GRPLTLibrary User Manual

Version: 1.0

Release Date: 2025-04-22

Shenzhen Goodix Technology Co., Ltd.

Trademarks and Permissions

GOODiX and other Goodix trademarks are trademarks of Shenzhen Goodix Technology Co., Ltd. All other trademarks and trade names mentioned in this document are the property of their respective holders.

Disclaimer

Information contained in this document is intended for your convenience only and is subject to change without prior notice. It is your responsibility to ensure its application complies with technical specifications.

Shenzhen Goodix Technology Co., Ltd. (hereafter referred to as “Goodix”) makes no representation or guarantee for this information, express or implied, oral or written, statutory or otherwise, including but not limited to representation or guarantee for its application, quality, performance, merchantability or fitness for a particular purpose. Goodix shall assume no responsibility for this information and relevant consequences arising out of the use of such information.

Without written consent of Goodix, it is prohibited to use Goodix products as critical components in any life support system. Under the protection of Goodix intellectual property rights, no license may be transferred implicitly or by any other means.

Shenzhen Goodix Technology Co., Ltd.

Headquarters: 26F, Goodix Headquarters, No.1 Meikang Road, Futian District, Shenzhen, China

TEL: +86-755-33338828 Zip Code: 518000

Website: www.goodix.com

Preface

Purpose

This document introduces GRPLTLibrary (a secondary development library) of GRPLT (a mass production tool) for GR5xx SoCs, including the encapsulated APIs, supported variables, and callback functions, as well as the specific application method, to help users quickly get started with secondary development of GRPLT.

Audience

This document is intended for:

- Device user
- Developer
- Test engineer
- Technical support engineer

Release Notes

This document is the initial release of *GRPLTLibrary User Manual*, corresponding to *GRPLTLibrary.dll*.

Revision History

Version	Date	Description
1.0	2025-04-22	Initial release

Contents

Preface I

1 Introduction 1

2 APIs 2

2.1 GR5_Init 2
2.2 GR5_UnInit 3
2.3 GR5_SetCallbackFunction 4
2.4 GR5_SetValue 5
2.5 GR5_SetValueByID 6
2.6 GR5_GetValue 7
2.7 GR5_GetTestStatus 7
2.8 GR5_GetTestResults 8
2.9 GR5_SetReadTransparent 9
2.10 GR5_WriteData 10
2.11 GR5_SetEncryptBinFile 11
2.12 GR5_InitSetting 12

3 Callback Functions 14

3.1 Declaration 14
3.2 List of Callback Functions 14

4 Variables 16

4.1 List of Variables 16

5 Operational Instructions 20

5.1 Operating Environment 20
5.2 Application Method 20

6 Appendix: TestConfig Structures 21

1 Introduction

The GRPLTLibrary secondary development library (hereinafter referred to as **GRPLTLibrary**) provides a series of APIs that help users quickly get started with secondary development of GRPLT (a mass production tool) for GR5xx SoCs, to better meet application requirements.

GRPLTLibrary encapsulates the provided APIs into a DLL library file, *GRPLTLibrary.dll*, which supports secondary development applications in C++ and C#.

The system component diagram for secondary application development using GRPLTLibrary is as follows:

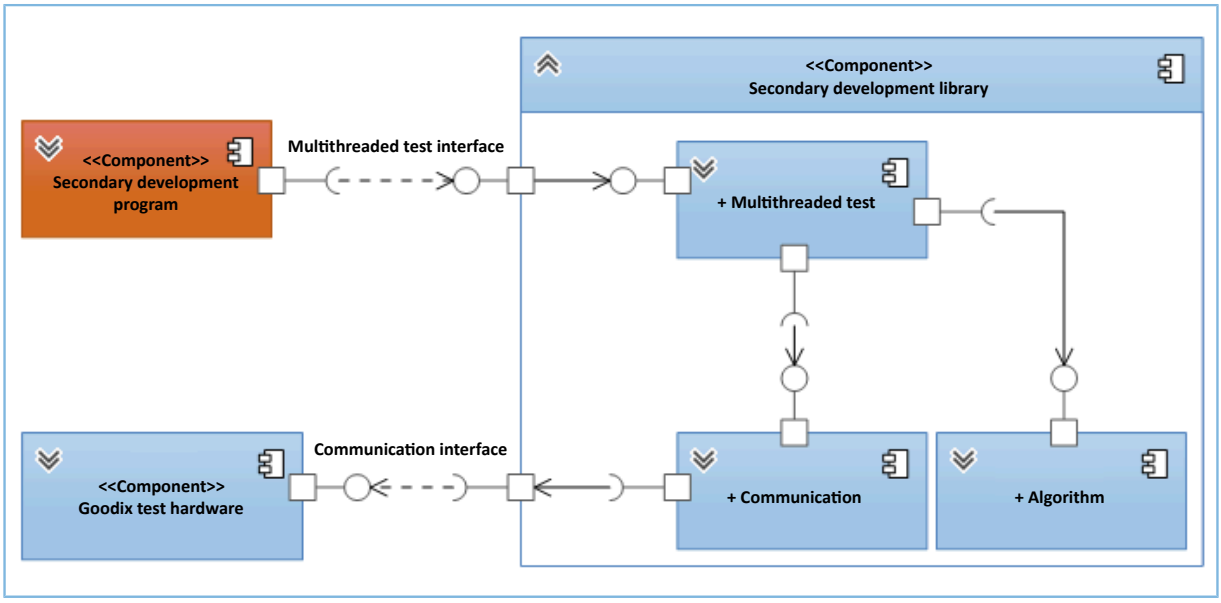

Figure 1-1 System component diagram

2 APIs

The APIs encapsulated in GRPLTLibrary are listed below:

Table 2-1 APIs

No.	Function Name	Functional Description
1	GR5_Init	Initialize GRPLTLibrary.
2	GR5_UnInit	Deinitialize GRPLTLibrary to release the library resources.
3	GR5_SetCallbackFunction	Register callback functions.
4	GR5_SetValue	Set variable values.
5	GR5_SetValueByID	Set parameters according to the device ID.
6	GR5_GetValue	Get variable values.
7	GR5_GetTestStatus	Get test states.
8	GR5_GetTestResults	Get test results.
9	GR5_SetReadTransparent	Set transparent data read.
10	GR5_WriteData	Send a command (maximum length: 1024) to the DUT.
11	GR5_SetEncryptBinFile	Encrypt a BIN file as EBIN file.
12	GR5_InitSetting	Initialize configurations.

2.1 GR5_Init

GR5_Init API is used to initialize GRPLTLibrary.

Declaration:

- C++ declaration

```
extern "C" int PASCAL EXPORT GR5_Init(char * registerCode, char * orderFileName);
```

- C# declaration

```
[DllImport("GRPLTLibrary.dll", CharSet = CharSet.Ansi, EntryPoint = "GR5_Init",
 ExactSpelling = false, CallingConvention = CallingConvention.StdCall)]
public static extern int GR5_Init(string registerCode, string orderFilePath);
```

Description:

Table 2-2 GR5_Init API

Function Prototype	int GR5_Init(char * registerCode, char * orderFileName) (taking C++ declaration as an example)
Input Parameter	• registerCode: registered character strings; can be any value except NULL • orderFileName: configuration file name; the configuration file should be stored in the same directory as the DLL file.

	Note: For the C# function, the parameter orderFileName is orderFilepath (the configuration file path).
Output Parameter	None
Return Value	Return an integer: • 0: Operation succeeded. • Others: Operation failed. This value is the error code (refer to Table 2-3).

Table 2-3 Error code definition

Error Code	Description
1	Not initialized
2	Parameter error
3	Invalid operation
4	Invalid device ID

Example:

```
int retCode = GRPLTLibrary.GR5_Init("testcode", "ORDER_20151124151045.order");
if (0 ! = retCode)
{
   MessageBox.Show(string.Format("Error: {0}", retCode));
}
```

2.2 GR5_UnInit

GR5_UnInit API is used to deinitialize GRPLTLibrary to release library resources.

Declaration:

- C++ declaration

```
extern "C" int PASCAL EXPORT GR5_UnInit();
```

- C# declaration

```
[DllImport("GRPLTLibrary.dll", CharSet = CharSet.Ansi, EntryPoint = "GR5_UnInit",
 ExactSpelling = false, CallingConvention = CallingConvention.StdCall)]
public static extern int GR5_UnInit();
```

Description:

Table 2-4 GR5_UnInit API

Function Prototype	int GR5_UnInit() (taking C++ declaration as an example)
Input Parameter	None
Output Parameter	None

Return Value	Return an integer: • 0: Operation succeeded. • Others: Operation failed. This value is the error code (refer to Table 2-3).

Example:

```
GRPLTLibrary.GR5_UnInit();
```

2.3 GR5_SetCallbackFunction

GR5_SetCallbackFunction API is used to register callback functions.

Declaration:

- C++ declaration

```
typedef int (PASCAL *GR5_CALLBACKFUNC)(int value, char * buf, int len);
extern "C" int PASCAL EXPORT GR5_SetCallbackFunction(char * functionName, GR5_CALLBACKFUNC
 func);
```

- C# declaration
  - Method 1: The second parameter of the callback function is a string.

```
public delegate int StringCallbackDelegate(int value, string str, int len);
[DllImport("GRPLTLibrary.dll", CharSet = CharSet.Ansi, EntryPoint =
 "GR5_SetCallbackFunction", ExactSpelling = false, CallingConvention =
 CallingConvention.StdCall)]
public static extern int GR5_SetStringCallbackFunction(string functionName,
 StringCallbackDelegate func);
```

  - Method 2: The second parameter of the callback function is an array.

```
public delegate int ArrayCallbackDelegate(int value, [MarshalAs(UnmanagedType.LPArray,
 SizeConst = 10000)]byte[] buf, int len);
[DllImport("GRPLTLibrary.dll", CharSet = CharSet.Ansi, EntryPoint =
 "GR5_SetCallbackFunction", ExactSpelling = false, CallingConvention =
 CallingConvention.StdCall)]
public static extern int GR5_SetArrayCallbackFunction(string functionName,
 ArrayCallbackDelegate func);
```

Description:

Table 2-5 GR5_SetCallbackFunction API

Function Prototype	int GR5_SetCallbackFunction(char * functionName, GR5_CALLBACKFUNC func) (taking C++ declaration as an example)
Input Parameter	• functionName: callback function name • func: callback function
Output Parameter	None
Return Value	Return an integer:

	• 0: Operation succeeded. • Others: Operation failed. This value is the error code (refer to Table 2-3).

Example:

```
private GRPLTLibrary.StringCallbackDelegate deviceChangedCallback;
deviceChangedCallback = new
 GRPLTLibrary.StringCallbackDelegate(DeviceChangedCallBackFunction);
retCode = GRPLTLibrary.GR5_SetStringCallbackFunction("DeviceChanged",
 deviceChangedCallback);
if (0 ! = retCode)
{
    MessageBox.Show(string.Format("Error: {0}", retCode));
}
private int DeviceChangedCallBackFunction(int value, string str, int len)
{
    this.Dispatcher.Invoke(new Action(() =>
    {
        int num = 10;
        int[] list = new int[num];

        GRPLTLibrary.GR5_GetDeviceList(list, ref num);

        textBoxLog.AppendText("Device Changed\r\n");
        for (int i = 0; i < num; i++)
        {
            textBoxLog.AppendText(string.Format("Dev[{0}]: {1:X8}\r\n", i,list[i]));
        }
        textBoxLog.ScrollToEnd();
    }));
    return 0;
}
```

2.4 GR5_SetValue

GR5_SetValue API is used to set variable values.

Declaration:

- C++ declaration

```
extern "C" int PASCAL EXPORT GR5_SetValue(char * variableName, char * value);
```

- C# declaration

```
[DllImport("GRPLTLibrary.dll", CharSet = CharSet.Ansi, EntryPoint = "GR5_SetValue",
 ExactSpelling = false, CallingConvention = CallingConvention.StdCall)]
public static extern int GR5_SetValue(string variableName, string value);
```

Description:

Table 2-6 GR5_SetValue API

Function Prototype	int GR5_SetValue(char * variableName, char * value) (taking C++ declaration as an example)
Input Parameter	• variableName: variable name

	• value: variable value
Output Parameter	None
Return Value	Return an integer: • 0: Operation succeeded. • Others: Operation failed. This value is the error code (refer to Table 2-3).

Example:

```
int retCode = GRPLTLibrary.GR5_SetValue("ShowDebugWindow", "True");
if (0 ! = retCode)
{
    MessageBox.Show(string.Format("Error: {0}", retCode));
}
```

2.5 GR5_SetValueByID

GR5_SetValueByID API is used to set variable values according to the device ID.

Declaration:

- C++ declaration

```
extern "C" int PASCAL EXPORT GR5_SetValueByID(int devID, char* variableName, char* value);
```

- C# declaration

```
[DllImport("GRPLTLibrary.dll", CharSet = CharSet.Ansi, EntryPoint = "GR5_SetValueByID",
 ExactSpelling = false, CallingConvention = CallingConvention.StdCall)]
public static extern int GR5_SetValueByID(int devID, string variableName, string value);
```

Description:

Table 2-7 GR5_SetValueByID API

Function Prototype	GR5_SetValueByID(int devID, char* variableName, char* value) (taking C++ declaration as an example)
Input Parameter	• devID: device ID • variableName: variable name • value: variable value
Output Parameter	None
Return Value	Return an integer: • 0: Operation succeeded. • Others: Operation failed. This value is the error code (refer to Table 2-3).

Example:

```
int retCode = GRPLTLibrary.GR5_SetValueByID(devID, "SetBaudRate", "115200");
if (0 ! = retCode)
{
```

```
    MessageBox.Show(string.Format("Error: {0}", retCode));
}
```

2.6 GR5_GetValue

GR5_GetValue API is used to get variable values.

Declaration:

- C++ declaration

```
extern "C" int PASCAL EXPORT GR5_GetValue(char * variableName, char * value, int * length);
```

- C# declaration

```
[DllImport("GRPLTLibrary.dll", CharSet = CharSet.Ansi, EntryPoint = "GR5_GetValue",
 ExactSpelling = false, CallingConvention = CallingConvention.StdCall)]
public static extern int GR5_GetValue(string variableName, StringBuilder value, ref int
 length);
```

Description:

Table 2-8 GetValue API

Function Prototype	int GR5_GetValue(char * variableName, char * value, int * length) (taking C++ declaration as an example)
Input Parameter	• variableName: variable name • length: input array length
Output Parameter	• value: variable value • length: output data length
Return Value	Return an integer: • 0: Operation succeeded. • Others: Operation failed. This value is the error code (refer to Table 2-3).

Example:

```
int length = 255;
StringBuilder szStr = new StringBuilder(length);
int retCode = GRPLTLibrary.GR5_GetValue("LibVersion", szStr, ref length);
if (0 ! = retCode)
{
    MessageBox.Show(string.Format("Error: {0}", retCode));
}
```

2.7 GR5_GetTestStatus

GR5_GetTestStatus API is used to get test states.

Declaration:

- C++ declaration

```
extern "C" int PASCAL EXPORT GR5_GetTestStatus(int devID, int * testStatus);
```

- C# declaration

```
[DllImport("GRPLTLibrary.dll", CharSet = CharSet.Ansi, EntryPoint = "GR5_GetTestStatus",
 ExactSpelling = false, CallingConvention = CallingConvention.StdCall)]
public static extern int GR5_GetTestStatus(int devID, ref int testStatus);
```

Description:

Table 2-9 GR5_GetTestStatus API

Function Prototype	int GR5_GetTestStatus(int devID, int * testStatus) (taking C++ declaration as an example)
Input Parameter	devID: device ID
Output Parameter	testStatus: test state • 1: ready • 2: testing • 3: finished • 4: idle
Return Value	Return an integer: • 0: Operation succeeded. • Others: Operation failed. This value is the error code (refer to Table 2-3).

Example:

```
int testStatus = 0;
int devID = value;
GRPLTLibrary.GR5_GetTestStatus(value, ref testStatus);
switch (testStatus)
{
    case 1:
        MessageBox.Show("Ready");
        break;
    case 2:
        MessageBox.Show("Testing");
        break;
    case 3:
        MessageBox.Show("Finished");
        break;
    case 4:
        MessageBox.Show("Idle");
        break;
    default:
        break;
}
```

2.8 GR5_GetTestResults

GR5_GetTestResults API is used to get test results.

Declaration:

- C++ declaration

```
extern "C" int PASCAL EXPORT GR5_GetTestResults(int devID, char * testResults,int * length);
```

- C# declaration

```
[DllImport("GRPLTLibrary.dll", CharSet = CharSet.Ansi, EntryPoint = "GR5_GetTestResults",
 ExactSpelling = false, CallingConvention = CallingConvention.StdCall)]
public static extern int GR5_GetTestResults(int devID, StringBuilder testResults, ref int
 length);
```

Description:

Table 2-10 GR5_GetTestResults API

Function Prototype	int GR5_GetTestResults(int devID, char * testResults,int * length) (taking C++ declaration as an example)
Input Parameter	• devID: device ID • length: input array length
Output Parameter	• testResults: test result (example: “ModuleID=00000001|PinOS=Pass|PixelOSNum=0|StableCurrent=10000|SleepingCurrent=10|ErrorCode=0x00”) • length: output data length
Return Value	Return an integer: • 0: Operation succeeded. • Others: Operation failed. This value is the error code (refer to Table 2-3).

Example:

```
int length = 1024;
int devID = value;
StringBuilder szStr = new StringBuilder(length);
GRPLTLibrary.GR5_GetTestResults(devID, szStr, ref length);
MessageBox.Show(szStr.ToString());
```

2.9 GR5_SetReadTransparent

GR5_SetReadTransparent API is used to set transparent data read.

Declaration:

- C++ declaration

```
extern "C" int PASCAL EXPORT GR5_SetReadTransparent(int devID, int * readStatus);
```

- C# declaration

```
[DllImport("GRPLTLibrary.dll", CharSet = CharSet.Ansi, EntryPoint = " GR5_SetReadTransparent
 ", ExactSpelling = false, CallingConvention = CallingConvention.StdCall)]
```

```
public static extern int GR5_SetReadTransparent (int devID, ref int readStatus);
```

Description:

Table 2-11 GR5_SetReadTransparent API

Function Prototype	int GR5_SetReadTransparent(int devID, int * readStatus) (taking C++ declaration as an example)
Input Parameter	• devID: device ID • readStatus: data read mode; 0: default mode; 1: transparent data read mode
Output Parameter	None
Return Value	Return an integer: • 0: Operation succeeded. • Others: Operation failed. This value is the error code (refer to Table 2-3).

Example:

```
int readStatus = 1;
int devID = value;
GRPLTLibrary.GR5_SetReadTransparent(devID, ref readStatus);
```

2.10 GR5_WriteData

GR5_WriteData API is used to send a command (maximum length: 1024) to the DUT.

Declaration:

- C++ declaration

```
extern "C" int PASCAL EXPORT GR5_WriteData(int devID, unsigned char *writeData, int*
 length);
```

- C# declaration

```
[DllImport("GRPLTLibrary.dll", CharSet = CharSet.Ansi, EntryPoint = " GR5_WriteData ",
 ExactSpelling = false, CallingConvention = CallingConvention.StdCall)]
public static extern int GR5_WriteData (int devID, byte[] writeData, ref int length);
```

Description:

Table 2-12 GR5_WriteData API

Function Prototype	int GR5_WriteData(int devID, unsigned char *writeData, int* length) (taking C++ declaration as an example)
Input Parameter	• devID: device ID • writeData: input command protocol data array • length: input array length (max.: 1024)
Output Parameter	None

Return Value	Return an integer: • 0: Operation succeeded. • Others: Operation failed. This value is the error code (refer to Table 2-3).

Example:

```
byte[] strIn = {0x44,0x47,0x01,0x00,0x00,0x00,0x01,0x00};
int datalen = 8;
int devid = value;
GRPLTLibrary.GR5_WriteData(devid, strIn, ref datalen);
```

2.11 GR5_SetEncryptBinFile

GR5_SetEncryptBinFile API is used to encrypt a BIN file as EBIN file.

Declaration:

- C++ declaration

```
extern "C" int PASCAL EXPORT GR5_SetEncryptBinFile (char* pchFilePath, char* pchMacArray);
```

- C# declaration

```
[DllImport("GRPLTLibrary.dll", CharSet = CharSet.Ansi, EntryPoint = " GR5_SetEncryptBinFile
 ", ExactSpelling = false, CallingConvention = CallingConvention.StdCall)]
public static extern int GR5_SetEncryptBinFile (string strFilePath, string strMacArray);
```

Description:

Table 2-13 GR5_SetEncryptBinFile API

Function Prototype	GR5_SetEncryptBinFile (char* pchFilePath, char* pchMacArray) (taking C++ declaration as an example)
Input Parameter	• pchFilePath: absolute path of the BIN file to be encrypted • pchMacArray: null string (“”) For the C# function, the parameters are strFilePath and strMacArray.
Output Parameter	None
Return Value	Return an integer: • 0: Operation succeeded. • Others: Operation failed. This value is the error code (refer to Table 2-3).

Example:

```
int retCode = GRPLTLibrary.GR5_SetEncryptBinFile ("D:\\BLE\\Customer\\USER_TEST.bin",
 "F8BC126BF9F2");
//If there is no need to restrict the test PC, the MAC address can be passed as an empty
 string.
//int retCode = GRPLTLibrary.GR5_SetEncryptBinFile ("D:\\BLE\\Customer\\USER_TEST.bin", "");
if (0 ! = retCode)
```

```
{
    MessageBox.Show(string.Format("Error: {0}", retCode));
}
```

2.12 GR5_InitSetting

GR5_InitSetting API is used to initialize configurations. This API shall be called before calling the GR5_Init API; otherwise, configuration may fail.

Declaration:

- C++ declaration

```
extern "C" int PASCAL EXPORT GR5_InitSetting (stTestConfig* pConfig);
```

- C# declaration

```
[DllImport("GRPLTLibrary.dll", CharSet = CharSet.Ansi, EntryPoint = " GR5_InitSetting ",
 ExactSpelling = false, CallingConvention = CallingConvention.StdCall)]
public static extern int GR5_InitSetting (ref TestConfig testConfig);
```

Description:

Table 2-14 GR5_InitSetting API

Function Prototype	GR5_InitSetting (stTestConfig* pConfig) (taking C++ declaration as an example)
Input Parameter	pConfig: Refer to “Chapter 6 Appendix: TestConfig Structures”. **Note**: For the C# function, the parameter is testConfig.
Output Parameter	None
Return Value	Return an integer: • 0: Operation succeeded. • Others: Operation failed. This value is the error code (refer to Table 2-15).

Table 2-15 Error code definition

Error Code	Description
2	DUT reset time being 0 error
6	Empty configuration
7	User start-up configuration error
8	Bluetooth configuration error
9	Xtal and RSSI configuration error
10	NVDS configuration error
11	Flash configuration error

Error Code	Description
12	eFuse configuration error
13	User firmware configuration error

Example:

```
GRPLTLibrary.TestConfig testConfig = new GRPLTLibrary.TestConfig;
testConfig.rstTime = 500;
testConfig.testSettingConfig.rssiLimit = -100;
testConfig.testSettingConfig.xtalSystickError = 60;
//Set GPIO_2 as the calibration port.
testConfig.testSettingConfig.xtalGPIO.type = 0;
testConfig.testSettingConfig.xtalGPIO.pin = 2;

int retCode = GRPLTLibrary.GR5_InitSetting(ref testConfig);
if (0 ! = retCode)
{
    MessageBox.Show(string.Format("Error: {0}", retCode));
}
```

3 Callback Functions

Callback functions that are triggered under certain conditions can be configured using the GR5_SetCallbackFunction API in GRPLTLibrary.

3.1 Declaration

- C++ declaration

```
typedef int (PASCAL *GR5_CALLBACKFUNC)(int value, char * buf, int len);
```

- C# declaration
  - Method 1: The second parameter passed in is a string.

```
public delegate int StringCallbackDelegate(int value, string str, int len);
```

  - Method 2: The second parameter passed in is an array.

```
public delegate int ArrayCallbackDelegate(int value, [MarshalAs(UnmanagedType.LPArray,
 SizeConst = 10000)]byte[] buf, int len);
```

3.2 List of Callback Functions

The callback functions supported by GRPLTLibrary are listed as follows:

Table 3-1 List of callback functions

Function Name	Triggering Condition	Parameter		
		Value	Buf	Len
TestStarted	Test starts.	Device ID	None	None
TestFinished	Test ends.	Device ID	The test result (string) is returned in key-value pair format, and the corresponding values can be parsed using keys such as ChipUID/Flash UID, for example: “ChipUID=18030E411500C518017A40033D8A5326|BdAddress=|FLASH UID=7492090C04A5D109|Error=0x00”	Result length
TestDetailInfor	Receive test details, such as test state and result.	Device ID	Test details (string)	String length
GetBLEMacKey	After setting MAC and key	0	None	None
CustomOperation	Custom operation; after all DUT test items are finished and before the chip is powered off	Device ID	Fixed string “TestItem|Finish”	Data length

Function Name	Triggering Condition	Parameter		
		Value	Buf	Len
DutTestOperation	Custom operation; before user firmware programming, the user program can directly interact with the DUT test firmware.	Device ID	Fixed string “DUT Test Operation”	Data length
ReadTransparent	Transparent data read API	Device ID	Data received from the chip	Data length
WriteUserBefore	Before writing to the user area; return the existing data in the user area of the chip.	Device ID	Data (string) in the user area	String length
WriteUserFinish	After writing to the user area; return the existing data in the user area of the chip.	Device ID	Data (string) in the user area	String length
WriteFlashData	Write Flash BIN parameters.	Device ID	Flash parameters (Bin Num (1 byte) + Bin Addr (4 bytes, little endian) + Bin Path (255 bytes))	None
AllTestsFinished	All DUT tests end.	0	None	None

4 Variables

Certain variables in GRPLT can be read/written using the GR5_SetValue, GR5_GetValue, and GR5_SetValueByID APIs in GRPLTLibrary, to get the test information or configure the test conditions.

4.1 List of Variables

GRPLTLibrary supports reading/writing the following variables:

Table 4-1 Variables supported by GR5_SetValue and GR5_GetValue APIs

Variable Name	Description	Value	Permission
LibVersion	Library version	String	Read only
FWVersion	User firmware version. If multiple user firmware files are imported in the work order, each user firmware version returned will be separated by a comma (,).	String	Read only
LastError	Description of the last error	String	Read only
ShowDebugWindow	Flag to display the debug window	True or False	Read/Write
DebugWindowLogEnable	Flag to display log information in the debug window	True or False	Read/Write
LogDir	Log storage directory	String	Read/Write
DataDir	Data storage directory	String	Read/Write
SavaDataType	Data storage type	0: Do not store the data. 1: Store all the data. 2: Store the data for failed operations only.	Read/Write
StartTest	Set whether to manually start testing for a port.	0xffff (manually start testing for all 16 ports.) Each port corresponds to 1 bit: 1: Enable 0: Disable	Write only
CtrlPWM	Enable/Disable GU to generate PWM waveforms.	1: Enable 0: Disable	Write only
LogFileName	Log file name	String	Read/Write
DataFileName	Data file name	String	Read/Write
RecvQrCodeEnable	Enable/Disable the QR code acquiring functionality.	1: Enable 0: Disable	Write only
SetQRCode	A special item for which a QR code must be entered before testing	For example: “25|0199055PSHFB35X517C231101HX”,	Write only

Variable Name	Description	Value	Permission
		where the first two digits (such as “25”) represent the length of the QR code in decimal, with a maximum of 32; in the main part of the string, the first two digits indicate the corresponding UART port number, for example: 01 stands for UART1. Support entering multiple QR codes at the same time, with each QR code separated by a “|”.	
SetMacKeyEnable	Enable/Disable getting user MAC and key.	1: Enable 0: Disable	Read/Write
SetMac	Set MAC by users.	16 groups of MAC addresses are passed in at once; if any is missing, fill it with zero. The MAC addresses should be converted from hexadecimal to character format: 16 bytes for each group of ports, totally 16*16 bytes. For the detailed port format, refer to Table 4-3.	Read/Write
SetKey	Set the key by users.	16 groups of keys are passed in at once; if any is missing, fill it with zero. The keys should be converted from hexadecimal to character format: 36 bytes for each group of ports, totally 16*36 bytes. For the detailed port format, refer to Table 4-3.	Read/Write
GetMac	Get MAC by users.	16 groups of MAC addresses are output at once; if any is missing, fill it with zero. The MAC addresses should be converted from hexadecimal to character format: 19 bytes for each group of ports, totally 16*19 bytes. For the detailed port format, refer to Table 4-3.	Read only
GetKey	Get the key by users.	16 groups of keys are output at once; if any is missing, fill it with zero. The keys should be converted from hexadecimal to character format: 39 bytes for each group of ports, totally 16*39 bytes. For the detailed port format, refer to Table 4-3.	Read only

Variable Name	Description	Value	Permission
SetUserRegionEnable	Enable/Disable writing to the user area in eFuse.	1: Enable 0: Disable	Write only
SetUserRegionValue	Write data to the user area.	Data format: length (the length of the data to be written; fixed two-digit decimal; max.: 32), UART (UART serial number to be written; decimal; fixed length of 3 digits starting with 1), and data to be written (hexadecimal). If there are multiple data segments, separate each segment with a comma ",". The write length of all segments must be consistent with the initially specified length. Writing different lengths of data to different UART ports is not supported. For example: 16,104**0102030405060708091011121314151A** 16: number of bytes to be written to the user area 104: UART4 The **bold** part represents a 16-byte data string to be written to the user area.	Write only
SetMergeBinPath	Set the absolute file path of the MergeBin.	The absolute path of MergeBin must be set before calling the Init API; otherwise, the firmware cannot be got.	Write only

Table 4-2 Variables supported by GR5_SetValueByID API

Variable Name	Description	Value	Permission
SetBaudRate	Set the baud rate.	String, such as “115200”	Write only

Port data format:

Table 4-3 Port data format

Portx	Byte	0	1	2	3	N-1	N	Port_H is the higher decimal digits.
		x	Port_H	Port_L	MAC0	MAC5	","	Port_L is the lower decimal digits (1 to 16).
		x	Port_H	Port_L	Key0	Key15	","	The last byte is a comma.
								The first byte is 1’.

Port data example:

```
GRPLTLibrary.GR5_SetValue("SetMac", "101010203040506,102010203040506,104010203040506,");
GRPLTLibrary.GR5_SetValue("SetKey", "10101020304050607080910111213141516,");
GRPLTLibrary.GR5_SetValue("SetUserRegionValue",
 "16,1010A020304050607080910111213141 51A,104010203040506070809101112131415 1A,
                          107010203040506070809101112131415 1B");
```

5 Operational Instructions

This chapter gives a detailed explanation of the environmental requirement and specific application method for GRPLTLibrary.

5.1 Operating Environment

Environmental requirements for GRPLTLibrary are as follows:

Table 5-1 Hardware environment

Name	Description
CPU	1.2 GHz and faster
Memory	1 GB and larger
USB	Power output: ≥ 500 mA; ripple: ≤ 100 mV
Test hardware	Goodix hardware (PLT) for module mass production test

Table 5-2 Software environment

Name	Description
Operating system	Windows 7/Windows 10/Windows 11
IDE	VS2013

5.2 Application Method

During secondary development with GRPLT, users can use GRPLTLibrary according to the following steps (or refer to the example code for secondary development):

1. Copy the library file *GRPLTLibrary.dll* and the configuration file *DeParams.xml* into the directory where the executable file (*.exe*) of the development tool is located.
2. Add the library function declarations to the secondary development application code (refer to "Chapter 2 APIs").
3. Call the GR5_Init API to initialize GRPLTLibrary. The orderFileName parameter of this API should be the *.order* work order file generated by *OrderSetting.exe*.
4. Write callback functions (or delegates) and call the GR5_SetCallbackFunction API to set callback functions that are triggered under certain conditions.
5. According to actual needs, call other functions in the library.
6. At the end of the program, call the GR5_UnInit API to release library resources.

6 Appendix: TestConfig Structures

- The stUartConfig structure is used to configure UART. It is defined as follows:

```
typedef struct
{
    /// <summary>
    /// 0 for NOPARITY
    /// 1 for ODDPARITY
    /// 2 for EVENPARITY
    /// 3 for MARKPARITY
    /// </summary>
    uint8_t nParity;
    /// <summary>
    /// 0 for ONE_STOPBIT
    /// 1 for ONE5_STOPBITS
    /// 2 for TWO_STOPBITS
    /// </summary>
    uint8_t nStopBits;
    /// <summary>
    /// Uart ByteSize, 4-8
    /// </summary>
    uint8_t nByteSize;
    /// <summary>
    /// Uart BaudRate
    /// </summary>
    uint32_t nBaudRate;
}stUartConfig;
```

- The stIOConfig structure is used to configure I/O pins. It is defined as follows:

```
typedef struct
{
    /// <summary>
    /// 0 for Normal
    /// 1 for AON
    /// 2 for MSIO
    /// </summary>
    uint8_t nIOType;
    /// <summary>
    /// IO PIN
    /// </summary>
    uint8_t nIOPin;
    /// <summary>
    /// IO Mux
    /// </summary>
    uint8_t nMux;
}stIOConfig;
```

- The stCheckCustomConfig structure is used to configure checking information. It is defined as follows:

```
typedef struct
{
    /// <summary>
    /// 1/0, 1 for enable checking whether app can be launched after burning
    /// </summary>
    uint8_t nEnable;
    stUartConfig uartConfig;
```

```
}stCheckCustomConfig;
```

- The stBDAddressConfig structure is used to configure Bluetooth addresses. It is defined as follows:

```
typedef struct
{
    /// <summary>
    /// 1/0, 1 for enable writing BD Address to DUT
    /// </summary>
    uint8_t nEnable;
    /// <summary>
    /// 1/0, 1 for enable writing BD Address to NVDS, only work when write BD Address
        is enabled
    /// </summary>
    uint8_t nWriteInNVDS;
    /// <summary>
    /// 0 for Range mode, 1 for from Txt mode
    /// </summary>
    uint8_t nAddressMode;
    /// <summary>
    /// 1/0, 1 for enable checking address duplicate in txt
    /// </summary>
    uint8_t nCheckFileDuplicate;
    /// <summary>
    /// For Range mode, the string format should be like 23:34:56:78:90:AB
    /// </summary>
    char startAddr[18];
    /// <summary>
    /// For Range mode, the string format should be like 23:34:56:78:90:AB
    /// </summary>
    char endAddr[18];
    /// <summary>
    /// For Txt mode, the absolute file path
    /// </summary>
    char filePath[GRPLT_MAX_PATH_LEN];
}stBDAddressConfig;
```

- The stTestSettingConfig structure is used to configure test parameters. It is defined as follows:

```
typedef struct
{
    /// <summary>
    /// RSSI limit(dBm), RSSI should ble >= this
    /// </summary>
    int iRSSILimit;
    /// <summary>
    /// Xtal sys errors, recommended value 60
    /// </summary>
    uint8_t nXtalSystickErrors;
    /// <summary>
    /// Xtal test times, 0 or 1 for test only 1 time
    /// </summary>
    uint8_t nXtalNums;
    /// <summary>
    /// Xtal offset, offset between each Xtal test, only used in multi Xtal tests
    /// </summary>
    uint8_t nXtalOffset;
    /// <summary>
    /// GPIO used for XTAL, just use IOType and IOPin, IOMux is not needed.
    /// If use GPIO_2, then IOType = 0, IOPin = 2
```

```
    /// when use AON_1, then IOType = 1, IOPin = 1
    /// </summary>
    stIOConfig xtalGPIO;
}stTestSettingConfig;
```

- The stNVDSConfig structure is used to configure NVDS. It is defined as follows:

```
typedef struct
{
    /// <summary>
    /// page nums in 4K, should be >= 1
    /// </summary>
    uint8_t nPageSize;
    /// <summary>
    /// NVDS address, must set this value when there is value to be written in NVDS
        (BD Address, xtal value for example)
    /// The default value of 5513 series is 0x0107E000, and the default value of
        5515 series is 0x010FF000
    /// </summary>
    uint32_t unStartAddr;
    /// <summary>
    /// The absolute file path of NVDS json file, When Write NVDS File is enabled,
        must pass the path
    /// </summary>
    char NVDSPath[GRPLT_MAX_PATH_LEN];
}stNVDSConfig;
```

- The stExtIOConfig structure is used to configure external I/O pins. It is defined as follows:

```
typedef struct
{
    /// <summary>
    /// Used for Ext.Flash
    /// 0 for SPI
    /// 1 for QSPI 0
    /// 2 for QSPI 1
    /// 3 for QSPI 2
    /// </summary>
    uint8_t nExtFlashType;
    /// <summary>
    /// QSPI/SPI CS config, used for Ext. Flash
    /// </summary>
    stIOConfig extCSConfig;
    /// <summary>
    /// QSPI/SPI CLK config, used for Ext. Flash
    /// </summary>
    stIOConfig extCLKConfig;
    /// <summary>
    /// QSPI IO0 or SPI MOSI, used for Ext. Flash
    /// </summary>
    stIOConfig extIO0Config;
    /// <summary>
    /// QSPI IO1 or SPI MISO, used for Ext. Flash
    /// </summary>
    stIOConfig extIO1Config;
    /// <summary>
    /// QSPI IO2, used for Ext. Flash
    /// </summary>
    stIOConfig extIO2Config;
    /// <summary>
```

```
    /// QSPI IO3, used for Ext. Flash
    /// </summary>
    stIOConfig extIO3Config;
}stExtIOConfig;
```

- The stEraseFlashConfig structure is used to erase Flash data. It is defined as follows:

```
typedef struct
{
    /// <summary>
    /// Start Address
    /// </summary>
    uint32_t unStartAddr;
    /// <summary>
    /// Endd Address
    /// </summary>
    uint32_t unEndAddr;
}stEraseFlashConfig;
```

- The stBinConfig structure is used to configure firmware. It is defined as follows:

```
typedef struct
{
    /// <summary>
    /// Save Address
    /// </summary>
    uint32_t unSaveAddr;
    /// <summary>
    /// Absolute file path
    /// </summary>
    char filePath[GRPLT_MAX_PATH_LEN];
}stBinConfig;
```

- The stFlashConfig structure is used to configure Flash. It is defined as follows:

```
typedef struct
{
    /// <summary>
    /// 1/0, 1 for enable downloading flash data to DUT
    /// </summary>
    uint8_t nDownloadEnabled;
    /// <summary>
    /// 1/0, 1 for enable erase internal flash data
    /// </summary>
    uint8_t nEraseInternalEnabled;
    /// <summary>
    /// 1/0, 1 for enable erase external flash data
    /// </summary>
    uint8_t nEraseExternalEnabled;
    /// <summary>
    /// 0 for Internal Flash
    /// 1 for External Flash
    /// </summary>
    uint8_t nFlashMode;
    /// <summary>
    /// Ext. Flash IO Config
    /// </summary>
    stExtIOConfig extIOConfig;
    /// <summary>
```

```
    /// Internal Erase Flash Setting, support up to 3 sections
    /// </summary>
    stEraseFlashConfig intEraseConfig[GRPLT_MAX_ERASE_CNT];
    /// <summary>
    /// External Erase Flash Setting, support up to 3 sections
    /// </summary>
    stEraseFlashConfig extEraseConfig[GRPLT_MAX_ERASE_CNT];
    /// <summary>
    /// Download Flash Config, support up to 5 files
    /// </summary>
    stBinConfig downloadConfig[GRPLT_MAX_FLASH_BIN];
}stFlashConfig;
```

- The steFuseConfig structure is used to configure eFuse. It is defined as follows:

```
typedef struct
{
    /// <summary>
    /// 0/1, 1 for enable Write KeyInfo
    /// </summary>
    uint8_t nWriteKeyInfo;
    /// <summary>
    /// 0/1, 1 for enable Write Key Info with one machine one secrect only work with
        Write Key Info enabled
    /// </summary>
    uint8_t nOneToOneSecrect;
    /// <summary>
    /// 0/1, 1 for enable Write custom encrypt info, conflict with write key info
    /// </summary>
    uint8_t nWriteCusEnc;
    /// <summary>
    /// The data offset of custom encrypt info, should be <= 31
    /// </summary>
    uint8_t nWriteCusOffset;
    /// <summary>
    /// when using write key info mode, must pass the absolute path of Encrypt_key_info.ebin
    /// </summary>
    char modeBinFilePath[GRPLT_MAX_PATH_LEN];
    /// <summary>
    /// when using write key info mode, must pass the absolute path of Mode_control.ebin
    /// </summary>
    char keyBinFilePath[GRPLT_MAX_PATH_LEN];
    /// <summary>
    /// When using write custom info, must pass the dll path.
    /// </summary>
    char cusEncDllFilePath[GRPLT_MAX_PATH_LEN];
}steFuseConfig;
```

- The stDownloadAppConfig structure is used to configure application firmware download. It is defined as follows:

```
typedef struct
{
    /// <summary>
    /// 0/1, 1 for enable Write Image Info Only
    /// </summary>
    uint8_t nImageInfoOnly;
    /// <summary>
    /// Save Address, when EditAddr is enabled, use this value
    /// when EditAddr is disabled, program will not use this value
    /// </summary>
```

```
    uint32_t unSaveAddr;
    /// <summary>
    /// Absolute file path
    /// </summary>
    char filePath[GRPLT_MAX_PATH_LEN];
}stDownloadAppConfig;
```

- The stCustomAppConfig structure is used to configure custom application firmware download. It is defined as follows:

```
typedef struct
{
    /// <summary>
    /// Boot option
    /// </summary>
    uint8_t nBoot;
    /// <summary>
    /// 0 for Pre Download
    /// 1 for Local Download
    /// </summary>
    uint8_t nDownloadMode;
    /// <summary>
    /// Edit Save Address
    /// </summary>
    uint8_t nEditAddr;
    /// <summary>
    /// Download custom app config ,support up to 8 bins, can be bin/ebin
    /// </summary>
    stDownloadAppConfig downloadConfig[GRPLT_MAX_APP_BIN];
    /// <summary>
    /// Pre download custom app config, support up to 8 bins, can be bin/ebin
    /// </summary>
    stBinConfig preDownloadConfig[GRPLT_MAX_APP_BIN];
    /// <summary>
    /// Mergebin path, when path is not empty, use the setting from mergebin
       (including boot, bins' config)
    /// </summary>
    char mergeBinPath[GRPLT_MAX_PATH_LEN];
}stCustomAppConfig;
```

- The stTestConfig structure is used to initialize test configurations. It is defined as follows:

```
typedef struct
{
    stCheckCustomConfig checkCustomConfig;
    stBDAddressConfig bdAddressConfig;
    stTestSettingConfig testSettingConfig;
    stNVDSConfig nvdsConfig;
    stFlashConfig flashConfig;
    steFuseConfig eFuseConfig;
    stCustomAppConfig customConfig;
    /// <summary>
    /// The launch time of DUT after reset. Program need to reset all the DUTs and enter
        DFU status after starting test, recommend 500ms at least
    /// </summary>
    uint16_t uRstDUTTime;
}stTestConfig;
```