game applications.

Using the MP communication protocol, communication occurs in the following steps as one cycle:

1. The parent delivers (broadcasts) data to all children.
2. All children return a response to the parent.
3. The parent notifies (broadcasts) that the communication cycle is finished.

TWL-06-0009-001-A
Released: October 20, 2008

Note that each child communicates directly with only the parent and does not communicate directly with other children. Also, the real-time aspect is a priority. As such, another feature is that instead of being able to perform communication of one to several cycles in one picture frame (1/60th of a second), the amount of sendable/receivable data in one cycle is comparatively small.

1.3 Data Sharing

Data sharing is a communication method for realizing on the MP communication protocol a technique called *sharing data in real-time with all communicating devices*, used frequently by game applications. This technique is realized in such a way that the parent collects data from each child, lumps it together, and then delivers it to all children as *shared data*.

Pulling this together (as in the steps mentioned in section 1.2), the steps are:

1. The parent distributes shared data to all children.
2. Each child responds with its own specific information to the parent.
3. The parent collects returned information as shared data for the next send.

Note that the shared data received by each child is the data the parent collected from each child in the previous cycle.

Key sharing treats each device's key data as shared data.

Data sharing is one sample application of the MP communication protocol. Key sharing is one example of how to use data sharing. These three terms should not be spoken of on the same level but, depending on the circumstances, they may be described in parallel in the manual or sample programs. Do not confuse these terms. This manual primarily covers data sharing.

2 Operating the Wireless Manager

2.1 Organization of the Wireless Manager

In NITRO and TWL, the wireless communication unit is connected to the ARM7 bus (see the hardware block diagram in the "*NITRO Programming Manual*"). In other words, the wireless communication unit is under the control of the subprocessor (ARM7).

Therefore, to control the communication features from the *main processor* (ARM9) in a normal game, it is necessary to go through the *subprocessor* (ARM7). Many WM-related APIs have been implemented as asynchronous functions for streaming requests to the ARM7 in FIFO. Because the result of the request is also sent via FIFO, it is received by the main processor, causing the callback stored by the user to be invoked. This allows you to obtain the result.

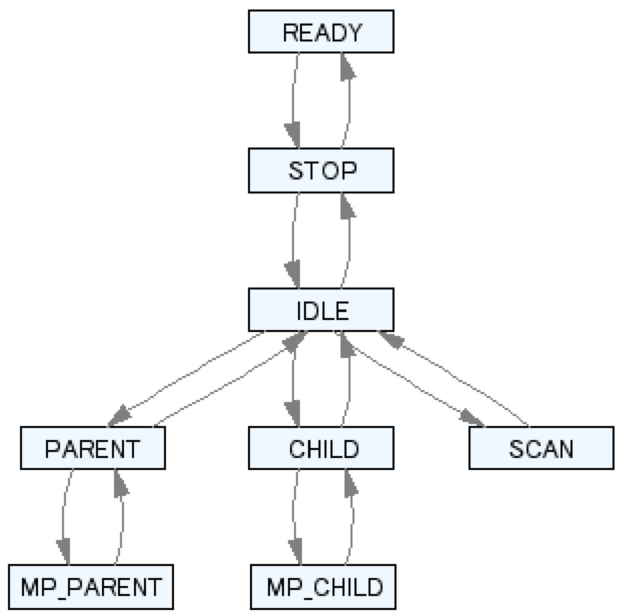
In the sample programs covered in this tutorial, this *issue request and receive results with callback* operation is treated as one set. Serial processing is basically realized as follows (A and A’, and B and B’ are sets).

1. Call function A to send a request to ARM7. (Make the callback set at this time A’.)
2. Send notification that A’ was called and processing is complete. A’ calls function B to send the next request.
3. B’ is called, which was set when calling B, and B’ calls C.
4. (The same pattern repeats.)

2.2 Transitioning Between Internal States

The ARM7 that controls wireless communication is a machine that goes back and forth between several internal states. The following figure shows the main internal states.

Figure 2-1 Transitions Between Wireless Communication Internal States



Only the states needed for this explanation appear in this diagram. There is a more detailed figure showing transitions between internal states in the "*NITRO SDK Function Reference Manual*."

Each of the transitions shown above with arrows has a corresponding function. You can advance the processing by calling them in order. With few exceptions, you can transition from a state only to a neighboring state connected by arrows. For example, you cannot transition in a single bound from IDLE to MP_CHILD.

3 Implementing the dataShare-Model

This completes the discussion of the fundamentals necessary for understanding the SDK sample program (`dataShare-Model`). Additional considerations of the sample program follow.

The sample program implements data sharing using the WH library wrapper libraries, `wh.h`, `wh_config.h`, and `wh.c` (hereafter referred to as the *WH library*). The functions needed to implement normal wireless communication programs are gathered in the WH library. (The sample source code for the WH library is stored below the `$TwlSDK/build/demos/wireless_shared/wh` directory.)

This section explains how to use the WH library for sample program tasks, such as:

- Initializing
- Connecting
- Processing Synchronously
- Disconnecting and Terminating Processing

3.1 Initializing

This section describes the procedure to implement data sharing using the WH library.

The various wireless settings listed in `wh_config.h` should be adjusted to match the specifications of the application. Wireless communication establishes virtual communication channels called “ports,” and if the application uses MP communication in addition to data sharing, the port numbers for each must not conflict.

The maximum send/receive size for data sharing is limited by the number of children connected. To use data sharing for large volumes of data, these values must be adjusted accordingly.

The DMA channels used by the WM library are also set here. Change the values so that there is no conflict with the DMA channels used by the application for other processes, such as the FS library and GX library.

```
// DMA number used by wireless
#define WH_DMA_NO              2

// Max. number of children (Not including parents)
#define WH_CHILD_MAX           15

// Max. size of shareable data
#define WH_DS_DATA_SIZE         12

// Max size of data that can be sent in one communication
// If using normal communication in addition to data sharing, increase
// this value as needed. Be sure to add the size of the additional
// headers/footers resulting from sending multiple packets.
// For details, see docs/TechnicalNotes/WirelessManager.doc.
// GUIDELINE: Guideline Standard Points (6.3.2)
// We recommend the keeping the time required for a single MP communication
// as calculated by the reference’s wireless manager (WM)→Tables/information→
//  wireless communication time calculation sheet
// under 5,600 microseconds.
#define WH_PARENT_MAX_SIZE     (WH_DS_DATA_SIZE * (1 + WH_CHILD_MAX) + 4)
#define WH_CHILD_MAX_SIZE      (WH_DS_DATA_SIZE)

// Port used for normal MP communication
#define WH_DATA_PORT           14

// Priority used for normal MP communication
#define WH_DATA_PRIO           WM_PRIORITY_NORMAL

// Port used for data sharing
#define WH_DS_PORT             13
```

Next, define the type of data to share. When the maximum number of connected children is 15, a maximum of 12 bytes can be shared using data sharing. (The sharable data size varies with the maximum number of connected children.) The data size must not exceed 12 bytes.

```
typedef struct ShareData_ {
   u8  macadr[4];  // MAC address
   u32 count;     // frame count
   u16 level;     // signal reception strength
   s16 data;      // graph display information

}ShareData;
```

Next, confirm the shared data region for send/receive in the program. For the receive buffer, the region (in bytes) must be at least (shared data size x (maximum number of connected children + 1)).

```
static u8        sSendBuf[256] ATTRIBUTE_ALIGN(32);
static u8        sRecvBuf[256] ATTRIBUTE_ALIGN(32);
```

Next, set the V-Blank interrupt. The V-Blank interrupt is needed in section "3.3 Processing Synchronously."

```
    // interrupt setting
    OS_SetIrqFunction( OS_IE_V_BLANK , VBlankIntr );
    (void)OS_EnableIrqMask( OS_IE_V_BLANK );
    (void)GX_VBlankIntr( TRUE );
    (void)OS_EnableIrq();
    (void)OS_EnableInterrupts();
```

The necessary preparations in the program are completed, so use `WH_Initialize` to initialize wireless communications.

The `WH_Initialize` function allocates the send/receive data buffer necessary for wireless communication and performs all processing necessary to initialize the wireless hardware. We recommend using the `WH_Initialize` function unless you want to perform detailed settings in the program.

3.2 Connecting

In the sample program, the process enters the main loop after the `WH_Initialize` function ends and then branches by referencing the wireless communications state returned by the `WH_GetSystemState` function and the state variable `sSysMode`, which is changed using a menu selection.

This is the relevant portion of the sample program:

```
switch (whstate)
{
case WH_SYSSTATE_ERROR:
    // WH state has priority when error occurs
    changeSysMode(SYSMODE_ERROR);
    break;

case WH_SYSSTATE_MEASURECHANNEL:
    {
        u16 channel = WH_GetMeasureChannel();
        sTgid++;
        (void)WH_ParentConnect(WH_CONNECTMODE_DS_PARENT, sTgid, channel);
    }
    break;

default:
    break;
}

PR_ClearScreen(&sInfoScreen);

// Load test.
forceSpinWait();

switch (sSysMode)
{
case SYSMODE_SELECT_ROLE:
    // Role (Parent & Child) selection screen
    ModeSelectRole();
    break;

case SYSMODE_SELECT_CHANNEL:
    // Channel selection screen.
    ModeSelectChannel();
    break;

case SYSMODE_LOBBY:
    // Lobby screen.
    ModeLobby();
    break;
```

If initialization is successful, the state becomes `WH_SYSSTATE_IDLE` (the idle state) immediately after the `WH_Initialize` function ends. The initial value of `sSysMode` is `SYSMODE_SELECT_ROLE`.

The first routine to get called is `ModeSelectRole`. If Start (Parent mode) has been selected in `ModeSelectRole`, the parent-mode connection process is performed. If Start (Child mode) has been selected, the child-mode connection process is performed.

3.2.1 Connecting in Parent Mode

The parent must select a channel to use before communications can begin. In the sample program, the channel is decided in one of two ways: manual selection and automatic selection.

Manual selection is performed by the `ModeSelectChannel` routine, which is called when *Select channel* is chosen from the menu screen. The `WH_GetAllowedChannel` function is used to get a list of usable communications channels from which to the selection can be made.

For automatic selection, the `WH_StartMeasureChannel` function is first used to measure the radio-wave usage condition, and after this is completed the `WH_GetMeasureChannel` function is called to get the most open channel. You can determine whether the `WH_StartMeasureChannel` function has completed measurement of radio wave usage by checking whether it has returned `WH_SYSSTATE_MEASURECHANNEL`.

After that, to start connection of data sharing in parent mode, the `WH_ParentConnect` function gets called with the first argument set to `WH_CONNECTMODE_DS_PARENT` and the third argument set to the selected channel.

```
switch (sRoleMenuWindow.selected)
{
case 0:
    if (sForcedChannel == 0)
    {
        // Based on radio wave usage rate, get optimal channel and connect.
        (void)WH_StartMeasureChannel();

    }
    else
    {
        sTgid++;

        // Update userGameInfo in accept-entry state
        updateGameInfo(TRUE);

        // Delete cached parent information
        MI_CpuClear8(sBssDesc, sizeof(sBssDesc));

        // Start connection using manulally selected channel.
        (void)WH_ParentConnect(WH_CONNECTMODE_DS_PARENT, sTgid, sForcedChannel);
    }
    changeSysMode(SYSMODE_LOBBY);
    break;
```

3.2.2 Connecting in Child Mode

To begin a connection in the child mode of data sharing requires first scanning to find parents and then deciding which parent to make a connection with.

In the sample program, the `ModeSelectRole` routine begins scanning for parents by calling the `WH_StartScan` function.

```
case 1:
    {
        // Start searching for parents.
        static const u8 ANY_PARENT[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
        enum
        { ALL_CHANNEL = 0 };

        initWindow(&sSelectParentWindow);
       setupWindow(&sSelectParentWindow, 16, 16, WIN_FLAG_SELECTABLE, 8*2, 8, 16);
        (void)WH_StartScan(scanCallback, ANY_PARENT, ALL_CHANNEL);
        changeSysMode(SYSMODE_SCAN_PARENT);
    }
    break;
```

The callback specified in the first argument of the `WH_StartScan` function gets called each time a parent is discovered during scanning. The `scanCallback` routine actually specified for this argument in the sample program performs a process that registers discovered parents in a list. A bitmap of valid channels is specified for the third argument, but it is not necessary to check ahead of time whether the specified channels are valid.

The `ModeSelectParent` routine, which is called during scanning, displays the parents found by scanning, and waits for the user to select one.

To start the connection in the child mode of data sharing, first end the scan with the `WH_EndScan` function and check that the `WH_GetSystemState` function returns `WH_SYSSTATE_IDLE`, and set `WH_CONNECTMODE_DS_CHILD` to the first argument and call the `WH_ChildConnect` function.

```
// Has user closed the parent-search screen?
if ((sSelectParentWindow.state == WIN_STATE_CLOSED))
{
    if (WH_GetSystemState() == WH_SYSSTATE_SCANNING)
    {
        // If scanning for parents, end the scan process
        (void)WH_EndScan();
        return;
    }

    if (WH_GetSystemState() == WH_SYSSTATE_IDLE)
    {
        if (sSelectParentWindow.selected < 0)
        {
            WH_Finalize();
            changeSysMode(SYSMODE_SELECT_ROLE);
            return;
        }
        // If not scanning and user has selected a parent, commence data sharing
        (void)WH_ChildConnect(WH_CONNECTMODE_DS_CHILD,
                              &(sBssDesc[sSelectParentWindow.selected]));
        changeSysMode(SYSMODE_LOBBYWAIT);
    }
}
```

3.3 Processing Synchronously

When the connection completes normally in the parent or child mode of data sharing, the state obtained with the `WH_GetSysState` function transitions to `WH_SYSSTATE_DATASHARING` (data sharing).

For stable wireless communication, you must call the synchronous processing function `WH_StepDS` before starting the MP communication cycle of that frame. For the V-count where the WM library prepares MP communication, the parent is 260 and the child is 240 by default. It is designed to be as efficient as possible when called with the V-Blank interrupt (V-count is 192). With this in mind, the sample program calls the `WH_StepDS` function inside the `updateShareData` routine immediately after the start of the V-Blank interrupt (i.e., immediately after the `OS_WaitVBlankIntr` function).

```
static void updateShareData(void)
{
    if (WH_GetSystemState() == WH_SYSSTATE_DATASHARING)
    {
        if (WH_StepDS(sSendBuf))
        {
            u16 i;
            for (i = 0; i < WM_NUM_MAX_CHILD + 1; ++i)
            {
                u8 *adr;
                ShareData *sd;

                adr = (u8 *)WH_GetSharedDataAdr(i);
                sd = (ShareData *) & (sRecvBuf[i * sizeof(ShareData)]);

                if (adr != NULL)
                {
                    MI_CpuCopy8(adr, sd, sizeof(ShareData));
                    sRecvFlag[i] = TRUE;

                }
                else
                {
                    sd->level = 0;
                    sd->data = 0;
                    sRecvFlag[i] = FALSE;
                }
            }

            sNeedWait = FALSE;

        }
        else
        {
            u16 i;
            for (i = 0; i < WM_NUM_MAX_CHILD + 1; ++i)
            {
                sRecvFlag[i] = FALSE;
            }

            sNeedWait = TRUE;
        }

    }
    else
    {
        u16 i;
        for (i = 0; i < WM_NUM_MAX_CHILD + 1; ++i)
        {
            sRecvFlag[i] = FALSE;
        }

        sNeedWait = FALSE;
    }
}
```

For data shared by synchronous processing, obtain the top address using the WH_GetSharedDataAdr function and copy it to the receive data region allocated in the program.

In the main loop, use the WH_GetConnectMode function to determine the connection mode. If in the parent mode, call the ModeParent routine. If in the child mode, call the ModeChild routine. The send/receive results are displayed in each routine.

The link strength icon is also displayed, which graphically shows the strength of the communication link while sharing data.

3.4 Disconnecting and Terminating Processing

When you want to disconnect a specific child from its parent with a user operation, use the WM_Disconnect function. When you want to disconnect multiple or all children at once, use the WM_DisconnectChildren function.

Using the WH library, when you want to terminate wireless communication by calling the WH_Finalize function, you can perform the appropriate end processing by evaluating the connection mode and the current WH library state. With the WH_Finalize function, wireless communication transitions to the IDLE state. From there, by sequentially calling the WM_PowerOff, WM_Disable, and WM_Finish functions (or the WM_End function to perform all three), you can completely finish, including disconnecting the wireless communication hardware.

4 WH Library

The WH library is a collection of functions needed to implement normal wireless communication programs. This section explains the functions collected in the WH library (including functions not used in the sample programs).

4.1 Function Reference (Initialization, Termination, Reset)

This section discusses the functions that initialize, terminate, and reset

4.1.1 WH_Initialize Function

```
C Specification:      int WH_Initialize(void);
Arguments:            None
Return Values:        TRUE    Success
                      FALSE   Failure
```

Normally, `WH_Initialize` automatically allocates the communication send/receive data buffer necessary for wireless communication and initializes wireless communication hardware. The wireless communication state transitions to IDLE. When TRUE is returned, the `WM_Init` function succeeded, and the `WM_Enable` function was called successfully. Initialization processing is complete when the return value of the `WH_GetSystemState` function becomes `WH_SYSSTATE_IDLE`.

You must create a heap in the main memory for the internally-called `OS_Alloc` function.

4.1.2 WH_Finalize Function

```
C Specification:      int WH_Finalize(void);
Arguments:            None
Return Values:        TRUE    Success
                      FALSE   Failure
```

`WH_Finalize` calls the appropriate end process determined from the WH library state and the connection mode. The wireless communication state transitions to IDLE after processing completes. When TRUE is returned, the function call for end processing succeeded. The processing ends when the return value of the `WH_GetSystemState` function becomes `WH_SYSSTATE_IDLE`.

To completely terminate wireless communication, you must call the `WM_PowerOff`, `WM_Disable`, and `WM_Finish` functions, or just the `WM_End` function.

4.1.3 WH_Reset Function

```
C Specification:      int WH_Reset(void);
Arguments:            None
Return Values:        TRUE    Success
                      FALSE   Failure
```

`WH_Reset` transitions the wireless communication state to IDLE regardless of the current state (such as connection mode). When TRUE is returned, the `WH_Reset` call succeeded. The process completes when the return value of `WH_GetSystemState` becomes `WH_SYSSTATE_IDLE`.

4.2 Function Reference (Connection)

This section discusses the two connection functions, `WH_ForceChannel` and `WH_Connect`.

4.2.1 WH_ParentConnect Function

```
C Specification:      BOOL WH_ParentConnect(int mode, u16 tgid, u16 channel);
Arguments:            mode           Connection mode
                      tgid           Parent communication tgid
                      channel        Parent communication channel
Return Values:        TRUE           Success
                      FALSE          Failure

Connection mode definitions:
enum {
   WH_CONNECTMODE_MP_PARENT,  // Parent MP connection mode
   WH_CONNECTMODE_MP_CHILD,   // Child MP connection mode
   WH_CONNECTMODE_KS_PARENT,  // Parent key-sharing connection mode
   WH_CONNECTMODE_KS_CHILD,   // Child key-sharing connection mode
   WH_CONNECTMODE_DS_PARENT,  // Parent data-sharing connection mode
   WH_CONNECTMODE_DS_CHILD,   // Child data-sharing connection mode
   WH_CONNECTMODE_NUM
};
```

This function starts the wireless communication connection in parent mode. It automatically transitions to data sharing and key sharing. When TRUE is returned, the function call for connection processing succeeded. Whether for a parent or child, the process completes when the `WH_GetSystemState` function returns the following return values: MP connection is `WH_SYSSTATE_CONNECTED`, data sharing is `WH_SYSSTATE_DATASHARING`, and key sharing is `WH_SYSSTATE_KEYSHARING`.

4.2.2 WH_ChildConnect Function

```
C Specification:      BOOL WH_ChildConnect(int mode, WMBssDesc *bssDesc);
Arguments:            mode            Connection mode
                      bssDesc         bssDesc of parent connecting to
Return Values:        TRUE            Success
                      FALSE           Failure

Conneciton mode definitions:
enum {
   WH_CONNECTMODE_MP_PARENT,  // Parent MP connection mode
   WH_CONNECTMODE_MP_CHILD,   // Child MP connection mode
   WH_CONNECTMODE_KS_PARENT,  // Parent key-sharing connection mode
   WH_CONNECTMODE_KS_CHILD,   // Child key-sharing connection mode
   WH_CONNECTMODE_DS_PARENT,  // Parent data-sharing connection mode
   WH_CONNECTMODE_DS_CHILD,   // Child data-sharing connection mode
   WH_CONNECTMODE_NUM
};
```

This function starts the wireless communication connection in child mode.

4.3 Function Reference (MP Communication)

4.3.1 WH_SetReceiver Function

```
C Specification:      void WH_SetReceiver(WHReceiver proc);
Arguments:            proc    WHReceiver type callback function
Return Values:        None

WHReceiver type definitions:
              typedef void (*WHReceiver)(u16 aid, u16* data, u16 size);
```

`WH_SetReceiver` sets the MP communication data receive callback function. There is no need to set this when data sharing or key sharing.

The send source `aid`, received data, and receive data size are passed to the callback function.

4.3.2 WH_SendData Function

```
C Specification:      int WH_SendData(
void *data, u16 datasize, WHSendCallbackFunc callback);
Arguments:    data            top address of send data
              datasize        size of send data
              callback        WHSendCallbackFunc type callback function
Return Values:TRUE            Success
              FALSE           Failure

WHSendCallbackFunc type definitions:
              typedef void (*WHSendCallbackFunc)(BOOL result);
```

`WH_SendData` starts an MP communication data send. You do not need to call this when data sharing or key sharing. When TRUE is returned, the `WM_SetMPDataToPortEx` function call succeeded. The process completes when the callback function is called.

The send results are passed to the callback function. You must not change the contents of the send data buffer until the callback function is called.

4.4 Function Reference (Data Sharing)

4.4.1 WH_StepDS Function

```
C Specification:      int WH_StepDS(void *data);
Arguments:            data    top address of send data
Return Values:        TRUE    Success
                      FALSE   Failure
```

`WH_StepDS` proceeds to the next step in the synchronous processes for data sharing. When TRUE is returned, the process is complete. To get shared data, use the `WH_GetSharedDataAdr` function.

For stable wireless communication, you must call this function before starting the MP communication cycle of that frame. We recommend calling it immediately after starting the V-Blank interrupt.

4.4.2 WH_GetSharedDataAdr Function

```
C Specification:      u16 *WH_GetSharedDataAdr(u16 aid);
Arguments             aid     aid of child you want to get shared data for
Return Values:        top     address of shared data of specified child
                      NULL    is returned when it fails.
```

Call `WH_GetSharedDataAdr` when you want to get data sharing shared data by specifying the child.

4.5 Function Reference: Key Sharing

4.5.1 WH_GetKeySet Function

```
C Specification:      int WH_GetKeySet(WMKeySet *keyset);
Arguments:            keyset  pointer to buffer that stores shared key data
Return Values:        TRUE    Success
                      FALSE   Failure
```

`WH_GetKeySet` stores key data shared with key sharing in the buffer. The process is complete when TRUE is returned.

For stable wireless communication, the `WH_GetKeySet` function must be called before starting MP communication cycle of that frame. We recommend calling immediately after starting the V-Blank interrupt.

4.6 Function Reference: Get State

4.6.1 WH_GetAllowedChannel Function

```
C Specification:      u16 WH_GetAllowedChannel(void);
Arguments:            None
Return Values:        bit pattern of communication channels permitted for use
```

Internally, `WH_GetAllowedChannel` calls the `WM_GetAllowedChannel` function.

4.6.2 WH_GetConnectMode Function

```
C Specification:      int WH_GetConnectMode(void);
Arguments:            None
Return Values:        the set connection mode
```

`WH_GetConnectMode` returns the connection mode set as an argument with the `WH_ChildConnect` function. The return values are undetermined until `WH_ChildConnect` is called. Until the next time `WH_ChildConnect` is called, the previously-set connection mode is returned.

4.6.3 WH_GetBitmap Function

```
C Specification:              u16 WH_GetBitmap(void);
Arguments:            None
Return Values:        bit pattern showing connected terminal
```

The bit corresponding to the connected terminal is set to 1. The lowest bit corresponds to the parent (`aid`=0), and the highest bit corresponds to the 15th child (`aid`=15).

4.6.4 WH_GetSystemState Function

```
C Specification:      int WH_GetSystemState(void);
Arguments:            None
Return Values:        Internal state of WH library

Definitions of WH library internal states:
enum {
   WH_SYSSTATE_STOP,         // initial state
   WH_SYSSTATE_IDLE,         // standing by
   WH_SYSSTATE_SCANNING,     // scanning
   WH_SYSSTATE_BUSY,         // connecting
   WH_SYSSTATE_CONNECTED,    // connection complete (communication is possible in
this state)
   WH_SYSSTATE_DATASHARING, // connected with data-sharing enabled
   WH_SYSSTATE_KEYSHARING,  // connected with key-sharing enabled
   WH_SYSSTATE_ERROR,       // error has occurred
   WH_SYSSTATE_NUM
};
```

`WH_GetSystemState` obtains the current internal state of the WH library.

4.6.5 WH_GetLastError Function

```
C Specification:              int WH_GetLastError(void);
Arguments:                    None
Return Values:                error code

Definitions of error codes:
enum {
   // your own error codes
   WH_ERRCODE_DISCONNECTED = WM_ERRCODE_MAX, // disconnected from parent
   WH_ERRCODE_PARENT_NOT_FOUND,              // no parent
   WH_ERRCODE_NO_RADIO,                      // wireless use not possible
   WH_ERRCODE_LOST_PARENT,                   // parent not found
   WH_ERRCODE_MAX
};
```

`WH_GetLastError` obtains the details of the error that just occurred.

5 Appendix

5.1 WH_StateInXXXX and WH_StateOutXXXX Functions

As stated in "2.1 Organization of the Wireless Manager," the Wireless Manager API performs wireless communication with a combination of the request send function (call function) and the callback function that receives notification.

For internal functions in the WH library, the request send function is `WH_StateInXXXX` and the callback function is `WH_StateOutXXXX`.

5.1.1 Parent/Child Shared Functions in WH and WM

Table 5-1 Corresponding WM Functions (Parent/Child Shared Functions)

WH Library Function Names	Corresponding Wireless Manager Function
WH_StateInInitialize	WM_Init
WH_StateInEnable WH_StateOutEnable	WM_Enable
WH_StateInPowerOn WH_StateOutPowerOn	WM_PowerOn
WH_StateInReset WH_StateOutReset	WM_Reset
WH_StateInSetMPData WH_StateOutSetMPData	WM_SetMPDataToPortEx
WH_StateInPowerOff WH_StateOutPowerOff	WM_PowerOff
WH_StateInDisable WH_StateOutDisable	WM_Disable

5.1.2 Parent Functions in WH and WM

Table 5-2 Corresponding WM Functions (Parent Functions)

WH Library Function Names	Corresponding Wireless Manager Function

`WH_StateInMeasureChannel`
`WH_NextMeasureChannel`
`WH_StateOutMeasureChannel`
	`WM_GetAllowedChannel`
	`WM_MeasureChannel`

`WH_StateInSetParentParam` `WH_StateOutSetParentParam`	`WM_SetParentParameter`
`WH_StateInStartParent` `WH_StateOutStartParent`	`WM_StartParent`
`WH_StateInStartParentMP` `WH_StateOutStartParentMP`	`WM_StartMP` `WM_StartDataSharing` is also called when in data sharing mode.
`WH_StateInStartParentKeyShare` `WH_StateOutStartParentKeyShare`	`WM_StartKeySharing`
`WH_StateInEndParentKeyShare` `WH_StateOutEndParentKeyShare`	`WM_EndKeySharing`
`WH_StateInEndParentMP` `WH_StateOutEndParentMP`	`WM_EndMP`
`WH_StateInEndParent` `WH_StateOutEndParent`	`WM_EndParent`
`WH_StateInDisconnectChildren` `WH_StateOutDisconnectChildren`	`WM_DisconnectChildren`

5.1.3 Child Functions in WH and WM

Table 5-3 Corresponding WM Functions (Child Functions)

WH Library Function Names	Corresponding Wireless Manager Function
WH_StateInStartScan WH_NextScan WH_StateOutStartScan	WM_GetAllowedChannel WM_StartScan
WH_StateInEndScan WH_StateOutEndScan	WM_EndScan
WH_StateInStartChild WH_StateOutStartChild	WM_StartConnect
WH_StateInStartChildMP WH_StateOutStartChildMP	WM_StartMP WM_StartDataSharing is also called when in data sharing mode.
WH_StateInStartChildKeyShare WH_StateOutStartChildKeyShare	WM_StartKeySharing
WH_StateInEndChildKeyShare WH_StateOutEndChildKeyShare	WM_EndKeySharing
WH_StateInEndChildMP WH_StateOutEndChildMP	WM_EndMP
WH_StateInEndChild WH_StateOutEndChild	WM_Disconnect