



Synway SHR Series

SHR-4D/PCI

Radar Data Recording Board

User Manual

Version 1.2.0

Hangzhou Synway Information Engineering Co., Ltd

www.synway.net

Contents

| | |
|--|------------|
| Contents | i |
| Copyright Declaration | iii |
| Revision History | iv |
| Chapter 1 Driver-provided API Functions | 1 |
| 1.1 Driver Architecture | 1 |
| 1.2 Board Initialization and Uninstallation..... | 1 |
| 1.3 Application Programming Mode..... | 2 |
| 1.4 Driver-provided API Functions | 2 |
| 1.4.1 Functions for Initialization..... | 3 |
| 1.4.1.1 Rd_OpenDevice..... | 3 |
| 1.4.1.2 Rd_CloseDevice | 3 |
| 1.4.2 Functions for Radar Data Recording | 3 |
| 1.4.2.1 Rd_StartRec..... | 3 |
| 1.4.2.2 Rd_StopRec..... | 5 |
| 1.4.2.3 Rd_DeleteRecListFile | 6 |
| 1.4.2.4 Rd_RecToMem | 6 |
| 1.4.3 Functions for Radar Data Playback..... | 9 |
| 1.4.3.1 Rd_StartPlay | 9 |
| 1.4.3.2 Rd_StopPlay | 10 |
| 1.4.3.3 Rd_GetPlayPercent | 10 |
| 1.4.3.4 Rd_DeletePlayListFile..... | 10 |
| 1.4.3.5 Rd_SetBaudRate | 11 |
| 1.4.3.6 Rd_PlayToMem..... | 11 |
| 1.4.4 Callback Function..... | 13 |
| 1.4.4.1 Rd_SetCallBack | 13 |
| 1.4.5 Other Functions..... | 14 |
| 1.4.5.1 Rd_GetLastErrMsg | 14 |
| 1.4.5.2 Rd_QueryOp..... | 15 |
| 1.4.5.3 Rd_SetUserInfo..... | 16 |
| 1.4.5.4 Rd_GetFileInfo | 17 |
| Chapter 2 Board Configuration | 18 |
| 2.1 Manual Compilation..... | 18 |
| 2.2 Compilation by Configuration Program..... | 18 |
| 2.3 Channel Numbering Rule..... | 19 |

| | |
|---|-----------|
| Chapter 3 Demo Program | 20 |
| Appendix A Technical/sales Support | 21 |

Copyright Declaration

All rights reserved; no part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, without prior written permission from Hangzhou Synway Information Engineering Co., Ltd (hereinafter referred to as 'Synway').

Synway reserves all rights to modify this document without prior notice. Please contact Synway for the latest version of this document before placing an order.

Synway has made every effort to ensure the accuracy of this document but does not guarantee the absence of errors. Moreover, Synway assumes no responsibility in obtaining permission and authorization of any third party patent, copyright or product involved in relation to the use of this document.

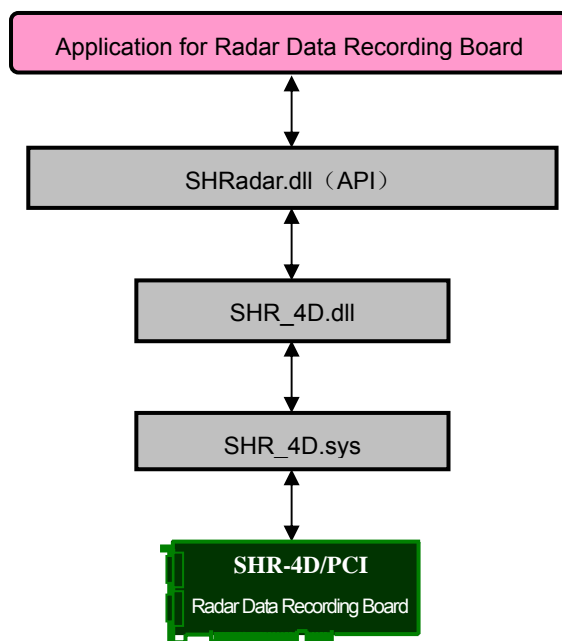
Revision History

| Version | Date | Comments |
|---------------|---------|---|
| Version 1.0.0 | 2005-12 | Initial Publication |
| Version 1.1.0 | 2006-6 | Changes: Added and described the functions for data recording and playback via memory |
| Version 1.2.0 | 2006-12 | Updated with the driver |
| | | |

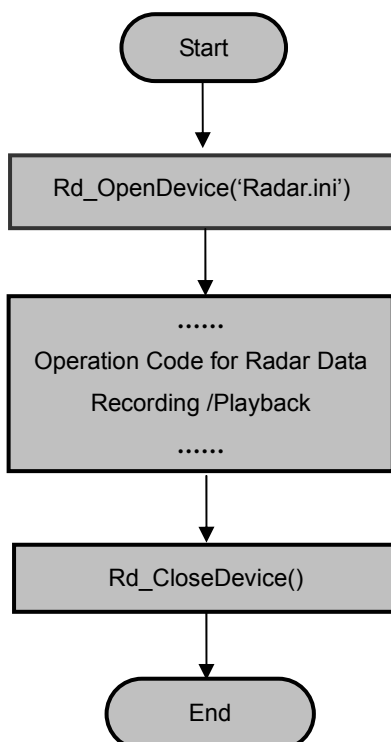
Note: Please visit our website <http://www.Synway.net> to obtain the latest version of this document.

Chapter 1 Driver-provided API Functions

1.1 Driver Architecture



1.2 Board Initialization and Uninstallation



1.3 Application Programming Mode

Our driver accepts two application programming modes, namely event mode (callback mode) and polling mode.

- **Event Mode:** The application defines a callback function based on a callback declaration provided by the driver, and passes the function pointer to the driver during initialization. At runtime, the driver, when necessary, calls the corresponding callback function in the application automatically.
- **Polling Mode:** The application creates a timer, and self-defines each of various runtime statuses which changes according to the driver's current runtime status, and executes the next step—an API function call.

1.4 Driver-provided API Functions

All the API functions provided by the driver are encapsulated in the library file 'SHRadar.dll', and the function prototypes contained in the header file 'SHRadar.h'. Below is a complete list of the functions.

| Classification | Function Name |
|------------------------------------|---------------------------------------|
| Functions for initialization | Rd_OpenDevice |
| | Rd_CloseDevice |
| Functions for radar data recording | Rd_StartRec |
| | Rd_RecToMem |
| | Rd_StopRec |
| | Rd_DeleteRecListFile |
| Functions for radar data playback | Rd_StartPlay |
| | Rd_PlayToMem |
| | Rd_StopPlay |
| | Rd_GetPlayPercent |
| | Rd_DeletePlayListFile |
| | Rd_SetBaudRate |
| Callback function | Rd_SetCallBack |
| Other functions | Rd_GetLastErrMsg |
| | Rd_QueryOp |
| | Rd_SetUserInfo |
| | Rd_GetFileInfo |

1.4.1 Functions for Initialization

1.4.1.1 Rd_OpenDevice

Prototype Declaration:

```
int WINAPI Rd_OpenDevice(LPSTR lpstrConfigFile)
```

Entry Parameters:

| | |
|-----------------|-------------------------|
| lpstrConfigFile | Configuration file name |
|-----------------|-------------------------|

Return Values:

1: Initialization successful

-1: Initialization failed

Description:

This function is used for board initialization and memory allocation.

1.4.1.2 Rd_CloseDevice

Prototype Declaration:

```
void WINAPI Rd_CloseDevice()
```

Entry Parameters:

Nil

Return Values:

Nil

Description:

It is an unload function provided by the driver. The application calls this function upon exit to release relevant resources.

1.4.2 Functions for Radar Data Recording

1.4.2.1 Rd_StartRec

Prototype Declaration:

```
int WINAPI Rd_StartRec(int nCh, char *pcFileName, DWORD dwSize, char cUserInfo[800])
```

Entry Parameters:

| | |
|----------------|--------------------------------------|
| nCh | Channel number |
| PcFileName | Radar-data-saved file name |
| DwSize | Maximum file size for data recording |
| cUserInfo[800] | User information |

Return Values:

1: Function call successful

-1: Function call failed

Description:

This function is used to activate radar data recording on a particular channel (via the first call) or append record files (via the second or subsequent call). The driver assigns each channel a task queue for recording, which contains all record file names specified by the application, and automatically writes the board-collected data into the current file upon receiving them. In case the application sets a callback function for data recording, the driver automatically calls this function which in turn notifies the application of the relevant information. The current file is removed from the queue once it is full, and the next file is used. If there is no file available in the queue, the driver stops data recording on the channel and calls the callback function to inform the application of this. The function 'Rd_SetCallBack' can be used to set the callback function.

The task queue depth per channel is 100, i.e. the application can successively call this function for up to 100 times and directly pass the 100 file names to the driver.

Each target file generated during data acquisition is formatted with a file header of 1024 bytes at the top. The format of the file headers is shown as follows in the header file 'SHRadar.h'.

```
typedef struct tagFILEHEAD    // rec file header
{
    char    cStyle[20];        // file type 'Radar_Data'
    DWORD   dwSize;            // file size
    DWORD   dwSpeed;           // data transmission rate
    char    cStartTime[20];    // start time
    char    cEndTime[20];      // end time
    int     nChannel;          // record channel
    char    cObligate[152];    // obligate
    char    cUserInfo[800];    // user information
}FILEHEAD, *PFILEHEAD;
```

The meaning of each field and the number of bytes occupied are shown in the following table.

| No. | Bytes | Field Name | Description |
|-----|-------|------------|--|
| 1 | 20 | cStyle | File type: 'Radar_Data' (fixed) |
| 2 | 4 | dwSize | File size (including the length of the file header) |
| 3 | 4 | dwSpeed | Data transmission rate (now unused) |
| 4 | 20 | cStartTime | The exact start time for data transmission (written as a character string 'YYYY/MM/DD HH:MM:SS') |
| 5 | 20 | cEndTime | The exact end time for data transmission (written as a character string 'YYYY/MM/DD HH:MM:SS') |
| 6 | 4 | nChannel | Record channel |
| 7 | 148 | cObligate | Obligate (set 'cObligate=0') |
| 8 | 800 | cUserInfo | User information (left for users to write) |

Related Functions: [Rd_StopRec](#), [Rd_DeleteRecListFile](#), [Rd_QueryOp](#), [Rd_SetCallBack](#)

1.4.2.2 Rd_StopRec

Prototype Declaration:

```
void WINAPI Rd_StopRec (int nCh)
```

Entry Parameters:

nCh: Channel number

| | |
|-----|----------------|
| nCh | Channel number |
|-----|----------------|

Return Values:

Nil

Description:

1. Stops data recording on Channel nCh and clears task queues from this channel when data are being recorded into files
2. Stops data recording on Channel nCh when data are being recorded into memory

Related Functions: [Rd_StartRec](#)

1.4.2.3 Rd_DeleteRecListFile

Prototype Declaration:

```
int WINAPI Rd_DeleteRecListFile(int nCh,char *pcFileName)
```

Entry Parameters:

| | |
|------------|---|
| nCh | Channel number |
| PcFileName | File name to be removed from a task queue |

Return Values:

- 1: Successful
- 1: File can not be removed as data are being recorded. Failed
- 2: This file does not exist in the task queue.

Description:

Removes the record file with the name specified in the parameter 'PcFileName' from the task queue on the corresponding channel

1.4.2.4 Rd_RecToMem

Prototype Declaration:

```
int WINAPI Rd_RecToMem(int nCh,  
    LPBYTE pBuf,  
    DWORD dwBufSize,  
    DWORD dwOutParamVal,  
    DWORD dwOutCondition,  
    MEMBLOCKHANDLER OnRecMemBlockDone,  
    PVOID pV);
```

Entry Parameters:

| | |
|----------------|---|
| nCh | Channel number |
| pBuf | Address pointer to buffer storing recorded data |
| dwBufSize | Size of buffer storing recorded data |
| dwOutParamVal | <p>One of the following values can be assigned to the parameter for event output during memory data recording.</p> <pre>enum { END_HALF_BUFFER = 0, MEM_BYTES = 1, };</pre> <p>Details:</p> <p>If dwOutParamVal = END_HALF_BUFFER,</p> <p>the callback function 'OnRecMemBlockDone' is called when radar data fills the buffer up to half the value of dwBufSize;</p> <p>If dwOutParamVal = MEM_BYTES,</p> <p>the callback function 'OnRecMemBlockDone' is called when radar data fills the buffer up to the value of dwOutCondition.</p> |
| dwOutCondition | A parameter to specify event output conditions, valid when dwOutParamVal = MEM_BYTES |

| | |
|-------------------|--|
| OnRecMemBlockDone | <p>Callback pointer</p> <p>Callback prototype:</p> <pre>void WINAPI OnRecMemBlockDone(int ch, int nEndState, LPBYTE pBuf, DWORD dwStopOffset, PVOID pV)</pre> <p>Details:</p> <p>ch: Channel number;</p> <p>nEndReason: =0, indicates that radar data is being recorded on the channel; =1, indicates that radar data recording is complete on the channel;</p> <p>pBuf: Pointer to buffer storing recorded data</p> <p>dwStopOffset: Pointer address in the buffer</p> <p>pV: The driver uses this parameter to pass the very pointer provided by the application when calling Rd_RecToMem to the callback function. This parameter is defined by the application and can be set to 'NULL' when unused.</p> |
| pV | <p>PVOID pointer, often used for the application to pass its data structure to the callback function. It is passed by the driver to the callback function, and can be set to 'NULL' when unused.</p> |

Return Values:

1: Successful

-1: Failed

Description:

The driver records data into the buffer 'pBuf' in the application. Upon satisfying certain conditions during data acquisition, the callback pointer 'OnRecMemBlockDone' provided by the application via a function call is used to ask the application to remove data from the buffer. The parameter for event output of memory data can be set as 'END_HALF_BUFFER' or 'MEM_BYTES'. On condition that 'dwOutParamVal = END_HALF_BUFFER', the callback function 'OnRecMemBlockDone' is called when radar data fills the buffer up to half the value of dwBufSize; on condition that 'dwOutParamVal = MEM_BYTES', the callback function 'OnRecMemBlockDone' is called when radar data fills the buffer up to the value of dwOutCondition.

Related Functions: [Rd_StopRec](#)

1.4.3 Functions for Radar Data Playback

1.4.3.1 Rd_StartPlay

Prototype Declaration:

int WINAPI Rd_StartPlay (int nCh, char * pcFileName, int nBaudRate, float fStartOffset)

Entry Parameters:

| | |
|--------------|--|
| nCh | Channel number |
| pcFileName | File name |
| nBaudRate | <p>Specifies the baud rate for radar data playback. If nBaudRate = -1, the default value of the configuration item 'PlaybackBaudrate=' for the corresponding channel in the configuration file is used. If users are required to set the baud rate, they have the choices listed below.</p> <p>300bps, 600 bps, 1200 bps, 2400 bps, 4800 bps, 9600 bps, 19200 bps, 38400 bps</p> |
| fStartOffset | Start point for file playback, which is expressed as a percentage value (0~100) of the total duration. |

Return Values:

1: Start playback

-1: Fail to start playback

Description:

This function is used to activate radar data playback on a particular channel (via the first call) or append playback files (via the second or subsequent call). The driver assigns each channel a task queue for playback, which contains all playback file names specified by the application, and automatically gets data from the file and replays them to the radar terminal. If the application has set a callback function for data playback when a data file playback is complete, the driver automatically calls this function which in turn notifies the application of the relevant information, and at the same time, removes the current file from the queue and gets the next file for playback. If there is no file available in the queue, the driver stops data playback on the channel and calls the callback function to inform the application of this. The function '[Rd_SetCallBack](#)' can be used to set the callback function.

The task queue depth per channel is 100, i.e. the application can successively call this function for up to 100 times and directly pass the 100 file names to the driver.

Related Functions: [Rd_StopPlay](#), [Rd_GetPlayPercent](#), [Rd_DeleteRecListFile](#), [Rd_QueryOp](#), [Rd_SetCallBack](#)

1.4.3.2 Rd_StopPlay

Prototype Declaration:

```
void WINAPI Rd_StopPlay (int nCh)
```

Entry Parameters:

| | |
|-----|----------------|
| nCh | Channel number |
|-----|----------------|

Description:

1. Stops data playback on Channel nCh and clears task queues from this channel during playback of file data
2. Stops data playback on Channel nCh and informs the application during playback of memory data

Related Functions: [Rd_StartPlay](#)

1.4.3.3 Rd_GetPlayPercent

Prototype Declaration:

```
float WINAPI Rd_GetPlayPercent(int nCh)
```

Entry Parameters:

| | |
|-----|----------------|
| nCh | Channel number |
|-----|----------------|

Return Values:

>0: The played percentage (0~100) of the current playback file

-1: Failed

Description:

Gets the played percentage of the current playback file on channel nCh.

Related Functions: [Rd_StartPlay](#)

1.4.3.4 Rd_DeletePlayListFile

Prototype Declaration:

```
int WINAPI Rd_DeletePlayListFile(int nCh,char *pcFileName)
```

Entry Parameters:

| | |
|------------|---|
| nCh | Channel number |
| pcFileName | File name to be removed from a task queue |

Return Values:

1: Successful

-1: File can not be removed as data are being replayed. Failed

-2: This file does not exist in the list of playback files.

Description:

Removes a particular file from the task queue for playback on the corresponding channel

Related Functions: [Rd_StartPlay](#)

1.4.3.5 Rd_SetBaudRate**Prototype Declaration:**

```
int WINAPI Rd_SetBaudRate(int nCh,int nBaudRate)
```

Entry Parameters:

| | |
|-----------|---|
| nCh | Channel number |
| NBaudRate | Baud rate. Board-supported baud rates for playback are 300bps, 600 bps, 1200 bps, 2400 bps, 4800 bps, 9600 bps, 19200 bps, 38400 bps. |

Return Values:

1: Successful

-1: Fail to change the playback baud rate on this channel

Description:

Changes the playback baud rate of the data file being played on Channel nCh.

1.4.3.6 Rd_PlayToMem**Prototype Declaration:**


```
int WINAPI Rd_PlayToMem(int nCh,
    LPBYTE pBuf,
    DWORD dwBufSize,
    MEMBLOCKHANDLER OnPlayMemBlockDone,
    PVOID pV);
```

Entry Parameters:

| | |
|--------------------|--|
| nCh | Channel number |
| pBuf | Address pointer to buffer storing data to be played |
| dwBufSize | Size of buffer storing data to be played |
| OnPlayMemBlockDone | <p>Callback pointer</p> <p>Callback prototype:</p> <pre>void WINAPI OnRecMemBlockDone(int ch, int nEndState, LPBYTE pBuf, DWORD dwStopOffset, PVOID pV)</pre> <p>Details:</p> <p>Ch: Channel number;</p> <p>nEndReason: =0 indicates that radar data is being replayed on the channel; =1 indicates that radar data playback is complete on the channel;</p> <p>pBuf: Pointer to buffer storing data to be played;</p> <p>dwStopOffset: Pointer address in the buffer;</p> <p>pV: The driver uses this parameter to pass to the callback function the very pointer provided by the application when calling Rd_PlayToMem. This parameter is defined by the application and can be set to 'NULL' when unused.</p> |
| pV | PVOID pointer, often used when the application passes data structure to the callback function. It is passed by the driver to the callback function, and can be set to 'NULL' when unused. |

Return Values:

1: Successful

-1: Failed

Description:

The application first creates one (or two) buffer and fills it with data to be played, then calls this function once (or consecutively twice) to start playback. The driver activates playback operation upon receiving playback command from the memory. When a buffer has played back all its data, the driver calls the callback function OnPlayMemBlockDone to pass to the application information about this buffer (including channel number, the reason for playback stop, buffer pointer, pointer address and structure pointer specified by the application during this function call), and at the same time, automatically plays the data in the next buffer. The application can create the next buffer and pass the relevant information to the driver via callback function settings.

Related Functions: [Rd_StopPlay](#)

1.4.4 Callback Function

1.4.4.1 Rd_SetCallBack

Prototype Declaration:

```
int WINAPI Rd_SetCallBack(int nType, PVOID pfCallBack, PVOID pVoid);
```

Entry Parameters:

| | |
|-------|---|
| NType | <p>Callback function type</p> <p>The following values can be used.</p> <p>enum</p> <pre>{ RD_CB_RECORD_END = 0, RD_CB_PLAY_END = 1, };</pre> <p>This function is used to set the type of the callback function. See below for details.</p> <p>nType=RD_CB_RECORD_END: Call this function to set the callback function for data recording;</p> <p>nType=RD_CB_PLAY_END: Call this function to set the callback function for data playback.</p> |
|-------|---|

| | |
|------------|---|
| PfCallBack | <p>Callback function pointer provided by the application</p> <p>If the value of nType is RD_CB_RECORD_END or RD_CB_PLAY_END, the prototype of the callback function must be written in the following format.</p> <pre>void WINAPI xxx (int iCh, DWORD dwPara, LPSTR lpstr, PVOID pVoid)</pre> <p>Details:</p> <p>iCh: Channel number</p> <p>dwPara: If 'nType=RD_CB_RECORD_END', this parameter indicates the number of unused files ready for storing recorded data in the task queue on the particular channel;</p> <p>If 'nType=RD_CB_PLAY_END', this parameter indicates the number of unused files ready for storing playback data in the task queue on the particular channel.</p> <p>lpstr: If 'nType=RD_CB_RECORD_END', this parameter indicates the name of the current file with data recorded or being recorded in the task queue on the particular channel;</p> <p>If 'nType=RD_CB_PLAY_END', this parameter indicates the name of the current file with data played or being played in the task queue on the particular channel.</p> <p>pVoid: The driver passes to the callback function via this function, the very pointer provided by the application while calling Rd_SetCallBack. It is defined by the application and can be set to 'NULL' when unused.</p> |
| PVoid | <p>The driver passes to the callback function via this function, the very pointer provided by the application while calling Rd_SetCallBack. It is defined by the application and can be set to 'NULL' when unused.</p> |

Description:

This function is used for the driver to set the callback function for the application.

1.4.5 Other Functions

1.4.5.1 Rd_GetLastErrMsg

Prototype Declaration:

```
void WINAPI Rd_GetLastErrMsg(LPSTR lpstrErrMsg)
```

Entry Parameters:

| | |
|-------------|---|
| lpstrErrMsg | Pointer that points to error message character string |
|-------------|---|

Return Values:

Nil

Description:

When there are run-time errors, this function can be used to get the last error message character string.

1.4.5.2 Rd_QueryOp

Prototype Declaration:

```
int WINAPI Rd_QueryOp(int nCh, int nQueryType)
```

Entry Parameters:

| | |
|------------|---|
| nCh | Channel number |
| nQueryType | <p>Query type</p> <p>The following values can be used:</p> <p>enum</p> <p>{</p> <p>RD_QUERY_OPCH = 0, // default return value is 1</p> <p>RD_QUERY_EnPlay = 1, //can play back or not, default value is 1</p> <p>RD_QUERY_EnRecord = 2, ///can record or not, default value is 1</p> <p>RD_QUERY_OpPlay = 3, //data playback status</p> <p>RD_QUERY_OpRecord = 4, //data recording status</p> <p>RD_QUERY_BaudRate = 5, //baud rate for playback, returned baud rate value</p> <p>RD_QUERY_RecordByte = 6, //number of data bytes recorded</p> <p>RD_QUERY_PlayListLen = 7, //length of task queues for playback</p> <p>}</p> |

| | |
|--|---|
| | RD_QUERY_RecListLen = 8, <i>//length of task queues for recording</i> }; |
|--|---|

Return Values:

-1: Failed. Wrong channel number or no such query type

≥0: Return different values based on different nQueryType

| | |
|-------------------|---|
| RD_QUERY_OPCH | 0: Channel is abnormal, unusable 1: Channel is normal |
| RD_QUERY_OpPlay | enum { PLAY_IDLE = 0, <i>//idle</i> PLAY_PLAYING = 1, <i>//replaying</i> PLAY_WAITEND = 2, <i>//waiting for end</i> }; |
| RD_QUERY_OpRecord | enum { REC_IDLE = 0, <i>//idle</i> REC_RECORDING = 1, <i>//recording</i> }; |

Description:

Queries various driver's runtime statuses.

1.4.5.3 Rd_SetUserInfo

Prototype Declaration:

```
int WINAPI Rd_SetUserInfo(char *pcFileName, char cUserInfo[800])
```

Entry Paramaters:

| | |
|-----------------------|------------------|
| PcFileName | File name |
| cUserInfo[800] | User information |

Return Values:

-1: Failed

1: Successful

Description:

Modifies the user-defined field information in file header of the radar data file specified by the parameter pcFileName

1.4.5.4 Rd_GetFileInfo**Prototype Declaration:**

```
int WINAPI Rd_GetFileInfo(char *pcFileName, char *pcStartTime, char *pcEndTime, int *pnCh, char *pcUserInfo)
```

Entry Parameters:

| | |
|-------------|---------------------------|
| PcFileName | File name |
| PcStartTime | Start time, 20byte |
| PcEndTime | End time, 20byte |
| pnCh | Record channel |
| pcUserInfo | User information, 800byte |

Return Values:

-1: Failed

1: Successful

Description:

Gets file header information from the specified file which can be any one of the '.rdd' files stored in the HD (files with data recorded or being recorded)

Chapter 2 Board Configuration

'C:\Radar\Radar.ini' is the default installation directory for the configuration file 'Radar.ini' incorporated in the driver, which can be compiled manually or by the configuration program 'ShConfig.exe'.

2.1 Manual Compilation

Below is the content of a typical configuration file 'Radar.ini'.

[SystemConfig] *//system configuration subsection, used to set system parameters*

nTotalBoards=1 *// total number of boards installed in system*

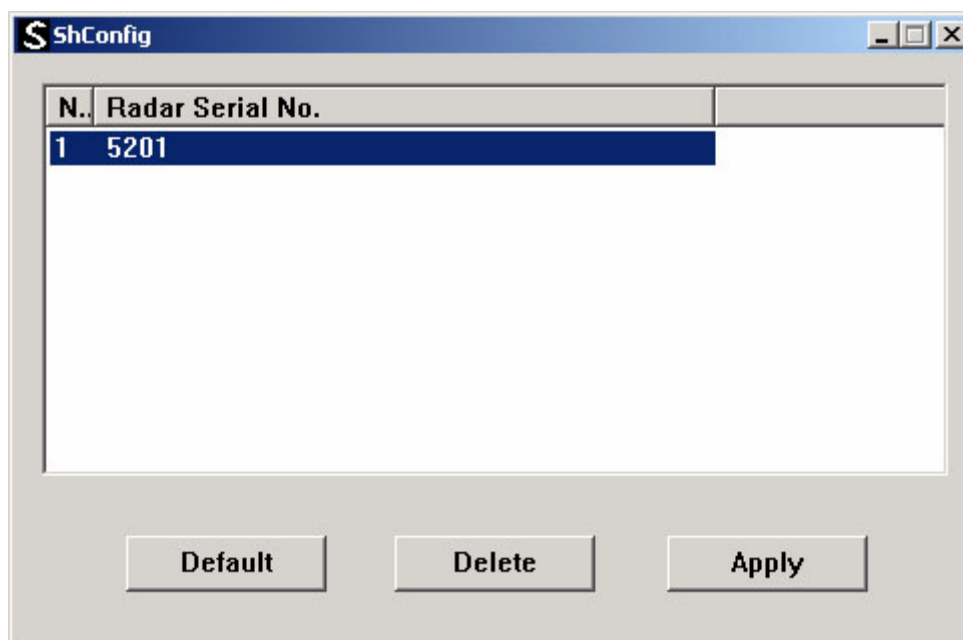
[BoardId=0] *//board configuration subsection, used to set board parameters*

BoardSerialNumber=5201 *// board serial number, available in the label*

PlaybackBaudrate=19200, 19200, 19200, 19200 *//default baud rate for data playback on each channel*

2.2 Compilation by Configuration Program

Run the configuration program 'ShConfig.exe' as shown in the figure below:



Click on 'Default' to obtain the board serial number, and then click 'Apply' to save corresponding configuration information into the configuration file (If you want to remove unused radar boards, just highlight them and click 'remove').

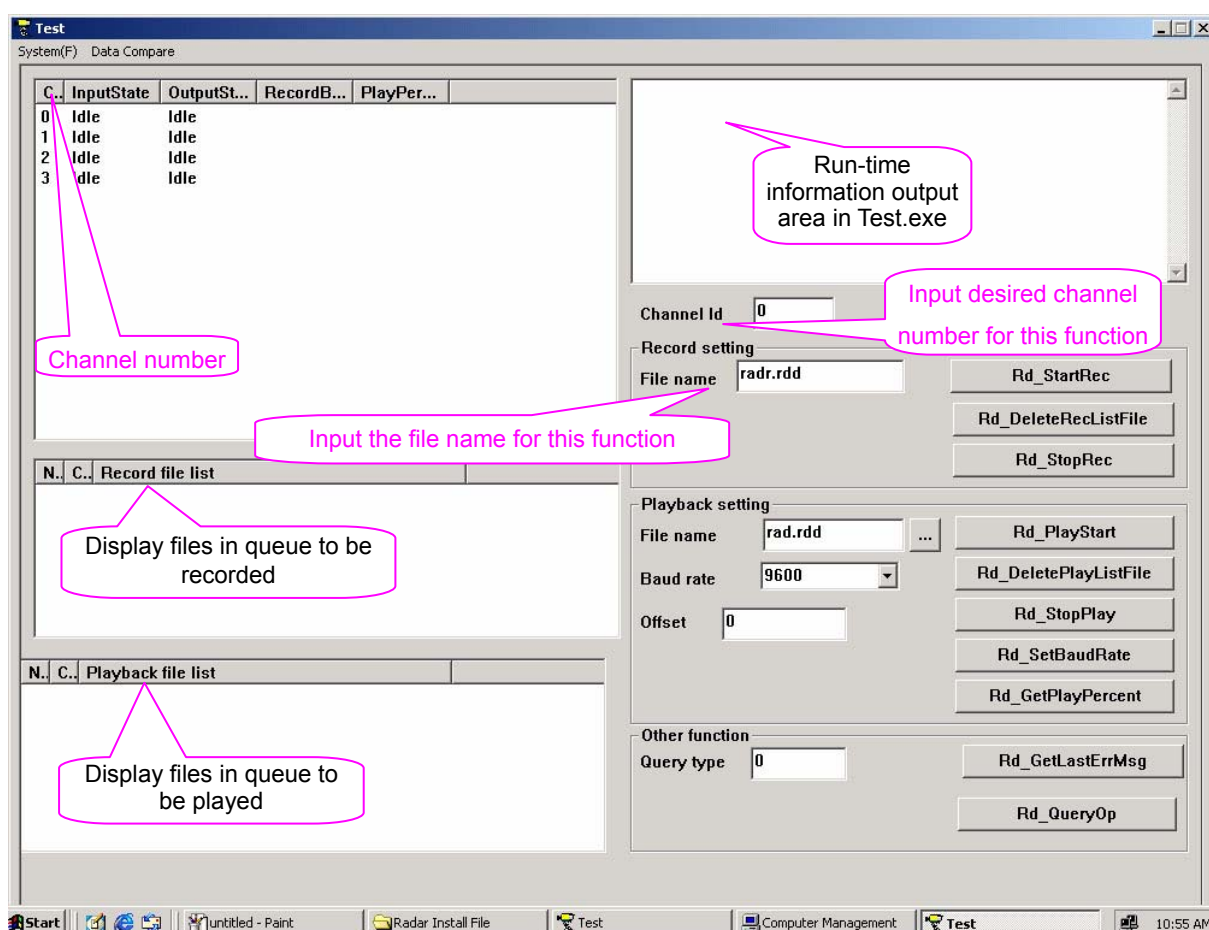
2.3 Channel Numbering Rule

Both the boards in the system and the channels on the board are numbered beginning from 0. Correspondingly channel numbering for the application also starts from 0. As a result, the relationship among channel number for the application (ch), board number (x) and on-board channel number (i) can be expressed by the formula ' $ch = x*4+i$ '. Notice each channel number appearing in the API functions stated in this manual implies the channel number for the application.

Chapter 3 Demo Program

Test.exe, a function test program included with the driver, is meant for testing API functions. Each function has a corresponding button in this program. Below is a simple instruction for use.

- (1) Connect serial lines to on-board interfaces;
- (2) Run the configuration program 'ShConfig.exe' to well configure the radar data recording board;
- (3) Run Test.exe. And the user interface upon successful run of Test.exe is shown as follows.



All you need to do is to input parameters with respect to the function to be tested, and click on the corresponding button.

Appendix A Technical/sales Support

Thank you for choosing Synway. Please contact us should you have any inquiry regarding our products. We shall do our best to help you.

Headquarters

Hangzhou Synway Information Engineering Co., Ltd
<http://www.synway.net/>
11/12/F, QiLun Building, No.385, Wen'er Road, Hangzhou,
P.R.China, 310012
Tel: +86-571-88860561
Fax: +86-571-88850923

Technical Support

Tel: +86-571-88864579
Mobile: +86-13735549651
Email: techsupport@sanhuid.com
Email: techsupport@synway.net
MSN: scycindy_sh@hotmail.com

Sales Department

Tel: +86-571-88860561
Tel: +86-571-88864579
Fax: +86-571-88850923
Email: sales@synway.net