Radio Stack Software Development Kit

(Version 1.1)

for the following
Software Languages/Environments:

Microsoft Visual C++
Borland C++ Builder
Microsoft Visual Basic
Borland Delphi

This Radio Stack SDK requires programming knowledge on one the following programming languages: Microsoft Visual C++ or Borland C++ Builder or Microsoft Visual Basic or Borland Delphi

Improvements over the 1.0 version

General
LCD Display routines are improved. The radio stack component which is updated because the user turns the knob is updated before other displays are updated.
Radio Stack components can be enabled/disabled with checkboxes in the main screen of the application.
Support for rotary encoders panel added: VOR1OBS, VOR2OBS, AltiPres, Heading HDG, Heading Drift, ADFHDG

Comm 1/2
Channels support added, channels can be selected and programmed (32 channels)
Active/CDI mode added with VOR function
Volume read support for comm and nav

AudioPanel
Audio volume read support added
The Inner, Middle and Outer Marker illumination is added on the Audio Panel.
The COM1/COM2 selection is now controlled in the Audio Panel with the MIC selector.

Autopilot
Vertical speed / Altimeter height values can be negative on display.
There was an error when using the Autopilot inner knob. This is fixed and will change the altitude now by the correct value of 100 feet per click. (This was 20ft.)

ADF
Elapsed Time function added.
In MS FS, the ADF has the possibility to set fractions. Originally, the Bendix King Radio Stack has no decimal point and searches eventual frequencies with a value behind the decimal point automatically. This was not functioning properly in the previous Radio Stack software. This has been fixed.
ADF volume read support added

Transponder
When using the Transponder, the "R" signal is now blinking when enabled.
All the modes can now be selected in transponder (ALT/ON/TST/SBY/OFF)

DME
Minutes away from beacon is now also calculated displayed when nautical miles value is known.

List of Contents

1. Introduction

The TRC Radio Stack is an extreme realistic set of avionics, controlled by an electronics interface board called RSC and software drivers, developed by TRC Development b.v.

There are drivers available which connect directly with Microsoft Flight Simulator FS2002 or FS2004 (it is necessary to obtain a license for FSUIPC from Pete Dowson when using FS2004).

Because of many requests from cockpit builders with programming skills, TRC Development has produced a so-called Radio Stack SDK. The Radio Stack SDK enables you to drive the Radio Stack devices from your own (Flight Simulator) software.

To use the Radio Stack SDK, you must be a programmer. It is beyond the scope of this manual and beyond TRC Development to learn you how to program. SimKits / TRC Development will therefore not give any support on general programming questions, but on serious bug reporting only.

Basically, with the use of the Radio Stack SDK, there is no limit to use the Radio Stack Devices from any type of software written by yourself.

With the use of the Radio Stack SDK, you can read and write values directly to and from the Radio Stack Devices.

This Radio Stack SDK requires programming knowledge on one the following programming languages:

Language/Environment:
1. Microsoft Visual C++
2. Borland C++ Builder
3. Microsoft Visual Basic
4. Borland Delphi

This manual introduces how to use the Radio Stack SDK variables for the different programming environments.

When you find a bug in the Radio Stack SDK, there is only one way to send a comment or request for help.
There is a special form on the website under “Support”. When you encounter a problem, which is in your opinion not a programmer error, please enter the appropriate information into the “Bug Report” form and we will respond to you. No other bug reports can be accepted.

2. Recommended to read

TRC has more documentation available, which we recommend strongly to read prior to the use of the Radio Stack SDK and installation of any hardware and software.

The manuals and information are available at http://www.simkits.com/manuals.php

The software available at: http://www.simkits.com/software.php

The construction manuals available at http://www.simkits.com/manuals.php

3. How does the TRC Radio Stack work?

The TRC Radio Stack is designed from scratch to emulate the Silver Crown line from Bendix King as much as possible and as detailed as possible. Custom made LCD's display the same information as on the original Radio Stack. Special switches and mechanical parts have been produced using high quality electronics and plastic injection molding and made mainly from high quality ABS plastics.

The TRC Radio Stack has many advantages over other commercially available flight simulator Radio Stacks. Each module (radio) is a separate device, which you can position at any place in your panel. As most other commercially available radio stacks are a full single unit, the TRC Radio Stack allows you to buy just the radio you want. No more, no less!

The separate units from the Radio Stack are controlled from the RSC (Radio Stack Controller). This is an electronics board which connects to each separate Radio Stack module via a ribbon cable and on its turn the RSC connects to your PC – where flight simulator software is running – via a single USB connector. So whatever radio you would like to order, the RSC is always necessary.

The separate units from the Radio Stack are only available as built and tested products. This is due to the sensitive and dedicated electronics and complex composition of all parts.

The RSC is a Printed Circuit Board containing its own micro processor, memory and electronics to drive the individual radio stack devices, read out the dials (when connected) and to control the instrument lighting. The RSC is a stand-alone Printed Circuit Board which needs to be powered (5 volts only) from a PC AT Power Supply through a standard disk drive power connector. The board powers all instruments. The micro processor communicates via a single USB with the PC where TRC Development's driver and Radio Stack SDK software is running.

The RSC has unique features. The software (firmware) inside the board is downloaded from the PC via USB automatically, every time you power up the board. The software - which resides normally on the PC hard disk - can therefore be "refreshed" any time by downloading the latest drivers and firmware from the SimKits website. This assures you of having the latest improved software always available.

The RSC has a very large number of I/O lines. There are 10 I/O connectors available to control 10 different units. There is an I/O connector which is for future expansion with a Moving Map/GPS (not yet available).

The other connectors are to control the following devices:

CN1 - Audio Panel
CN2 - GPS (future expansion)
CN3 - NAV/COM1
CN4 - NAV/COM2
CN5 - DME
CN6 - ADF
CN7 - Autopilot
CN8 - Transponder
CN9 - Moving Map (future expansion)
CN10 – Dials for Altimeter Press. adjustment., VOR1, VOR2, ADF, Heading Indicator, Heading Bug *)

**) These knobs can be connected and used when you do not use The Real Cockpit or SimKits gauges but a Video Monitor or TFT Screen instead. The function of these knobs can be disabled or enabled during startup of the driver software.*

4. Principle of control

The displays are controlled by an LCD controller. Each separate radio which has an LCD display, also has an LCD controller chip built in. The LCD controller data is serial sent from the RSC to the individual radio. At the same time all switches, buttons and rotary encoders are read out and the data is sent back to the RSC.

The communication update rate between gauges/RSC/PC is now over 35 times per second. Many years of development and “try-and-error” are already behind our programmers and now these fine routines are ready to be used by yourselves.

Dials which are available on all the radios and the separate dials for Altimeter Press. adjustment, VOR1, VOR2, ADF, Heading Indicator, Heading Bug, which you can turn (by hand) are so-called Rotary Encoders.

These Rotary Encoders cannot be read out via the Radio Stack SDK directly. The routines inside the Radio Stack SDK control this part of the movement completely, freeing you from the difficult job to write your own control routines for the rotary encoder to read if it is turning left or right.

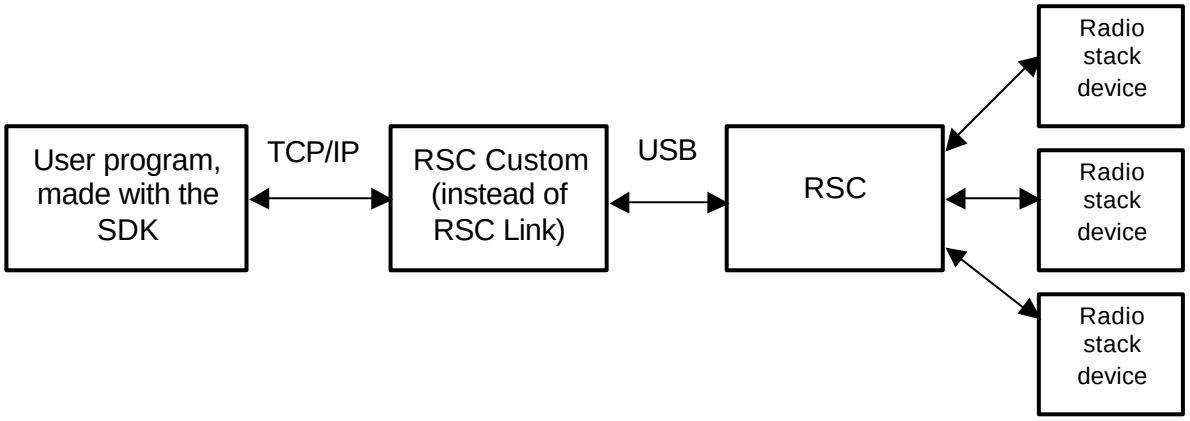
The knob movement therefore is read out by the Radio Stack SDK software, interpreted and available as up or down counted values, which are shown as an increased or decreased value on the active display. Using that information (together with other information from your flight simulator program) you can write the proper values (which are shown on the displays) back to the flight simulator program.
AT start-up, you need to write the proper start-up values into the display registers.

5. Communication between PC and the TRC Radio Stack

At the user level, communication with the Radio Stack devices is achieved using TCP/IP. This type of communication enables your program to drive the SimKits devices on the same computer where the RSC is installed or via another computer utilizing a standard network.

The Radio Stack SDK software handles the TCP/IP communication layer, so using the Radio Stack SDK does not require in deep TCP/IP programming knowledge.

The graph below describes the data flow from the user program to the Radio Stack Devices:



Actual communication between the RSC and the computer is realised via USB.

Although part the communication between your software and the Radio Stack SDK is carried out using a network protocol, it is not necessary to drive the devices via a network or another computer, nor do you have to have a network card installed.
However, if you configure the right IP address in your application, it is possible to control the devices via a network from another computer.

The Radio Stack SDK software does not read or write any values to a flight simulator program. A link to a flight simulator must be made by your application software.

If you prefer to drive the instruments from Microsoft Flight Simulator 2002/4 we recommend you to use the ready available RSC Link software, which interfaces with FSUIPC from Pete Dawson. FSUIPC is the link between the TRC Link software and Microsoft Flight Simulator 2002/4.

For interfacing with FS2004 via FSUIPC, a separate license is needed from Pete Dowson.

Using the sample projects provided with this Radio Stack SDK as a start, you can create a custom project within a short period of time.

6. Installing the Radio Stack SDK software

The installation software will install the files and software necessary to use the Radio Stack SDK and the Calibration software.

All necessary files to start using the Radio Stack SDK are packed in the file **RSCsetup.exe** (and Installshield installer) and will be automatically installed using Installshield.

However, there are 2 files, the USB driver **trcdrv.sys** and the **trccntl.inf** file packed together in **radiodriver.zip**, which need to be downloaded from the SimKits website software download area and unzipped into the directory:

C:\Program Files\TRC Development\Radio Stack SDK

When you have downloaded **RSCsetup.exe**, please exit all other programs and start the **RSCsetup.exe** program.

Now Installshield will install the Radio Stack SDK components in the following directories:

The files below will be installed in the directory:
C:\Program Files\TRC Development\Radio Stack SDK

RSCcustom.exe
This program is the Radio Stack SDK software.

DeIsL1.isu
This file is needed if you want to un-install the Radio Stack SDK

_DEISREG.ISR
This file is needed if you want to un-install the Radio Stack SDK

_ISREG32.DLL
This file is needed if you want to un-install the Radio Stack SDK

The files below will be installed in the directory:
(These files are common for all programming environments)

for Windows 98
C:\Windows\System

for Windows NT/2000
C:\WinNT\System32

for Windows XP
C:\Windows\System32

Trcradio.drv
This is a driver file for the RSC board.

vcl50.bpl
This file is needed for running the TRCCustom.exe program

cc3250.dll
This file is needed for running the TRCCustom.exe program

cc3250mt.dll
This file is needed for running the TRCCustom.exe program

bcbsmp50.bpl
This file is needed for running the TRCCustom.exe program

borlndmm.dll
This file is needed for running the TRCCustom.exe program

vclx50.bpl
This file is needed for running the TRCCustom.exe program

The files below will be installed in the directory:

C:\Program Files\TRC Development\Radio Stack SDK\Examples\Borland C++ Builder

Project1.bpr
This file contains information about the settings of the project.

Project1.cpp
This file contains C++ source code for the general application.

RADIOsampleCB.exe
This is the example program.

Project1.res
This file contains information about the icons, bitmaps, etc.

TRC.ico
This is the TRC Development icon.

TRC_TCP.cpp
This is example source code, which gives a few examples on how you could drive the SimKits devices with your own software. This code handles the communication layer.

TRC_TCP.h
This file contains the declarations for the TRC_TCP.cpp file.

Unit1.cpp
This is example source code, which gives a few examples on how you could drive the SimKits devices with your own software. This code handles the user interface.

Unit1.dfm
This file contains information about the layout and names of program items such as buttons, text, etc.

Unit1.h
This file contains the declarations for the Unit1.cpp file.

Unit2.cpp
This is source code for the settings dialog.

Unit2.dfm
This file contains information about the layout and names of program items such as buttons, text, etc.

Unit2.h
This file contains the declarations for the Unit2.cpp file

The files below will be installed in the directory:

C:\Program Files\TRC Development\Radio Stack SDK\Examples\Microsoft Visual C++

ChildView.cpp
This is example source code, which gives a few examples on how you could drive the SimKits devices with your own software. This code handles the user interface.

ChildView.h
This file contains the declarations for the ChildView.cpp file.

Instruments.cpp
This is source code for the list of available SimKits devices.

Instruments.h
This file contains the declarations for the Instruments.cpp file.

MainFrm.cpp
This file contains C++ source code for the general application.

MainFrm.h
This file contains the declarations for the MainFrm.cpp file.

Options.cpp

This is source code for the options menu.

Options.h
This file contains the declarations for the Options.cpp file.

resource.h
This file contains the declarations for the layout of the program.

StdAfx.cpp
This file contains source code which points to required external resources.

StdAfx.h
This file contains the declarations for the StdAfx.cpp file.

TRC_Radio Stack SDK.cpp
This file contains C++ source code for the general application.

TRC_Radio Stack SDK.dsp
This file contains information about the settings of the project

TRC_Radio Stack SDK.dsw
This file contains information about the settings of the project

TRC_Radio Stack SDK.h
This file contains the declarations for the TRC_Radio Stack SDK.cpp file.

TRC_Radio Stack SDK.rc
This file contains information about the icons, bitmaps, etc.

TRC_TCP.cpp
This is example source code, which gives a few examples on how you could drive the SimKits devices with your own software. This code handles the communication layer.

TRC_TCP.h
This file contains the declarations for the TRC_TCP.cpp file.

TRC_Radio Stack SDK.clw
This file contains information for the ClassWizard of Microsoft Visual C++

The file below will be installed in the directory:

C:\Program Files\TRC Development\Radio Stack SDK\Examples\Microsoft Visual C++\Debug

RADIOsampleVC.exe
This is the example program.

The files below will be installed in the directory:

C:\Program Files\TRC Development\Radio Stack SDK\Examples\Microsoft Visual C++\res

TRC.ico
This is the TRC Development icon.

RADIO_Radio Stack SDK.exe.manifest
This file contains information about the settings of the project.

TRC_Radio Stack SDK.rc2
This file contains information about the icons, bitmaps, etc.

The files below will be installed in the directory:

C:\Program Files\TRC Development\Radio Stack SDK\Examples\Microsoft Visual Basic

RADIOsampleVB.exe
This is the example program.

Dialog.frm
This file contains information about the layout of the settings dialog.

Form1.frm
This file contains information about the layout of the main program.

Form1.frx
This file contains supplemental information for Form1.frm

Project1.vbp
This file contains information about the settings of the project.

Project1.vbw
This file contains information about the settings of the project.

TRC_TCP.bas
This is example source code, which gives a few examples on how you could drive the SimKits devices with your own software.

TRC_Radio Stack SDK.bas
This file contains C++ source code for the general application.

The files below will be installed in the directory:

C:\Program Files\TRC Development\Radio Stack SDK\Examples\Borland Delphi

Project1.dof
This file contains information about the settings of the project.

Project1.dpr
This file contains information about the settings of the project.

RADIOsampleD.exe
This is the example program.

Project1.res
This file contains information about the icons, bitmaps, etc.

TRC.ico
This is the TRC Development icon.

TRC_TCP.pas
This is example source code, which gives a few examples on how you could drive the SimKits devices with your own software. This code handles the communication layer.

Unit1.dfm
This file contains information about the layout of the main program.

Unit1.pas
This file contains C++ source code for the general application.

Unit2.dfm
This file contains information about the layout of the settings dialog.

Unit2.pas
This file contains C++ source code for the settings dialog.

7. Installing the USB driver software

The USB driver software is installed by connecting the RSC to your PC.

Your PC will recognize a new hardware product and wants to install a driver for it.

Make the following steps:

Step 1:
Connect the RSC to the power supply. Make sure the power supply is switched on.

Step 2:
Connect the USB cable to the RSC and connect the USB cable to your computer.

Your screen will show a message that new USB hardware has been detected. Immediately thereafter the “New hardware found” screen comes up. Now click **Next**.

Click on “Search for the best driver for your device. (Recommended).”

Specify a location, choose the folder *C:\Program Files\TRC Development\Radio Stack SDK* (or the folder where you extracted the **trcdrv.sys** and **.inf** files in)

Now click next to install the software.

If the software is installed properly, the following screen is shown.

This concludes the installation of the USB hardware driver. Now you are ready to install the Interface Software called RSCcustom.

8. Use of IP address and Port settings

The IP address and port number used in your application and the **RSCcustom.exe** program must be configured correctly, otherwise the Radio Stack SDK software won't work.

The IP address used in your application must match the IP address of the computer where the RSC and RSCCustom.exe are installed.

The port number used for RSCCustom.exe must match the port number used for your application, to allow communication between the two programs.

These settings depend on your hardware configuration. Below you'll find examples on possible settings.

If you have the RSC installed at the same computer as where your application runs on:

Use *127.0.0.1* as IP address for your application. The **RSCcustom.exe** program and your application must always have the same port number configured. The default port number is *1929.* If this port is in use (which is unlikely), then try a different number.

If you have the RSC board and RSCCustom.exe installed at a different computer as where your application runs you must use the IP address of the computer where the RSC is installed.

You can find the IP address of a computer with tools like winipcfg or ipconfig. Use the windows manual if you don't know how to find the IP address. The **RSCcustom.exe** program and your application always must have the same port number configured. The default port number is *1929.* If this port is in use (which is unlikely), then try a different number.

Warning: The RSCcustom.exe program must always run at the same computer as where your RSC (Radio Stack Controller) is installed.

9. Instruments Variable names and I/O connection

The *Read/Write* row indicates whether the device can be read or written with the software.

Note: values may be written including up to 3 decimals.

Device Variable Name	Lowest Value	Highest Value	Read/ Write	Information
adf_active	100	1700	RW	Mhz
adf_disable	0	1	RW	on - off
adf_stby	100	1700	RW	Mhz
adf_volume	0	1023	R	volume
alti_height	-1000	99000	RW	ft
audio_adf	0	1	RW	on - off
audio_aux	0	1	RW	on - off
audio_com1	0	1	RW	on - off
audio_com2	0	1	RW	on - off
audio_disable	0	1	RW	on - off
audio_dme	0	1	RW	on - off
audio_ics	0	1	RW	on - off
audio_indicatori	0	1	RW	on - off
audio_indicatorm	0	1	RW	on - off
audio_indicatoro	0	1	RW	on - off
audio_intercommode 1)	1	3	R	position
audio_markermode 2)	1	3	R	position
audio_mic 3)	1	6	R	position
audio_mkr	1	3	RW	on - off
audio_nav1	0	1	RW	on - off
audio_nav2	0	1	RW	on - off
audio_spr	0	1	RW	on - off
audio_volume	0	1023	R	volume
autopilot_alt	0	99900	RW	ft
autopilot_altlock	0	1	RW	on - off
autopilot_ap	0	1	RW	on - off
autopilot_apr	0	1	RW	on - off
autopilot_disable	0	1	RW	on - off
autopilot_hdg	0	1	RW	on - off
autopilot_nav	0	1	RW	on - off
autopilot_rev	0	1	RW	on - off
autopilot_vs	-9900	9900	RW	ft/min
comm1_comactive	118	135.95	RW	Mhz
comm1_comreceive	0	1	RW	on - off
comm1_comstby	118	135.95	RW	Mhz
comm1_comtransmit	0	1	RW	on - off
comm1_comvolume	0	1023	R	volume
comm1_disable	0	1	RW	on - off
comm1_navactive	118	117.95	RW	Mhz
comm1_navlocalizer	-10	10	RW	position

Device Variable Name	Lowest Value	Highest Value	Read/ Write	Information
comm1_navobs	0	360	RW	degrees
comm1_navstby	108	117.95	RW	Mhz
comm1_navtofrom 4)	0	2	RW	position
comm1_navvolume	0	1023	R	volume
comm2_comactive	118	135.95	RW	Mhz
comm2_comreceive	0	1	RW	on - off
comm2_comstby	118	135.95	RW	Mhz
comm2_comtransmit	0	1	RW	on - off
comm2_comvolume	0	1023	R	volume
comm2_disable	0	1	RW	on - off
comm2_navactive	108	117.95	RW	Mhz
comm2_navlocalizer	-10	10	RW	position
comm2_navobs	0	359	RW	degrees
comm2_navstby	108	117.95	RW	Mhz
comm2_navtofrom 5)	0	2	RW	position
comm2_navvolume	0	1023	R	volume
dme_disable	0	1	RW	on - off
dme_distance	0	99.9	RW	nm
dme_freq	108	118	RW	Mhz
dme_select	1	2	R	DME Radio
dme_speed	0	999	RW	kts
extraknob_adfhdg	0	360	RW	degrees
extraknob_altipress	28.1	31.6	RW	inch Hg
extraknob_hdgbug	0	360	RW	degrees
extraknob_hdgdrift	0	360	RW	degrees
extraknob_vor1obs	0	360	RW	degrees
extraknob_vor2obs	0	360	RW	degrees
transp_disable	0	1	RW	on - off
transp_mode 6)	1	6	R	position
transp_squawk	0	7777	RW	code

1) audio_intercommode
Value 1 = Pilot
Value 2 = Crew
Value 3 = All

2) audio_markermode
Value 1 = TEST
Value 2 = LO
Value 3 = HI

3) audio_mic
Value 1 = COM3
Value 2 = COM2
Value 3 = COM1
Value 4 = COM1/2
Value 5 = COM2/1
Value 6 = TEL

4) comm1_navtofrom
Value 0 = Unknown
Value 1 = Up
Value 2 = Down

5) comm2_navtofrom
Value 0 = Unknown
Value 1 = Up
Value 2 = Down

6) transp_mode
Value 1 = ALT
Value 2 = ON
Value 3 = TST
Value 4 = SBY
Value 6 = OFF

Error codes

A function always returns an error code consisting of an integer number.

The table below explains what each code means

TRC_SendData()
TRC_RecvData()

-1 = error: general error
 0 = no error
 1 = error: board not connected
 2 = error: device not calibrated
 3 = error: wrong command syntax
 4 = error: wrong variable syntax
 5 = error: wrong value syntax
 6 = error: too few parameters
 7 = error: too many parameters
 8 = error: device not writeable

TRC_Startup()
0 = no error
This function returns *winsock* error codes. Search the net for what these codes mean.

TRC_Cleanup()
0 = no error
This function returns *winsock* error codes. Search the net for what these codes mean.

10. Communication Protocol description

When you do not use the source code examples included in the Radio Stack SDK to write your application, you need to know the protocol used to access RSCcustom.exe.

It means that you have sufficient knowledge of TCP/IP programming.

Reading
To read data from a SimKits device (for example the airspeed gauge), send the following string over TCP/IP:

`"get dme_distance"`
(the two words are separated by a normal space, not an underscore):

Immediately after you send this string, you will receive a reply over TCP/IP. What you will get, is either:

`"dme_distance 101"`
(the airspeed gauge has the value 101)

Or you might get:

`"error 1"`
(this means that the controller board is not connected. See error codes in chapter 9)

Writing
To write data to a SimKits device (for example the dme_distance), send the following string over TCP/IP:

`"set dme_distance 21"`

Immediately after you send this string, you will receive a reply over TCP/IP. What you will get, is either:

`"ok"`
*(t*his means that no error occurred)

Or you might get:

`"error 1"`

(this means that the controller board is not connected. See error codes in chapter 9)

If you need to send a number including decimals, make sure you use a period as a decimal separator, not a comma. (for example, you can send "set dme_distance 21.05").

Received number strings can include decimals.

It is recommended that your program always checks for returned messages to check for possible errors.

11. Precautions when handling the RSC

Please read the information below carefully before taking the RSC out of the antistatic protective plastic bag:

WARNING:

The Radio Stack Controller is a delicate piece of electronics. Please take precautions when touching the board. It may be damaged by static electricity!

1. Leave the Radio Stack Controller in its protecting antistatic bag as long as possible. Never leave it in or on top of the antistatic bag when power is connected or when the USB cable is connected. This may cause the board to be destroyed or your PC may be damaged!
2. Discharge yourselves - before touching the board - to something grounded.
3. Avoid touching the integrated circuits and other parts mounted on the board by handling the board by the edges.
4. The connectors attached to the instruments can only be connected one way, due to a positioning notch on each connector on the Radio Stack Controller as well as on the flat cable of the instruments.
5. Connect the instruments one by one to the Radio Stack Controller, before connecting the power supply and USB cable.
6. After connecting the instruments, first connect the power supply by using a standard AT Power supply and utilizing the disk drive power connector from the AT Power Supply.
7. Watch the small LED (red indicator) on the Radio Stack Controller, mounted between the Power Connector and the USB connector. This LED has the following conditions:

The LED is not lighted

- Power is off or:
- Power is connected and there is no USB connection to the PC.

The LED is steady On (does not flash)

- There is a USB connection to the computer, but the firmware (internal software for the Central Control Unit) is not yet loaded from the PC. You have to start the driver software (TRCLINK.EXE).

The LED Flashes

- Power is connected and the firmware is loaded into the Radio Stack Controller memory.

12. RSC Hardware Specifications

The data below is only for information purpose. It is not recommended to connect any other hardware to the RSC than the TRC Radio Stack components.
If you connect custom hardware to the RSC control board, you can damage the RSC or your own hardware .

The chips, which are connected to the input and output pins of the RSC board, are designed to drive low voltage - low amperage circuits only. You cannot connect for example leds, lamps, or relays to the RSC. This will most certainly destroy the RSC board.

-Digital inputs and outputs:

SYMBOL	PARAMETER	MIN	MAX	UNIT	CONDITIONS
V_{CC}	DC supply voltage	-0.5	+7	V	
$\pm I_{IK}$	DC input diode current		20	mA	for $V_I < -0.5$ or $V_I > V_{CC} + 0.5$ V
$\pm I_{OK}$	DC output diode current		20	mA	for $V_O < -0.5$ or $V_O > V_{CC} + 0.5$ V
$\pm I_O$	DC output source or sink current standard outputs bus driver outputs		 25 35	 mA mA	for -0.5 V < V_O < V_{CC} + 0.5 V

Chip supply voltage (Vcc) is 5V.
Chip input voltage (logical 1) must be 5V. or minimum 3.5V.
Chip input voltage (logical 0) must be 0V. or maximum 0.7V.
All digital inputs have a pull up resistor of 10K to Vcc.

The RSC is a delicate device and a shortage or other mistreatment can cause damage to the board easily. Such damages are not covered under warranty and will usually need a total replacement of the board.

13. Performing your first tests

It is assumed that you have successfully installed the Radio Stack SDK software and all its files on your computer. As you may have noticed, the installation has saved a number of files on your PC. Not all files are necessary for your programming. Some files are only necessary for a certain programming environment.

However, these additional files for each programming language are so small (a few hundred kilobytes maximum in general) that they will not fill-up your hard drive and it is recommended to leave them on your PC.

It is also assumed, that you have successfully connected the RSC board to a PC AT Power Supply, that you have connected the RSC Board to your PC and that your operating system has asked for the proper USB driver, which has been installed successfully.

First of all you need to checks if the TCP/IP protocol is installed on your PC, prior to using the Radio Stack SDK:

To check if the TCP/IP protocol is installed, you can perform the following checks:

Windows 98:
At: *"Start -> Settings -> Control Panel -> Network".* Inside the list of installed devices, the TCP/IP protocol must be present. If it is not, consult the Windows manual on how to install the TCP/IP protocol.

Windows 2000:
At: *"Start -> Settings -> Control Panel -> Network and Dial-Up Connections ",* right click your network device, and select *properties*. Inside the list of installed devices, the TCP/IP protocol must be present. If it is not, consult the windows manual on how to install the TCP/IP protocol.

Windows NT:
At: *"Start -> Settings -> Control Panel -> Network"* click on the *"Protocols"* tab. Inside the list of installed devices, the TCP/IP protocol must be present. If it is not, consult the windows manual on how to install the TCP/IP protocol.

Windows XP:
At: *"Start -> Settings -> Control Panel -> Network Connections"* right click your network device, and select *properties*. Inside the list of installed devices, the TCP/IP protocol must be present. If it is not, consult the windows manual on how to install the TCP/IP protocol.

The Radio Stack SDK software is the RSCcustom.exe. This program must be running when you want to run a program made using the Radio Stack SDK. Run **RSCcustom.exe** always at the same computer as where the RSC is installed. Be sure to enter the correct IP address and port number. The *IP* address and port number can be changed in every Radio Stack SDK program in *File -> Options.*

Now compile the example project supplied for your programming environment (RADIOsampleD.exe, RADIOsampleVB.exe, RADIOsampleVC.exe or RADIOsampleCB.exe)

When running the example program dialog, there is a *drop down list* where you can select a device. Select the device, which you must have calibrated before. Enter a number in the *edit box* at the right of the *set* button. Now press *set*. The device should indicate the number you

selected. If it does, you are ready to use the Radio Stack SDK. If the device does not indicate the correct value, go through the install steps again to see if you missed something.

14. Programming examples for Microsoft Visual C++

Example 1 - Driving a device - Startup

Before you can drive a device, a call to the startup function must be made once. This is for initializing all necessary components.

```
error = TRC_Startup(IPaddress, portnumber);
```

- Return value: `"error"`
- Data type: `int`
This return value contains an error code.
(Error codes can be found in chapter 9)

- First function variable: `"IPaddress"`
- Data type: `*char`
This is the IP address of the PC where the data shall be send to (and may be the same computer). Use standard windows tools (like winipcfg or ipconfig) to find the IP address of a PC. If the PC where your application runs on is the same as where the RSC is installed, then use the local host address, which is 127.0.0.1

- Second function variable: `"portnumber"`
- Data type: `unsigned short`
This is the port number. Default port number is 1929

Code example

```
int error = 0;
error = TRC_Startup("127.0.0.1", 1929);
```

Code example 2

```
int error = 0;
char *IPaddress;
unsigned short portnumber = 1929;
IPaddress = "127.0.0.1";
error = TRC_Startup(IPaddress, portnumber);
```

Example 2 - Driving a device - Sending Data

Once you have called the startup function, you can use the functions for sending information to the SimKits Devices. This function will send data to a device once:

```
error = TRC_SendData(instrument_name, instrument_value);
```

- Return value: `"error"`
- Data type: `int`

This return value contains an error code. If the function returns a valid error, then the state of the SimKits device will be undefined.
(Error codes can be found in chapter 9)

- First function variable: `"instrument_name"`
- Data type: `char*`

This is the name of the device to which you want to send a value. Each device has a pre-defined name. A table of these names is included in chapter 9.

- Second function variable: `"instrument_value"`
- Data type: `float`

This is the value, representing the movement positions or readable output of the device. You must use a value, which is of a valid magnitude for the appropriate device. For example, for the dme_distance, you can send a value between 0 and 99.9 (nm), but for the dme_speed, any value greater then 999 is incorrect and will give a wrong reading. A table with valid values for each device is included at chapter 9.

Code example

```
int error = 0;
char *instrument_name;
float instrument_value = 15;
instrument_name = "dme_distance";
error = TRC_SendData(instrument_name, instrument_value);
```

Code example

```
float instrument_value1 = 0;
float instrument_value2 = 0;
float instrument_value3 = 0;
/* If you want to update all of the devices with a smooth movement,
call these functions at least 20 times a second */
TRC_SendData("dme_distance", instrument_value1);
TRC_SendData("dme_speed", instrument_value2);
TRC_SendData("adf_active", instrument_value3);
// etc.
```

Example 3 - Driving a device - Receiving Data

This function enables you to receive information from the devices. The function reads a device once.

```
error = TRC_ RecvData(instrument_name, &instrument_value);
```

- Return value: `"error"`
- Data type: `int`

This return value contains an error code. If the error code is valid, then the value of `"instrument_value"` will be invalid.

- First function variable : `"instrument_name"`
- Data type: `char*`

This is the name of the device from which you want to read a value.
(See chapter 9 for valid device names.)

- Second function variable : `"instrument_value"`
- Data type: `float`

The function returns the value of the device to this variable. After this call, the variable `"instrument_value"` contains an updated value of the device when the error code returned was a non-error.

Code example

```
int error = 0;
char *instrument_name;
float instrument_value = 0;
instrument_name = "dme_distance";
error = TRC_RecvData(instrument_name, &instrument_value);
```

Code example

```
float instrument_value1 = 0;
float instrument_value2 = 0;
float instrument_value3 = 0;
// Read more then one device
TRC_RecvData("dme_distance", &instrument_value1);
TRC_RecvData("dme_speed", &instrument_value2);
TRC_RecvData("adf_active", &instrument_value3);
// etc.
```

Example 4 - Driving a device - Cleanup

Before you end your program, the function "TRC_Cleanup" must be called to free all used resources.

```
error = TRC_Cleanup();
```

- Return value: `"error"`
- Data type: `int`

This return value contains an error code.

Code example:

```
int error = 0;
error = TRC_Cleanup();
```

15. Programming examples for Borland C++ Builder

Example 1 - Driving a device - Startup

Before you can drive a device, a call to the startup function must be made once. This is for initializing all necessary components.

```
error = TRC_Startup(IPaddress, portnumber);
```

- Return value: `"error"`
- Data type: `int`
This return value contains an error code.
(Error codes can be found in chapter 9)

- First function variable: `"IPaddress"`
- Data type: `*char`
This is the IP address of the PC where the data shall be send to (and may be the same computer). Use standard windows tools (like winipcfg or ipconfig) to find the IP address of a PC. If the PC where your application runs on is the same as where the RSC is installed, then use the local host address, which is 127.0.0.1

- Second function variable: `"portnumber"`
- Data type: `unsigned short`
This is the port number. Default port number is 1929

Code example

```
int error = 0;
error = TRC_Startup("127.0.0.1", 1929);
```

Code example

```
int error = 0;
char *IPaddress;
unsigned short portnumber = 1929;
IPaddress = "127.0.0.1";
error = TRC_Startup(IPaddress, portnumber);
```

Example 2 - Driving a device - Sending Data

Once you have called the startup function, you can use the functions for sending information to the SimKits Devices. This function will send data to a device once:

```
error = TRC_SendData(instrument_name, instrument_value);
```

- Return value: `"error"`
- Data type: `int`

This return value contains an error code. If the function returns a valid error, then the state of the SimKits device will be undefined.
(Error codes can be found in chapter 9)

- First function variable: `"instrument_name"`
- Data type: `char*`

This is the name of the device to which you want to send a value. Each device has a pre-defined name. A table of these names is included in chapter 9.

- Second function variable: `"instrument_value"`
- Data type: `float`

This is the value, representing the movement positions or readable output of the device. You must use a value, which is of a valid magnitude for the appropriate device. For example, for the dme_distance, you can send a value between 0 and 99.9 (nm), but for the dme_speed, any value greater then 999 is incorrect and will give a wrong reading. A table with valid values for each device is included at chapter 9.

Code example

```
int error = 0;
char *instrument_name;
float instrument_value = 15;
instrument_name = "dme_distance";
error = TRC_SendData(instrument_name, instrument_value);
```

Code example

```
float instrument_value1 = 0;
float instrument_value2 = 0;
float instrument_value3 = 0;
/* If you want to update all of the devices with a smooth movement,
call these functions at least 20 times a second */
TRC_SendData("dme_distance", instrument_value1);
TRC_SendData("dme_speed", instrument_value2);
TRC_SendData("adf_active", instrument_value3);
// etc.
```

Example 3 - Driving a device - Receiving Data

This function enables you to receive information from the devices. The function reads a device once.

```
error = TRC_ RecvData(instrument_name, &instrument_value);
```

- Return value: `"error"`
- Data type: `int`

This return value contains an error code. If the error code is valid, then the value of `"instrument_value"` will be invalid.

- First function variable : `"instrument_name"`
- Data type: `char*`

This is the name of the device from which you want to read a value.
(See chapter 9 for valid device names.)

- Second function variable : `"instrument_value"`
- Data type: `float`

The function returns the value of the device to this variable. After this call, the variable `"instrument_value"` contains an updated value of the device when the error code returned was a non-error.

Code example

```
int error = 0;
char *instrument_name;
float instrument_value = 0;
instrument_name = "dme_distance";
error = TRC_RecvData(instrument_name, &instrument_value);
```

Code example

```
float instrument_value1 = 0;
float instrument_value2 = 0;
float instrument_value3 = 0;
// Read more then one device
TRC_RecvData("dme_distance", &instrument_value1);
TRC_RecvData("dme_speed", &instrument_value2);
TRC_RecvData("adf_active", &instrument_value3);
// etc.
```

Example 4 - Driving a device - Cleanup

Before you end your program, the function "TRC_Cleanup" must be called to free all used resources.

```
error = TRC_Cleanup();
```

- Return value: `"error"`
- Data type: `int`

This return value contains an error code.

Code example

```
int error = 0;
error = TRC_Cleanup();
```

16. Programming examples for Microsoft Visual Basic

Example 1 - Driving a device - Startup

Before you can drive a device, a call to the startup function must be made once. This is for initializing all necessary components.

```
Call TRC_Startup(Winsock1, IPaddress, portnumber)
```

- First function variable: `"Winsock1"`
- Data type: `Winsock`

This is the name of the winsock component you placed on top of your form.

- Second function variable: `"IPaddress"`
- Data type: `String`

This is the IP address of the PC where the data shall be send to (and may be the same computer). Use standard windows tools (like winipcfg or ipconfig) to find the IP address of a PC. If the PC where your application runs on is the same as where the RSC is installed, then use the local host address, which is 127.0.0.1

-Third function variable: `"portnumber"`
- Data type: `Integer`

This is the port number. Default port number is 1929

Code example

```
Call TRC_Startup(Winsock1, "127.0.0.1", 1929)
```

Code example

```
Dim IPaddress As String
IPaddress = "127.0.0.1"
Dim port As Integer
portnumber = 1929
Call TRC_Startup(Winsock1, IPaddress, portnumber)
```

Example 2 - Driving a device - Sending Data

Once you have called the startup function, you can use the functions for sending information to the SimKits Devices. This function will send data to a device once:

```
error = TRC_SendData(instrument_name, instrument_value)
```

-Return value: `"error"`
-Data type: `Long`
This return value contains an error code. If the function returns a valid error, then the state of the SimKits device will be undefined.
(Error codes can be found in chapter 9)

-First function variable: `"instrument_name"`
-Data type: `String`
This is the name of the device to which you want to send a value. Each device has a pre-defined name. A table of these names is included in chapter 9.

-Second function variable : `"instrument_value"`
-Data type: `Single`
This is the value, representing the movement positions or readable output of the device. You must use a value, which is of a valid magnitude for the appropriate device. For example, for the dme_distance, you can send a value between 0 and 99.9 (nm), but for the dme_speed, any value greater then 999 is incorrect and will give a wrong reading. A table with valid values for each device is included at chapter 9.

Code example

```
Dim error As Long
Dim instrument_name As String
Dim Instrument_value As Single
instrument_value = 15
instrument_name = "dme_distance"
error = TRC_SendData(instrument_name, instrument_value)
```

Code example

```
Dim Instrument_value1 As Single
Dim Instrument_value2 As Single
Dim Instrument_value3 As Single
instrument_value1 = 12
instrument_value2 = 140
instrument_value3 = 1
' If you want to update all of the devices with a smooth movement,
call these functions at least 20
' times a second
TRC_SendData("dme_distance", instrument_value1)
TRC_SendData("dme_speed", instrument_value2)
TRC_SendData("adf_active", instrument_value3)
' etc.
```

Example 3 - Driving a device - Receiving Data

This function enables you to receive information from the devices. The function reads a device once.

```
error = TRC_ RecvData(instrument_name, instrument_value)
```

-Return value: "error"
-Data type: Long
This return value contains an error code. If the error code is valid, then the value of "instrument_value" will be invalid.

-First function variable : "instrument_name"
-Data type: String
This is the name of the device from which you want to read a value.
(See chapter 9 for valid device names.)

-Second function variable : "instrument_value"
-Data type: Single
The function returns the value of the device to this variable. After this call, the variable "instrument_value" contains an updated value of the device when the error code returned was a non-error.

Code example

```
Dim error As Long
Dim instrument_name As String
Dim Instrument_value As Single
instrument_name = "dme_distance"
error = TRC_RecvData(instrument_name, instrument_value)
```

Code example

```
Dim Instrument_value1 As Single
Dim Instrument_value2 As Single
Dim Instrument_value3 As Single
' Read more then one device
TRC_RecvData("dme_distance", instrument_value1)
TRC_RecvData("dme_speed", instrument_value2)
TRC_RecvData("adf_active", instrument_value3)
' etc.
```

Example 4 - Driving a device - Cleanup

Before you end your program, the function "TRC_Cleanup" must be called to free all used resources.

Code example:

```
Call TRC_Cleanup()
```

17. Programming examples for Borland Delphi

Example 1 - Driving a device - Startup

Before you can drive a device, a call to the startup function must be made once. This is for initializing all necessary components.

```
error := TRC_Startup(IPaddress, portnumber);
```

-Return value: "error"
-Data type: Integer
This return value contains an error code.
(Error codes can be found in chapter 9)

-First function variable: "IPaddress"
-Data type: String
This is the IP address of the PC where the data shall be send to (and may be the same computer). Use standard windows tools (like winipcfg or ipconfig) to find the IP address of a PC. If the PC where your application runs on is the same as where the RSC is installed, then use the local host address, which is 127.0.0.1

-Second function variable: "portnumber"
-Data type: Word
This is the port number. Default port number is 1929

Code example 1:

```
var error : Integer = 0;
error := TRC_Startup('127.0.0.1', 1929);
```

Code example 2:

```
var error : integer = 0;
var IPaddress : String = '127.0.0.1';
var portnumber : Word = 1929;
error := TRC_Startup(IPaddress, portnumber);
```

Example 2 - Driving a device - Sending Data

Once you have called the startup function, you can use the functions for sending information to the SimKits Devices. This function will send data to a device once:

```
error := TRC_SendData(instrument_name, instrument_value);
```

-Return value: "error"
-Data type: Integer
This return value contains an error code. If the function returns a valid error, then the state of the SimKits device will be undefined.
(Error codes can be found in chapter 9)

-First function variable: "instrument_name"
-Data type: String
This is the name of the device to which you want to send a value. Each device has a pre-defined name. A table of these names is included in chapter 9.

-Second function variable: "instrument_value"
-Data type: Real
This is the value, representing the movement positions or readable output of the device. You must use a value, which is of a valid magnitude for the appropriate device. For example, for the dme_distance, you can send a value between 0 and 99.9 (nm), but for the dme_speed, any value greater then 999 is incorrect and will give a wrong reading. A table with valid values for each device is included at chapter 9.

Code example

```
var error : integer = 0;
var instrument_name : String = 'dme_distance';
var instrument_value : Real = 15;
error := TRC_SendData(instrument_name, instrument_value);
```

Code example

```
var instrument_value1 : Real = 0;
var instrument_value2 : Real = 0;
var instrument_value3 : Real = 0;
(* If U want to update all of the devices with a smooth movement, call
these functions at least 20 times a second *)
TRC_SendData('dme_distance', instrument_value1);
TRC_SendData('dme_speed', instrument_value2);
TRC_SendData('adf_active', instrument_value3);
// etc.
```

Example 3 - Driving a device - Receiving Data

This function enables you to receive information from the devices. The function reads a device once.

```
error := TRC_ RecvData(instrument_name, instrument_value);
```

-Return value: "error"
-Data type: Integer
This return value contains an error code. If the error code is valid, then the value of "instrument_value" will be invalid.

-First function variable : "instrument_name"
-Data type: String
This is the name of the device from which you want to read a value.
(See chapter 9 for valid device names.)

-Second function variable : "instrument_value"
-Data type: Real
The function returns the value of the device to this variable. After this call, the variable "instrument_value" contains an updated value of the device when the error code returned was a non-error.

Code example

```
var error : Integer = 0;
var instrument_name : String = 'dme_distance';
var instrument_value : Real = 0;
error := TRC_RecvData(instrument_name, instrument_value);
```

Code example

```
var instrument_value1 : Real = 0;
var instrument_value2 : Real = 0;
var instrument_value3 : Real = 0;
// Read more then one device
TRC_RecvData('dme_distance', instrument_value1);
TRC_RecvData('dme_speed', instrument_value2);
TRC_RecvData('adf_active', instrument_value3);
// etc.
```

Driving the devices - Cleanup

Before you end your program, the function "TRC_Cleanup" must be called to free all used resources.

```
error := TRC_Cleanup();
```

-Return value: `"error"`
-Data type: `Integer`
This return value contains an error code.

Code example

```
var error : Integer = 0;
error := TRC_Cleanup();
```

The TRC Radio Stack and its accompanying software is a product of:

Company Address:
TRC Development b.v.
Stationsweg 39
4241 XH ARKEL
THE NETHERLANDS

Postal Address:
TRC Development b.v.
P.O.Box 544
4200 AM GORINCHEM
THE NETHERLANDS

All Products carry a limited warranty of 12 months after purchase.
Please look for warranty text on our website: www.therealcockpit.com
All information requests and support requests can be directed to:
support@therealcockpit.com

Support is only given via email. It is our intention to react on each email within one working day.

Our office hours are:
Monday till Friday 09:00AM till 05:00PM (09:00-17:00) Central European Time.
(03:00AM – 02:00PM Eastern Time)

Phone: (from USA): 011 31 183 562 522, (from Europe): 0031 183 562 522
Fax: (from USA): 011 31 183 564 268, (from Europe): 0031 183 564 268

Shipping
Our main shipper is UPS for overnight delivery. For normal shipments we use TPG.

Order taking
We accept most credit cards. When ordering products, your credit card is only charged at the day of shipping. Our secure credit card handling over the internet is done by Bibit (www.bibit.com).

Privacy Statement
TRC Development will never use your email address for any other purpose than informing you about new products or other breaking news on The Real Cockpit products or software. When you do not wish to receive information by email, you can remove yourself from the mailing list at any time. Please see: www.therealcockpit.com.

Returning goods for repair or replace
Whenever you would like to return a product for repair or replace (at the choice of TRC Development b.v.) you will need a so-called RMA Number. RMA Numbers can be requested by email or by fax.

See our website support area for details on how to return a product. There you find a form which – when information is entered – will supply you with the proper RMA number:
http://www.therealcockpit.com/support/index.php?boxaction=support_return