Developing and Using RootScripts

Version 1.2
July 3, 2003

1 Introduction

A Root tester can execute precompiled "scripts" (sequences of Root tester commands) from its own local memory. The primary advantage of Script Mode is that it eliminates the communication delay of the serial port in processing commands, and allows a Root tester to interact with a device at much greater speeds than is otherwise possible.

Once fully debugged, scripts can be programmed into flash and executed when the Root-tester powers up. Thus you can customize your Root tester to implement a stand-alone test unit to meet virtually any requirement.

Script commands must adhere to a specific command syntax required by the Root tester2. relieve the user of the details of the specifics of this syntax and of constructing scripts themselves, RPM Systems provides a tool in the form of a MS Visual C/C++ Dynamically-Linked-Library called **rootscript.dll**. You write the script as a "C" program, compile and link it with rootscript.dll, and run the resulting executable; your application produces a script file which you can download to your Root tester via either the TapRoot application, or your own custom application. In fact, using TapRoot and a script created using **rootscript.dll** may be the quickest way to customize your Root tester to perform tests specific to your device.

Scripting is a powerful feature and can be used to generate simple or complex USB timing and traffic. It requires a thorough prior understanding your Root tester's operational capabilities. You should be familiar with the document entitled Root-1 Interface Specification and/or Root-2 Interface Specification prior to reading this document – paying particular attention the description of your Root tester's script mode commands, in addition to the standard command set.

2 Basic Concepts

To understand the basic concept behind creating a script using this technique, refer to the example script given in the Root-[1,2] Interface Specification . This simple script sets VCC, applies power, and ends. To actually create this script using rootscript.dll, we perform the following steps:

1) Write the following "C" application called "vccscript":

```
#include "stddefs.h"
#include "rootscript.h"

int main(void)
{
      FILE *usb_stream;
```

```
        UWORD ScriptBytes = 0;
        // open script file
        usb_stream = RS_Fopen("simple.rs") ;
        RS_Program(usb_stream);              /* begin script */
        ScriptBytes += RS_VCC(usb_stream,
                    100);                    /* Set voltage = 5.0 */
        ScriptBytes += RS_Power(usb_stream
                              ,1);           /* Power On */
        ScriptBytes += RS_End(usb_stream);   /* End script */
        RS_Fclose(usb_stream);
}
```

2) compile and link the above application with the supplied library **rootscript.lib** in the link to resolve all calls to the DLL.

3) run the resulting executable **vccscript.exe**, which creates the file **simple.rs** as its output – and this is your rootscript. This script can be immediately downloaded to Root-2 via the TapRoot application or a custom app of your design.

3 Getting Started

3.1 Required Files

The following files are included with the rootscript distribution and are necessary for building a custom application:

Stddefs.h:	type declarations; required for calling application;
rootscript.h:	constant declarations and function prototypes; required for calling application;
rootscript.lib:	corresponding library file for rootscript.dll; required in link with Calling application.

3.2 Type Declarations

rootscript function interface prototypes use the following type declarations for byte, word, and long data sizes:

```
typedef unsigned short  UWORD;
typedef unsigned char   UBYTE;
typedef unsigned long   ULONG;
```

These data types are declared in the file "stddefs.h". Any source file, which uses routines in rootscript.dll, should include this file.

3.3 Function Return Values

All rootscript function calls return a UWORD that represents the number of bytes generated (added to the length of the script file) as a result of the function invocation. This is available as a convenience – with it you can accumulate the total length of the script file – but it is not required.

Here is an example of a rootscript call to turn on power:

```
//power the device
ScriptBytes += RS_Power(usb_stream,POWER_ON);
```

3.4 Miscellaneous Predeclarations

The following declarations are also included in rootscript.h for your convenience, and are referenced in the following sections:

```
//useful declarations for valid values for Root-1 Commands

//predeclarations for R1_RootConfig command
#define ROOT1_MODE                  0
#define AUTO_MODE                   1
#define MANUAL_MODE                 0
#define ROOT1_TRIGGER               1
#define TRIGGERS_DISABLED           0
#define TRIGGER1_ENABLED            1
#define TRIGGER2_ENABLED            2
#define ROOT1_AUTORECOVERY          2
#define AUTORECOVERY_ENABLED  1
#define AUTORECOVERY_DISABLED 2

#define ROOT1_LED_INDICATORS    3
#define ENABLE_LED_INDICATORS   1
#define DISABLE_LED_INDICATORS  0

#define ROOT1_PUSHBUTTONS           4
#define ENABLE_PUSHBUTTONS          1
#define DISABLE_PUSHBUTTONS         0

#define ROOT1_BAUD                          5
//Baud set according to following table:
#define ROOT1_BAUD_2400                     0  //-- 2400
#define ROOT1_BAUD_9600                     1 // -- 9600
#define ROOT1_BAUD_19200                    2 // -- 19200
#define ROOT1_BAUD_38400                    3 // -- 38400
#define ROOT1_BAUD_57600                    4 // -- 57600
#define ROOT1_BAUD_115200                   5 // -- 115200
#define ROOT1_BAUD_230400                   6 // -- 230400
```

```
#define ROOT1_BAUD_460800                7 // -- 460800


#define ROOT1_CONNECT_SPEED_CONTROL      6
#define INHIBIT_HS                       1
#define ALLOW_HS                         0

//predeclarations for R1_Power
#define POWER_OFF                  0
#define POWER_ON                   1

//useful PID declarations for R1_DevRqst, R1_DevTransXXX
#define OUT_PID                    0x1
#define ACK_PID                    0x2
#define DATA0_PID                  0x3
#define PING_PID                   0x4
#define NYET_PID                   0x6
#define DATA2_PID                  0x7
#define IN_PID                     0x9
#define NAK_PID                    0xa
#define DATA1_PID                  0xB
#define SETUP_PID                  0xD
#define STALL_PID                  0xe
#define MDATA_PID                  0xf

// DevTrans Control Byte Defines
//bit 0 is reserved!
#define DT_LOW_SPEED    0x0000 //perform transaction at low speed
#define DT_FULL_SPEED   0x0002 //perform transaction at full speed
#define DT_HIGH_SPEED   0x0008 //perform transaction at high speed
#define DT_ISOCH        0x0004 //perform isochronous transaction
#define DT_USE_ALT_BUFFER 0x0010 //use alternate buffer
#define DT_IMMED        0x0080 //issue transaction immediately

//these can  only be used with block dev trans command
#define DT_STOP_ON_NAK  0x0200 //stop transaction when device NAKs

//used in conjunction with response from R1_RootStatus command
#define ROOTSTATUS_CONNECT_MASK                 0x43
#define ROOTSTATUS_POWER_MASK                   4
#define ROOTSTATUS_SUSPEND_MASK                 8
#define ROOTSTATUS_ENABLED_MASK                 0x10
#define ROOTSTATUS_AUTORECOVERY_MASK            0x20
#define ROOTSTATUS_NOT_CONNECTED                0
#define ROOTSTATUS_FULL_SPEED_CONNECT           2
#define ROOTSTATUS_LOW_SPEED_CONNECT            1
#define ROOTSTATUS_HIGH_SPEED_CONNECT           0x40
#define ROOTSTATUS_CONNECTED                    0x43




//DevRqst predefines
//DevRqst predefines
#define DR_PACKETSIZE_8       0
#define DR_PACKETSIZE_16      1
#define DR_PACKETSIZE_32      2
```

```
#define DR_PACKETSIZE_64      3
#define DR_HIGH_SPEED         2
#define DR_FULL_SPEED         1
#define DR_LOW_SPEED          0
```

4 Rootscript Functions

With the exception of the file maintenance commands, there is a one-to-one correspondence with the Rootscript commands documented in the following section, and Root-1 commands as defined in the Root-[1,2] Interface Specification. Refer to the interface spec for additional details on each command.

4.1 RS_Fopen --- Open Script File for Writing

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


FILE * RS_FOpen(char * filepath);
```

Description

RS_FOpen() opens a rootscript output file.

filepath: file path and name of file

Returns

0: error occurred opening file
File Handle: file opened successfully.

(The file handle is necessary in subsequent rootscript calls.)

Example:

```
{
	FILE *usb_stream;
	//..
	if (!(usb_stream = RS_Fopen ("test.rs")))
	{
		printf("error opening rootscript file\r\n");
	}
}
```

4.2 RS_Fclose -- Close Rootscript file

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


void  RS_Fclose (FILE * filehandle);
```

Description

RS_Fclose() closes the file associated with filehandle.

Returns

Nothing.

Example:

```
{
      FILE *usb_stream;
      //..
      RS_Fclose(usb_stream);
      printf("rootscript file closed \r\n");
}
```

4.3 RS_USB_Reset --- Resets the USB device

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_USB_Reset(FILE *filehandle);
```

Description

RS_USB_Reset() appends a USB_Reset command to the rootscript file specified by "filehandle".

Returns

Total bytes added to file pointed to by filehandle.

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      ScriptBytes += RS_USB_Reset(usb_stream);  // Reset device
}
```

4.4 RS_Suspend --- Suspend Device

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_Suspend(file *filehandle);
```

Description

RS_Suspend() appends a Global Suspend command to the rootscript file specified by "filehandle".

Returns

Total bytes added to file pointed to by filehandle.

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      ScriptBytes += RS_Suspend(usb_stream);    //suspend device
}
```

4.5 RS_Resume --- Resume Device

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_Resume(file *filehandle);
```

Description

RS_Resume appends a Global Resume command to the rootscript file specified by "filehandle".

Returns

Total bytes added to file pointed to by filehandle.

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      ScriptBytes += RS_Resume(usb_stream);     //resume device
}
```

4.6 RS_Power --- Control Power to USB Port

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_Power(file *filehandle ,UBYTE On);
```

Description

RS_Power()appends a Global Resume command to the rootscript file specified by "filehandle".

If "On" is non-zero, power is applied to the port, otherwise it is turned off.

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      ScriptBytes += RS_Power(usb_stream,POWER_ON);
}
```

4.7 RS_VCC --- Set VCC

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_VCC(file *filehandle ,UBYTE Value);
```

Description

RS_VCC() appends a Set VCC command to the rootscript file specified by "filehandle".

The relationship between the parameter "Value" and the level of VCC is given in volts by:

VCC = 4 + Value/100

Thus, a value of 100 would give a VCC of 5 volts.

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      UWORD ScriptBytes;
      FILE *usb_stream;
      //..
      //..
      ScriptBytes += RS_VCC(usb_stream,100);
}
```

4.8 RS_VccMeasI --- Single Precision Current Measurement

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_VccMeasI(file *filehandle);
```

Description

RS_VccMeasI() appends a "Measure Root Port ICC" command to the rootscript file specified by "filehandle".

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      ScriptBytes += RS_VccMeasI(usb_stream);
}
```

4.9 RS_VccMeasI_DP --- Double Precision Current Measurement

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_VccMeasI_DP(file *filehandle);
```

Description

RS_VccMeasI_DP() appends a "Measure Root Port ICC_DP" command to the rootscript file specified by "filehandle".

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      ScriptBytes += RS_VccMeasI_DP(usb_stream);
}
```

4.10 RS_RootStatus --- Get_Root1 Port Status

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_RootStatus(file *filehandle);
```

Description

RS_RootStatus() appends a "Get_Rootstatus" command to the rootscript file specified by "filehandle".

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      ScriptBytes += RS_RootStatus(usb_stream);
}
```

4.11 RS_RootConfig --- Configure Root-1

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_RootConfig(file *filehandle ,UBYTE Parameter,UBYTE Value);
```

Description

RS_RootConfig() appends a "Root Config" command to the rootscript file specified by "filehandle".

The valid parameters and their allowable values are:

Parameter	Value(s)
ROOT1_MODE	MANUAL_MODE, AUTO_MODE
ROOT1_TRIGGER	TRIGGERS_DISABLED, TRIGGER1_ENABLED, TRIGGER2_ENABLED, (TRIGGER1_ENABLED | TRIGGER2_ENABLED)
ROOT1_AUTORECOVERY(2)	AUTORECOVERY_DISABLED, AUTORECOVERY_ENABLED
ROOT1_LED_INDICATORS	ENABLE_LED_INDICATORS, DISABLE_LED_INDICATORS
ROOT1_PUSHBUTTONS	ENABLE_PUSHBUTTONS,DISABLE_PUSHBUTTONS
ROOT1_BAUD	ROOT1_BAUD_2400,ROOT1_BAUD_9600, ROOT1_BAUD_19200, ROOT1_BAUD_38400, ROOT1_BAUD_57600, ROOT1_BAUD_115200, ROOT1_BAUD_230400 ROOT1_BAUD_460800
ROOT1_CONNECT_SPEED_CONTROL	INHIBIT_HS,ALLOW_HS

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      //configure root1 for manual mode
      ScriptBytes += RS_RootConfig(usb_stream,ROOT1_MODE,MANUAL_MODE);
```

```
        //configure root1 for auto mode
        ScriptBytes += RS_RootConfig(usb_stream,ROOT1_MODE,AUTO_MODE);
        //disable both triggers
        ScriptBytes += RS_RootConfig(usb_stream,ROOT1_TRIGGER,
                    TRIGGERS_DISABLED);
        //enable both triggers
        ScriptBytes += RS_RootConfig(usb_stream,ROOT1_TRIGGER,
                    TRIGGERS_ENABLED | TRIGGER2_ENABLED);
}
```

4.12 RS_DataPort --- Strobe Data Port

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_DataPort(file *filehandle ,UBYTE data);
```

Description

RS_DataPort() appends a DataPort command to the rootscript file specified by "filehandle".

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      //toggle the dataport output
      ScriptBytes += RS_DataPort(usb_stream, 0xff);
      ScriptBytes += RS_DataPort(usb_stream, 0x00);
}
```

4.13 RS_DevRqst --- Issue Device Request

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_DevRqst(file *filehandle ,UBYTE Address, UBYTE *DataBuff,UWORD
DataLen);
```

Description

RS_DevRqst() appends a DevRqst command to the rootscript file specified by "filehandle".

The arguments are as follows:

Filehandle:	handle of rootscript file
Address:	address of the device
DataBuff:	pointer to data to send to the device
DataLen:	size of data pointed to by "data", in bytes

Entire USB SETUP transactions – the SETUP through STATUS phases -- can be issued to a downstream device using a single RS_DevRqst command. RS_DevRqst commands can be issued to devices configured by the Root-1 in automatic mode. You MUST match the address of the targeted device to the address assigned by Root-1 when it configured the device. The Root-1 will use this address to match the speed and packet size of the device in question when it issues the RS_DevRqst command (see RS_DevRqstMan() to issue device requests with explicit packet size and speed settings).

The data in DataBuff should minimally contain an 8-byte SETUP command – and it follows that DataLen should not be less than 8. In the case of a host-to-device directional transfer (OUT transactions following the SETUP phase), the additional data to be sent in OUT transactions should be concatenated to the SETUP command in DataBuff and the additional length of this data accounted for in DataLen.

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      //here's a SETUP command to get the configuration descriptor.
      //Note that if the direction of the SETUP was HOST to DEVICE, the
```

```
        //additional data to be sent after the SETUP would be defined
        //after the initial 8 bytes in the array below.
        UBYTE get_config[8] = {0x80,0x06,0x00,0x01,0x00,0x00,0x12,0x00};

        FILE *usb_stream;
        UWORD ScriptBytes;
        //..
        //..

        //call RS_DevRqst to get the config descriptor.
        //This example assumes that prior to this code fragment, a device
        //has been plugged into the Root-1 and was configured in
        //Automatic mode.  Therefore its address will be 2 (Root1 always
        //assigns the root downstream device an address of 2)
        //
        ScriptBytes += RS_DevRqst(usb_stream,2,get_config,
                           sizeof(get_config));

}
```

4.14 RS_DevRqstMan --- Issue Device Request Manually

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_DevRqstMan(file *filehandle ,UBYTE Address, UBYTE Speed,
                    UBYTE Packetsize, UBYTE *DataBuff, UWORD DataLen);
```

Description

RS_DevRqstMan() appends a DevRqst command to the rootscript file specified by "filehandle" - with parameters for setting device speed and Packetsize explicitly:

Address:	address of the device
Speed:	speed of transaction: 1 = full speed, 0 = low speed
Packetsize:	max packet size to use in request: 8,16,32, or 64 bytes (encoded)
DataBuff:	pointer to data to send to the device
DataLen:	size of data pointed to by "data", in bytes

RS_DevRqstMan is identical to RS_DevRqst, except that it allows the speed and packet size of the request to be explicitly set prior to the transaction. You can use this command to "manually" issue device requests to devices that have not been automatically configured by Root-1. Typically this command is used when Root-1 is running in manual mode, to communicate with devices when more control or customization over the setup process is desired.

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      //here is an example of how to set the address of a device in
      //manual mode, using RS_DevRqstMan.

      //define a command to set the address to 0x55
      UBYTE set_address[8] = {0,5,0x55,0,0,0,0,0};

      //call RS_DevRqstMan to set the address
      ScriptBytes += RS_DevRqstMan(usb_stream, 0, DR_FULL_SPEED,
                        PACKETSIZE_8, set_address,sizeof(set_address));
```

```
        }
}
```

4.15 RS_DevTransOut – issue a DevTrans Host-to-Device Request

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_DevTransOut(file *filehandle ,UBYTE Address, UBYTE Endpoint,
UBYTE pid, UBYTE Control, UBYTE DataPID, UBYTE* DataBuff,
                  UWORD DataLen);
```

Description

RS_DevTransOut() appends a "DevTrans Out " command to the rootscript file specified by "filehandle" – that is, a single "DevTrans" command with a host-to-device directional data transfer.

Address: address of the device
Endpoint: endpoint of the transfer
Pid: PID
Control: Control Byte for the transfer: can be the exclusive or of the following:

```
{DT_LOW_SPEED, DT_FULL_SPEED, DT_HIGH_SPEED};
DT_ISOCH                              //perform isochronous transaction
DT_USE_ALT_BUFFER 0x0010              //use alternate buffer
DT_IMMED                              //issue transaction immediately
```

DataPID: Data PID
DataBuff: pointer to data to send to the device
DataLen: size of data pointed to by "DataBuff", in bytes

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..

      UBYTE set_address[8] = {0,5,2,0,0,0,0,0};

      ScriptBytes += RS_DevTransOut(
```

```
            usb_stream,       //filehandle
            0,                //address = 0
            0,                //endpoint = 0
            SETUP_PID,        //SETUP pid
            DT_FULLSPEED,     //full-speed transaction
            DATA0_PID,        //data0 pid for data portion of
                              //transaction
            set_address,      //pointer to SET_ADDRESS command
            sizeof(set_address)//size of command
            );

}
```

4.16 RS_DevTransIn – issue a DevTrans Device-to-Host Request

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_DevTransIn(file *filehandle ,UBYTE Address, UBYTE Endpoint,
UBYTE pid,  UBYTE Control);
```

Description

RS_DevTransIn() appends a "DevTrans In" command to the rootscript file specified by "filehandle"-- that is, a "DevTrans" command with a device-to-host directional data transfer.

Address:	address of the device
Endpoint:	endpoint of the transfer
Pid:	PID
Control:	Control Byte for the transfer; can be the exclusive or of the following: Bit definitions:

```
{DT_LOW_SPEED, DT_FULL_SPEED, DT_HIGH_SPEED};
DT_ISOCH                            //perform isochronous transaction
DT_USE_ALT_BUFFER 0x0010            //use alternate buffer
DT_IMMED                            //issue transaction immediately
                                    //without waiting for sof
```

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      ScriptBytes += RS_DevTransIn(
            usb_stream,             //file handle
            2,                      //device address = 2
            0,                      //endpoint 0
            IN_PID,                 //IN pid
            DT_FULLSPEED                  //full speed transaction
            );
}
```

4.17 RS_BlockDevTrans --- Start Block Transaction (Root-2 Only)

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_BlockDevTrans(FILE *fileptr,BLOCK_DEV_TRANS_STRUCT *p);
```

Description

RS_BlockDevTrans () appends a "Start Block Transaciton" command to the rootscript file specified by "filehandle".

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      BLOCK_DEV_TRANS_STRUCT testH;
      //..
      //..
      //initialize our test structure
      testH = default_block_dev_transH;
      ScriptBytes += RS_BlockDevTrans (usb_stream,&testH);

}
```

4.18 RS_BlockDevTransStatus --- Get BlockTransaction Status (Root2 only)

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_BlockDevTransStatus (file *filehandle);
```

Description

RS_BlockDevTransStatus () appends a "Get Block Transaction Status" command to the rootscript file specified by "filehandle".

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      ScriptBytes += RS_BlockDevTransStatus (usb_stream);
}
```

4.19 RS_StopBlockDevTrans --- Stop A Block Transaction (Root2 only)

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD R2_ StopBlockDevTrans (file *filehandle);
```

Description

RS_StopBlockDevTrans() appends a "Stop Block transaction" command to the rootscript file specified by "filehandle.

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      ScriptBytes += RS_StopBlockDevTrans(usb_stream);
}
```

4.20 RS_Program--- Program Script File

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_Program(file *filehandle );
```

Description

RS_Program appends a "Program" command to the rootscript file specified by "filehandle".

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      ScriptBytes +=  RS_Program(usb_stream);
}
```

4.21 RS_Run--- Run Script File

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_Run(file *filehandle);
```

Description

RS_Run() appends a "Run" command to the rootscript file specified by "filehandle".

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      ScriptBytes += RS_Run(usb_stream);
}
```

4.22 RS_Get_DLL_Version--- Get Version

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_Get_DLL_Version(void);
```

Description

RS_Get_DLL_Version() returns the version of the DLL.

Returns

16-bit version number in hexadecimal form (i.e., 0x0100 = version 1.00).

Example:

```
{
      UWORD Version = RS_Get_DLL_Version();
      if (Version < 0x100)
      {
            printf("DLL out of date\r\n");
      }
      //..
      //..
      //..
}
```

4.23 RS_Message --- Send Message

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_Message(file *filehandle, UBYTE *message,
             UBYTE MsgLen);
```

Description

RS_Message()appends a rootscript "RS_Message" command to the rootscript file specified by "filehandle".

Filehandle:	handle of rootscript file
Message:	pointer to byte string
MsgLen:	length of message, in bytes

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{


      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      UBYTE Message[] = {0,1,2,3,4,5};
      //send message.
      ScriptBytes += RS_Message(usb_stream, Message,sizeof(Message));
      //..
      //..

}
```

4.24 RS_Response --- Set Response Mode

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_Response(file *filehandle, UBYTE Mode);
```

Description

RS_Response()appends a rootscript "RS_Response" command to the rootscript file specified by "filehandle".

Filehandle:	handle of rootscript file

Mode:	mode. Can be one of:
	`FULL_RESPONSE (0)`
	`QUIET (1)`

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{


      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..
      //be quiet!
      ScriptBytes += RS_Response(usb_stream,QUIET);
      //..
      //..

}
```

5 Rootscript Flow Control Functions

The following rootscript functions maintain a one-to-one correspondence with rootscript flow control commands described in the Root-1 Interface Specification.

Recall from the Root-1 Interface Specification that as rootscript commands are loaded into Root-1, they are assigned 16-bit indexes starting at index 0 and incrementing by 1 for each command. Thus each command in the rootscript has a unique index associated with

it. Program flow is controlled by allowing the script to transfer control to a new execution index.

5.1 RS_CurIdx--- Get Current Index

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_CurrIdx(void);
```

Description

RS_CurrIdx() returns the current rootscript index. This call is used in conjunction with other flow-control functions to create loops within a rootscript.

Note: the program index is automatically reset to 0 when a RS_Program() command is called.

Returns

16-bit rootscript index

Example:

```
{
      FILE *usb_stream = RS_Fopen("test.rs");
      UWORD ScriptBytes = RS_Program(usb_stream);


      UWORD Index = RS_CurIdx();
      //Index is  0 now.
      //Subsequent command (RS_Response) will be assigned
      //this index

      ScriptBytes += RS_Response(usb_stream,QUIET);

      Index = RS_CurIdx();
      //Index is 1 now. Subsequent command
      //(RS_Power) will be assigned this index.

      ScriptBytes += RS_Power(usb_stream,0);

      Index = RS_CurIdx();
      //Index is 2 now
      //..
      //..
}
```

5.2 RS_Goto--- Goto

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_Goto(file *filehandle, UWORD index);
```

Description

RS_Goto() appends a rootscript "goto" command to the rootscript file specified by "filehandle".

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;
      UWORD Index;
      //..
      //..

      ScriptBytes += RS_Power(usb_stream,POWER_ON);

      //get the current index.  This is the index
      //which the NEXT rootscript command (RS_USB_Reset)
      //will be assigned.
      Index = RS_CurIdx();
      ScriptBytes += RS_USB_Reset(usb_stream);
      ScriptBytes += RS_Goto(usb_stream,Index);

      //note infinite loop created above! It will repeatedly
      //reset the device.
}
```

5.3 RS_Call, RS_Return --- Function Call and Return

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_Call(file *filehandle, UWORD index);
UWORD RS_Return(file *filehandle);
```

Description

RS_Call() and RS_Return append rootscript "call" and "return" commands, respectively, to the rootscript file specified by "filehandle".

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      //following fragment creates a rootscript
      //that has 2 subroutines and calls them.
      //..
      //..
      FILE *usb_stream;
      UWORD ScriptBytes;
      //..
      //..

      UWORD PowerOnSubIndex,ResetSubIndex;

      //first subroutine applies power...
      PowerOnSubIndex = RS_CurIdx();
      ScriptBytes += RS_Power(usb_stream,POWER_ON);
      ScriptBytes += RS_Return(usb_stream);

      //second subroutine resets device..
      ResetSubIndex = RS_CurIdx();
      ScriptBytes += RS_USB_Reset(usb_stream);
      ScriptBytes += RS_Return(usb_stream);


      //more subroutines here...
      //..
      //..

      //subroutine calls....
      ScriptBytes += RS_Call(usb_stream,PowerOnSubIndex);
```

```
        ScriptBytes += RS_Call(usb_stream,ResetSubIndex);
}
```

5.4 RS_Timer --- Set Timer

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_Timer(file *filehandle, ULONG Count);
```

Description

RS_Timer() appends a "RS_Timer" command to the rootscript file specified by "filehandle".

Count: initial count, in 1-msec increments, to initialize timer

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{
      FILE *usb_stream;
      UWORD ScriptBytes;

      //initialize timer...
      ScriptBytes += RS_Timer(usb_stream,100);
}
```

5.5 RS_If --- If

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_If(file *filehandle, UBYTE Cond, UWORD index);
```

Description

RS_If() appends a rootscript "RS_If" command to the rootscript file specified by "filehandle".

Cond: parameter to be checked for assertion.

```
STS_Success             // transaction successful
STS_ACK                 // transaction resulted in ACK
STS_NAK                 // transaction resulted in NAK
STS_STALL               //  "             "      " STALL
STS_IGNORE              //  "             "      " IGNORE
STS_DCRCErr             //  received incorrect CRC
STS_DTogErr             //  received incorrect data toggle pid
STS_SyncErr             //  received incorrect sync byte
STS_Babble              //  device babble
STS_PIDErr              //  incorrect pid format
STS_ShPktErr            //  packet too short
STS_ConfigErr           //
```

Index: 16-bit index to which control will be conditionally returned.

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{

      //following fragment loops until device stops NAKing
      //....
      FILE *usb_stream;
      UWORD ScriptBytes;
      UWORD i;
      //..
      //..
      //get value of index
      Index = RS_CurIdx();
      i = RS_CurIdx();
      ScriptBytes += RS_DevTransIn(usb_stream,
            address,0,IN_PID,FULLSPEED);  //IN for status
```

```
        ScriptBytes += RS_If(usb_stream,STS_NAK,i);


        //script execution will flow here when device stops naking
        //..
        //..

}
```

5.6 RS_Cond, RS_Check --- Check Conditionals and Branch

Synopsis

```
#include "stddefs.h"
#include "rootscript.h"


UWORD RS_Cond(file *filehandle, UBYTE Cond, UWORD index, UBYTE state);
UWORD RS_Check(file *filehandle, UBYTE init);
```

Description

RS_Cond() and RS_Check() append rootscript "RS_Cond" and "RS_Check" commands, respectively, to the rootscript file specified by "filehandle". These commands are used together to create conditions upon which the script will pend.

Cond: parameter to be checked for assertion. Can be one of:

```
CHK_Connect       //device connect
CHK_Disconnect    //device disconnect
CHK_Resume        //device resume
CHK_TrigIn0       //Trigger 0 asserted
CHK_TrigIn1       //Trigger 1 asserted
CHK_Timeout       //timer timeout
CHK_BlockDone     //Block command completed
```

Index: 16-bit index to which control will be conditionally returned.

State: 0 (disable) or 1(enable) the condition

Init: Bit-field indicating whether to clear latched conditions prior to entering RS_Check. Can be an "or" of only the conditions which are latchable:

```
(1 << CHK_TrigIn0)      //clear trigger 0
(1 << CHK_TrigIn1)      //clear trigger 1
```

Returns

Total bytes added to file pointed to by filehandle

Example:

```
{

      //following fragment delays for 10 ms before proceeding...
```

```
//proceeding....

FILE *usb_stream;
UWORD ScriptBytes;
//..
//..
UWORD Index;

//set timer for 10 ms
ScriptBytes += RS_Timer(usb_stream,10);
//..
//..
//get value of index
Index = RS_CurIdx();

//note "Index+2" argument passed to RS_Cond
//this will cause program flow to jump 2 script commands
//ahead - after the RS_Cond and RS_Check -
//when device connects.

//enable timeout condition and index
ScriptBytes += RS_Cond(usb_stream,
            CHK_Timeout,((UWORD)(Index+2)),Enabled);

//issue RS_Check.  This will cause a pend until the timer
//times out, after which program flow will continue to the
//script index after this one....
ScriptBytes += RS_Check(usb_stream,0);

//script execution resumes here after timeout....
//..
//..
//following fragment pends on either trigger before
//proceeding....

//get value of index
Index = RS_CurIdx();

//note "Index+3" argument passed to RS_Cond.
//this will cause program flow to jump 3 script commands
//ahead - after the 2 RS_Conds and  1 RS_Check -

//enable trigger0 condition and index
ScriptBytes +=
RS_Cond(usb_stream,CHK_TrigIn0,((UWORD)(Index+3)),ENABLED);
//enable trigger1 condition and index
      ScriptBytes +=
RS_Cond(usb_stream,CHK_TrigIn1,((UWORD)(Index+3)),ENABLED);

//issue RS_Check.  This will cause a pend until
//either trigger fires.  Note that the triggers are cleared
//prior to pending.
ScriptBytes += RS_Check(usb_stream,(1 << CHK_TrigIn0)
                  | (1 << CHK_TrigIn1));

//script execution resumes here after either trigger....
//..
```

```
        //..
}
```

6 Examples

Two example rootscript demonstrations are bundled within a single MS Visual C/C++ workspace entitled Examples.dsw, in the Examples directory of the RootScript main directory. To view and execute the examples, open the workspace in Visual C, then set the active project to one of the examples: either KeyboardTest, or BulkOut. You can then compile the project and execute it; this will generate the script file for subsequent downloading.

6.1 KeyboardTest

This rootscript demonstration generates a script file "kbtest.rs" which performs the following test on a HID keyboard:

Sets the USB Vcc to 5.0V
Enables USB power
Then performs the following loop continuously:
    For Device Address = 1 to 15 {
        USB Reset
        SetAddress
        Step through all 8 combinations of the
            three keyboard LEDs at a 2 Hz rate
            (this uses a SetReport request as defined in the HID spec)
    End For

Compiling and running the KeyboardTest project under VC++ will generate the file "kbtest.rs". Using TapRoot, you can download this script to either Root-1 or Root-2 and use it to verify the operation of the leds on a standard HID keyboard.

6.2 BulkOut

This rootscript demonstration generates a Root-2 compatible script file "bulkout.rs", which performs the following exercise on a high or full speed device with a bulk out endpoint:

Enables USB power;
Resets an attached device and allows it to enumerate;d
Sends 1024 bytes of data to a bulk endpoint at 8 packets per frame (microframe if if the device is high speed), 64 bytes per frame.

Using TapRoot, you can download this script to Root-2 and use it to verify the operation on any full or high-speed device capable of sinking bulk data at the example rate and packet size.

7 Other Resources

Refer to the document entitled Developing PC-Based Host Applications For Root Testers for an example of how to write a custom application that downloads rootscripts – as opposed to using TapRoot.