Software Development Kit

for the following
Software Languages/Environments:

Microsoft Visual C++

Borland C++ Builder

Microsoft Visual Basic

Borland Delphi

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

TRC Development b.v.
Rotterdam Airport
The Netherlands
www.therealcockpit.com

Version 1.0 - July 2003

List of Contents

1. Introduction

SimKits are realistic Flight Simulator Instruments which are controlled by an electronics interface board called CCU and software drivers, developed by TRC Development b.v.

From the beginning of the introduction of the SimKits products, there are drivers available which connect directly with Microsoft Flight Simulator (also using FSUIPC, written by Pete Dowson).

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

To use the SDK, you must be a programmer. It is beyond the scope of this manual and beyond SimKits Support 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 SDK, there is no limit to use the SimKits instruments from any type of software written by yourself.

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

There is no need to write a calibration program yourself. Please use the existing calibration tool to calibrate your devices.

This 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 SDK variables for the different programming environments.

When you find a bug in the 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

Simkits has more documentation available, which we recommend strongly to read prior to the use of the 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 do SimKits work?

SimKits instruments and gauges are designed from scratch to emulate existing mechanical flight instruments. They are produced using plastic injection molding and made mainly from high quality ABS plastics.

The pointers and moving parts are controlled via (almost) standard servos as used in model airplanes or model racing cars. They are driven by dedicated electronics, the CCU.

Due to the very low forces inside the SimKits instruments, the (hobby) servo is a great electronic device and very well suitable for it's purpose. Practically only a few percent of its force is used to drive pointers and other moving mechanics in the SimKits instruments. Therefore the expected mechanical life is very long and the servos will last almost forever.

Although that standard servo mechanisms are used, not just any servo can be used in the SimKits, due to small mechanical differences and measurements between the brands on the market.

SimKits has choosen the Hitec HS300 or Hitec HS322 (or their exact equivalents) as driving servos.

Servos have the ability to move the outgoing axis for approx. over 180º. For most gauge applications, this is enough. The position of outgoing axis of the servo can be precisely positioned via a so-called pulse-width signal, provided by the CCU (Central Control Unit).

Some movements in a SimKits gauge however, expect a continous movement of over 360º. In this situation there is also a servo used, but the servo is (prior to mounting) modified in such a way, that a continous turn in clockwise and anti-clockwise direction can be achieved.

You can modify a servo yourselves, or buy one modified directly from SimKits.

Each SimKits instrument has a ribbon cable coming out. This ribbon cable connects to a certain connector on the CCU. Each individual pin on this connector supplies the necessary signals from and to a SimKits gauge.

The Central Control Unit (only available built and tested) is a Printed Circuit Board containing its own micro processor, memory and electronics to drive the servo motors, read out the dials and to control the instrument lighting. The Central Control Unit 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 SDK software is running.

The Central Control Unit 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 Central Control Unit has a very large number of I/O lines. There are 38 I/O connectors available to control 38 different instruments, switches, lights, etc. Some of these I/O connectors (like the analog outputs) are for future expansion and are not yet supported.

See also the separate CCU manual.

4. Principle of control

The SimKits instruments can be divided into 2 different classes of devices:

1. Controlled as a Device
2. Controlled by separate pin I/O

There are instruments which are controlled as a Device, which means that you cannot access or read from individual I/O lines for that device.

The reason for the division into 2 different types of instruments is obvious. As the development of the gauges drivers and its calibration software took the TRC Development team several man years, it would be very time consuming for you as a programmer to re-invent the wheel again.

For the gauges, controlled as a complete device, you benefit from special designed routines, built into the SDK correcting non-linearity of the servos, non-liniearity of the position sensors and fast read-back and write routines allowing the gauges to react as they do now. The communication update rate between gauges/CCU/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.

So how does it work exactly? For example, the Altimeter contains 2 servos. A standard servo and a modified servo. The modified servo drives the altimeter pointers via a gearw wheel mechanisme which is built like a clock (seconds, minutes, hours). However, unlike a lock, each pointer has to turn 10 times to have the sucessive pointer turning once. The movement of the fastest pointer, the middle pointer and the slowest pointer are measured by position sensors and photo interruptors.

The Altimeter also contains a knob, with which you can turn (by hand) a so-called Rotary Encoder. This Rotary Encoder cannot be read out via the SDK directly. The routines inside the SDK control this part of the movement completely, freeing you from the difficult job to write your own control routines for the servo of the pressure scale to move into the right direction and position.

The knob movement therefore is read out by the SDK software, interpreted and moves the pressure scale of the Altimeter, which values (the set pressure) you can read out. Using that information (together with other information from your flight simulator program) you can write the proper values to the Altimeter pointer.

Instruments controlled as a device

- Altimeter
- Airspeed Indicator
- Tacho meter
- Vertical Speed Indicator
- Attitude Indicator
- Heading Indicator
- Turn Coordinator
- ADF Indicator
- VOR 1 Indicator
- VOR 2 Indicator
- EGT/Fuel Flow Indicator
- Fuel Indicator Left/Right
- Warning Lights & Switch
- Oil Temp. & Press. Ind.
- Suction Gauge/Ammeter
- Wet Compass

Instruments which can be controlled from each pin on their connector
The tables in chapter 9 give information on which pins/signals can be controlled or read out directly via the SDK.

- Handbrake
- Avionics Switch
- Throttle
- Mixture
- Prop
- Master Switch
- Fuel selector + Shut off
- Flaps
- Trim wheel
- Starter Switch
- Rudder Pedals
- Yoke
- Autopilot Switches
- Circuit breakers (4 outputs, 15 inputs) [1)]
- Switches (16 inputs)

[1)] In order to drive the Circuit breakers, you need to decode the 4 Outputs present on the connector (CN 33) into 15 logical signals by means of a 4-to-16-line encoder. Output 0 of that encoder will then stay unconnected. The other 15 Outputs of the encoder need to control special hardware which enables a circuit breaker to pop out.

Ther are 15 inputs available on CN 33 to check if a circuit breaker is the IN or OUT position.

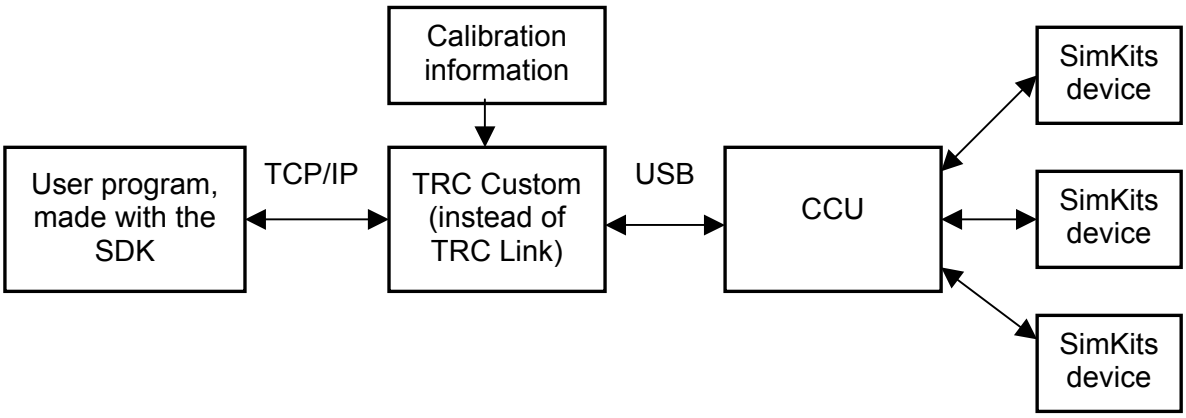
SimKits will provide in the near future especially designed hardware for the decoding

5. Communication between PC and SimKits gauges

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

The SDK software handles the TCP/IP communication layer, so using the SDK does not require indeep TCP/IP programming knowledge.

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



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

Although part the communication between your software and the 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 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 TRC Link software, which interfaces with FSUIPC from Pete Dawson. FSUIPC is the link between the TRC Link software and Microsoft Flight Simulator 2002/4.

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

6. Installing the SDK software

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

All necessary files to start using the SDK are packed in the file **SDKsetup.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 **driver.zip**, which need to be downloaded from the SimKits website software download area and unzipped into the directory:

C:\Program Files\TRC Development\SDK

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

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

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

TRCCustom.exe
This program is the SDK software.

Calibrations.cfg
This file contains the calibration information.

DelsL1.isu
This file is needed if you want to un-install the SDK

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

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

TRCPanel.cpl
This file is the calibration program. It is only accessible via the *control panel* and cannot run as an executable.

trccntl.drv
This is a driver file for the CCU 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\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.

TRCsampleCB.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 sourcecode, 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 sourcecode, 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 sourcecode 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\SDK\Examples\Microsoft Visual C++

ChildView.cpp
This is example sourcecode, 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 sourcecode 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 sourcecode 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 sourcecode which points to required external resources.

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

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

TRC_SDK.dsp
This file contains information about the settings of the project

TRC_SDK.dsw
This file contains information about the settings of the project

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

TRC_SDK.rc
This file contains information about the icons, bitmaps, etc.

TRC_TCP.cpp
This is example sourcecode, 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_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\SDK\Examples\Microsoft Visual C++\Debug

TRCsampleVC.exe
This is the example program.

The files below will be installed in the directory:
C:\Program Files\TRC Development\SDK\Examples\Microsoft Visual C++\res

TRC.ico
This is the TRC Development icon.

TRC_SDK.exe.manifest
This file contains information about the settings of the project.

TRC_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\SDK\Examples\Microsoft Visual Basic

TRCsampleVB.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 sourcecode, which gives a few examples on how you could drive the SimKits devices with your own software.

TRC_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\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.

TRCsampleD.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 sourcecode, 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 Central Control Unit 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 Central Control Unit to the power supply. Make sure the power supply is switched on.

Step 2:
Connect the USB cable to the Central Control Unit and connect the USB cable to your computer.

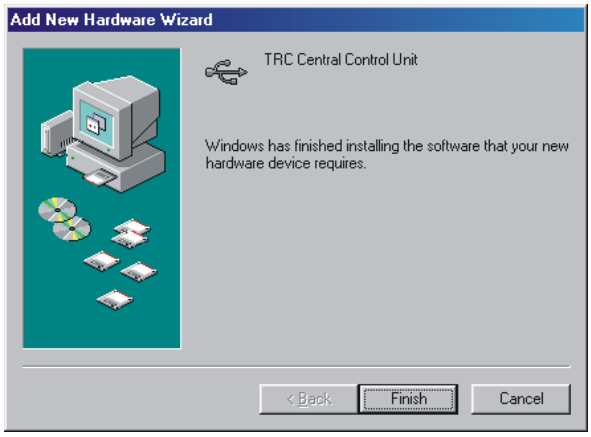
Your screen will show a message that new USB hardware has been detected. Immediately thereafter the “New hardware found” screen comes up. Now click **Next**.

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

Specify a location, choose the folder *C:\Program Files\TRC Development\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 Calibration Software and to install the Interface Software called TRC Custom.

8. Use of IP address and Port settings

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

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

The port number used for TRCCustom.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 CCU installed at the same computer as where your application runs on:

Use *127.0.0.1* as IP address for your application. The **TRCCustom.exe** program and your application must always have the same port number configured. The default port number is *1927.* If this port is in use (which is unlikely), then try a different number.

If you have the CCU and TRCCustom.exe installed at a different computer as where your application runs:

Use the IP address of the computer where the CCU 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 **TRCCustom.exe** program and your applicationalways must have the same port number configured. The default port number is *1927.* If this port is in use (which is unlikely), then try a different number.

Warning: The TRCCustom.exe program must always run at the same computer as where your CCU is installed.

9. Instruments Variable names and I/O connection

As explained in chapter 4, there are instruments which are controlled as a Device, which means that you cannot access or read from individual I/O lines for that device. For other instruments you can read or write directly to the pin of the connector of such a device.

The SimKits devices, which do not have a specified pin number cannot be custom made. Use the Original SimKits instruments for these devices. The *Read/Write* row indicates whether the device can be read or written with the software. The *I/O type* row indicates whether the hardware pin is a digital output (Dout), digital input (Din), analog input (Ain), or not specified (ns).

Some devices require some extra notes. This is indicated with a *1)* sign. You can find the correspondent note at the end of the variable types list. Please note that not all devices can be assigned a value, for example the throttle, it can only be read, not written. Some readable devices do not return the actual servo position, but the last value written to it. These are devices, which operate with a servo (a gauge) or a light (warning panel).

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

Instruments Controlled as a Device

Device Variable Name	Lowest Value	Highest Value	Read/ Write	Information
adf_compass	0	360	RW	degrees
adf_heading	0	360	R	degrees
airspeed	0	400	RW	knots
alti_height	0	100000	RW	feet
alti_pres_us	28.1	31.6	RW	inch Hg
alti_pres_metric	945	1050	RW	millibar
amp	-60	60	RW	ampere
attitude_bank	-90	90	RW	degrees
attitude_pitch	-25	25	RW	degrees
egt	312.1	712.1	RW	fahrenheit
fuelflow	0	19	RW	gallon/hour
fuellevel_left	0	26	RW	gallon
fuellevel_right	0	26	RW	gallon
vac [1)]	3	7	RW	inch Hg
heading_bug	0	360	R	degrees
heading_compass	0	360	RW	degrees
heading_correction [2)]	0	360	RW	degrees
oil_pres	0	115	RW	psi
oil_temp	75	245	RW	farenheit
tachometer	0	3500	RW	rpm
turn_inclino [3)]	-120	120	RW	position
turn_rate [4)]	-45	45	RW	degrees
verticalspeed	-2000	2000	RW	feet/min
vor1_compass	0	360	R	degrees
vor1_glideslope [5)]	-10	10	RW	position
vor1_nav [6)]	1	3	RW	position
vor1_gs [7)]	1	2	RW	position
vor1_localiser [8)]	-10	10	RW	position
vor2_compass	0	360	R	degrees
vor2_localiser [8)]	-10	10	RW	position
vor2_flag [9)]	1	3	RW	position
wetcompass	0	360	RW	degrees
warning_leftfuel [10)]	0	1	RW	on - off

warning_rightfuel [10)]	0	1	RW	on - off
Device Variable Name	**Lowest Value**	**Highest Value**	**Read/ Write**	**Information**
warning_fuel [10)]	0	1	RW	on - off
warning_bar [10)]	0	1	RW	on - off
warning_oilpress [10)]	0	1	RW	on - off
warning_leftvac [10)]	0	1	RW	on - off
warning_rightvac [10)]	0	1	RW	on - off
warning_vac [10)]	0	1	RW	on - off
warning_volts [10)]	0	1	RW	on - off

Instruments Controlled by pin I/O

Device Variable Name	Lowest Value	Highest Value	Read/ Write	Information	Connector/ Pin Number	I/O type
cb_avnfan [11)]	0	1	RW	on-off	CN33 pin 6	Din
cb_autopilot [11)]	0	1	RW	on - off	CN33 pin 7	Din
cb_gps [11)]	0	1	RW	on - off	CN33 pin 8	Din
cb_navcom1 [11)]	0	1	RW	on - off	CN33 pin 9	Din
cb_navcom2 [11)]	0	1	RW	on - off	CN33 pin 10	Din
cb_adf [11)]	0	1	RW	on - off	CN33 pin 11	Din
cb_xpndr [11)]	0	1	RW	on - off	CN33 pin 12	Din
cb_flap [11)]	0	1	RW	on - off	CN33 pin 13	Din
cb_inst [11)]	0	1	RW	on - off	CN33 pin 14	Din
cb_avnbus1 [11)]	0	1	RW	on - off	CN33 pin 15	Din
cb_avnbus2 [11)]	0	1	RW	on - off	CN33 pin 16	Din
cb_turncoord [11)]	0	1	RW	on - off	CN33 pin 17	Din
cb_instlts [11)]	0	1	RW	on - off	CN33 pin 18	Din
cb_altfld [11)]	0	1	RW	on - off	CN33 pin 19	Din
cb_warn [11)]	0	1	RW	on - off	CN33 pin 20	Din
switch_avionics	0	1	R	on - off	CN2 pin 2	Din
switch_masteralt	0	1	R	on - off	CN8 pin 2	Din
switch_masterbat	0	1	R	on - off	CN8 pin 3	Din
switch_fuelselector [12)]	1	4	R	on - off	CN13 pin 2-4	Din
switch_fuelcutoff	0	1	R	on - off	CN13 pin 5	Din
switch_starter [13)]	1	5	R	position	CN17 pin5-8	Din
switch_pushtotalk	0	1	R	on - off	CN34 pin 3	Din
switch_altstaticair	0	1	R	on - off	CN34 pin 4	Din
switch_fuelpump	0	1	R	on - off	CN34 pin 5	Din
switch_bcn	0	1	R	on - off	CN34 pin 6	Din
switch_land	0	1	R	on - off	CN34 pin 7	Din
switch_taxi	0	1	R	on - off	CN34 pin 8	Din
switch_nav	0	1	R	on - off	CN34 pin 9	Din
switch_strobe	0	1	R	on - off	CN34 pin 10	Din
switch_pitotheat	0	1	R	on - off	CN34 pin 11	Din
switch_extra1	0	1	R	on - off	CN34 pin 12	Din
switch_extra2	0	1	R	on - off	CN34 pin 13	Din
switch_extra3	0	1	R	on - off	CN34 pin 14	Din
switch_extra4	0	1	R	on - off	CN34 pin 15	Din
switch_extra5	0	1	R	on - off	CN34 pin 16	Din
switch_extra6	0	1	R	on - off	CN34 pin 17	Din
switch_extra7	0	1	R	on – off	CN34 pin 18	Din
trimwheel	0	1024	R	position	CN16 pin 4	Din
throttle	0	1024	R	position	CN4 pin 2	Ain
mixture	0	1024	R	position	CN5 pin 2	Ain

propadjust	0	1024	R	position	CN6 pin 2	Ain
Device Variable Name	**Lowest Value**	**Highest Value**	**Read/ Write**	**Information**	**Connector/ Pin Number**	**I/O type**
flap_lever	0	1024	R	position	CN15 pin 4	Ain
flap_indicator	0	1024	RW	position	CN15 pin 3	servo out
yoke_elevator	0	1024	R	position	CN19 pin 5	Ain
yoke_aileron	0	1024	R	position	CN19 pin 6	Ain
pedals_rudder	0	1024	R	position	CN18 pin 5	Ain
pedals_brake_l	0	1024	R	position	CN18 pin 6	Ain
pedals_brake_r	0	1024	R	position	CN18 pin 7	Ain
handbrake	0	1	R	position	CN1 pin 2	Din

Note: the “servo out” pin 3 on CN15 may drive a standard servo directly to drive an indicator for the flap position.

[1] *vac*
This is vacuum, not volts

[2] *heading_correction*
This is not the value of the rotary switch, but this value will be added to the "heading_compass" value.

[3] *turn_inclino*
This is the Turn Coordinator ball.

[4] *turn_rate*
This is the little airplane Indicator in the Turn Coordinator gauge.

[5] *vor1_glideslope*
Each bar is 5 units.

[6] *vor1_nav*
Value 1 = To
Value 2 = From
Value 3 = Unknown

[7] *vor1_gs*
Value 1 = GS
Value 2 = Unknown

One combination of the "vor1_nav" and "vor1_gs" flags is mechanically not possible:
vor1_gs = GS
vor1_nav = Unknown

[8] *vor1_localiser and* vor2_localiser
Each dot is 2 units.

[9] *vor2_flag*
Value 1 = To
Value 2 = From
Value 3 = Unknown

[10] *warning_...*
These are the text messages on the warning panel.

[11] *cb_...*

Currently, the circuit breaker variables are digital inputs only. In the future, outputs will become available with the use of a demultiplexer.

[12)] *switch_fuelselector*

	Pin number:
Value 1 = position Off	no pin connected
Value 2 = position L	CN13 pin 4 brought to Gnd, other pins no connection
Value 3 = position RCN13 pin 5	brought to Gnd, other pins no connection
Value 4 = position Both	CN13 pin 4 + 5 brought to Gnd, other pins no connection

[13)] *switch_starter*

	Pin number:
Value 1 = position Off	CN17 pin 5 brought to Gnd, other pins no connection
Value 2 = position R	CN17 pin 6 brought to Gnd, other pins no connection
Value 3 = position L	CN17 pin 7 brought to Gnd, other pins no connection
Value 4 = position Both	all pins no connection
Value 5 = position Start	CN17 pin 8 brought to Gnd, other pins no connection

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 SDK to write your application, you need to know the protocol used to access TRCCustom.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 airspeed"`
(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:

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

Or you might get:

`"error 5"`
(this means that the gauge is not calibrated. See error codes in chapter 9)

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

`"set airspeed 121"`
(the pointer of the gauge goes to position 121)

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 5"`

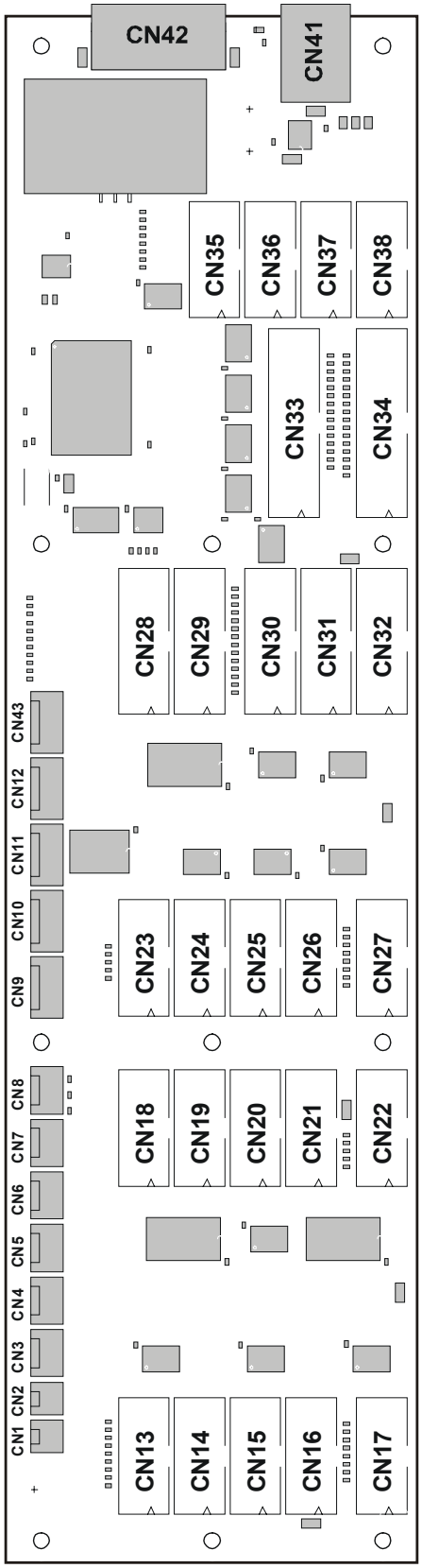
(this means that the gauge is not calibrated. See error codes in chapter 9)

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 airspeed 101.12").

Received number strings can include decimals.

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

11. Central Control Unit



List and position of the I/O connectors on the Central Control Unit

NOTE: The position of the texts on the picture on the left is not identical to the position of the text on the board itself.

CN1 Handbrake
CN2 Avionics Switch
CN3 Quartz Counter
CN4 Throttle
CN5 Mixture
CN6 Prop
CN7 Light Control Input
CN8 Master Switch
CN9 Analog Ctrld. Fuel Left/Right *)
CN10 Analog Ctrld. EGT/Fuel Flow*)
CN11 Analog Ctrld. Oil Temp/Pressure *)
CN12 Analog Ctrld. Suction G./Amindicator *)
CN13 Fuel selector + Shut off
CN14 Digital Clock
CN15 Flaps
CN16 Trim wheel
CN17 Starter Switch
CN18 Rudder Pedals
CN19 Yoke
CN20 Compass
CN21 Airspeed Indicator
CN22 Tachometer
CN23 Vertical Speed Indicator
CN24 Attitude Indicator
CN25 Turn Coordinator
CN26 VOR 1 Indicator
CN27 VOR 2 Indicator
CN28 ADF Indicator
CN29 Heading Indicator
CN30 Altimeter
CN31 Warning Lights & Switch
CN32 Autopilot Switch + Warnings
CN33 Circuit breakers (4 outputs, 15 inputs)
CN34 Switches (16 inputs)
CN35 Servo Ctrld. Fuel Left/Right *)
CN36 Servo Ctrld. EGT/Fuel Flow *)
CN37 Servo Ctrld. Oil Temp/Pressure *)
CN38 Servo Ctrld. Suction G./AMindicator *)

*) The small analog meters are available only with servo motors built in (CN35, 36, 37 and 38). The Analog Output controlled small analog meters (CN9, 10, 11 and 12) are not supported in the present software version. They also will not supported in future.

Note: All inputs are internally pulled-up to Vcc with a 10Kohm resistor on the CCU

12. Precautions when handling the CCU

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

WARNING:

The Central Control Unit is a delicate piece of electronics. Please take precautions when touching the board. It may be damaged by static electricity!

1. Leave the Central Control Unit 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 Central Control Unit as well as on the flat cable of the instruments.
5. Connect the instruments one by one to the Central Control Unit, 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 Central Control Unit, 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 ControlUnit) 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 Central Control Unit memory.

13. CCU Hardware Specifications

If you connect custom hardware to the CCU control board, you can damage the CCU or your own hardware if the hardware specifications are not taken into account!

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

-Analog inputs:

Peak input current is ±20 mA
Input voltage must be between 0v. and +5V. Do not apply a negative voltage!

-Analog outputs

Not supported in present software version.

If you want to use a digital output to drive a circuit, which draws more then *20 mA*, then connect the CCU output properly to a *transistor* or *darlington* circuit which in turn drives the high power circuit. Details about this topic are beyond the scope of this manual.
Please only connect custom electronic devices (not from SimKits) to the CCU when you have professional knowledge of electronics. The CCU is a delicated 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.

14. Performing your first tests

When you have built one of the SimKits gauges, you must first use the Calibration program to calibrate the gauge properly.

This process has to be carried out only once for each instrument and once again when you move the SimKits gauges to be driven from another computer.

The calibration parameters are saved in a file on your PC called **Calibrations.cfg and** reside in the directory *C:\Program Files\TRC Development\SDK* (or the directory where you installed the SDK files in). This config file is already created during installation and contains default values prior to calibration.

It is assumed that you have successfully installed the SDK software and all its files on your computer. As you may have noticed, the installation has save 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 harddrive and it is recommended to leave them on your PC.

It is also assumed, that you have sucessfully connected the CCU board to a PC AT Power Supply, that you have connected the CCU 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 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 SDK software is the TRCCustom.exe. This program must be running when you want to run a program made using the SDK. Run **TRCCustom.exe** always at the same computer as where the CCU is installed. Be sure to enter the correct IP address and port number. The *IP* address and port number can be changed in every SDK program in *File -> Options.*

Now compile the example project supplied for your programming environment (TRCsampleD.exe, TRCsampleVB.exe, TRCsampleVC.exe or TRCsampleCB.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 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 SDK. If the device does not indicate the correct value, go through the install steps again to see if you missed something.

15. 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 CCU 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 1927

Code example

```
int error = 0;
error = TRC_Startup("127.0.0.1", 1927);
```

Code example 2

```
int error = 0;
char *IPaddress;
unsigned short portnumber = 1927;
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 altimeter, you can send a value between 0 and 100000 (feet), but for the heading indicator, any value greater then 360 is incorrect and will give a wrong reading. A table with valid values for each device is included at chapter 9.
It is not possible to destroy a servo or SimKits device by sending an invalid value to it.

Code example
```
int error = 0;
char *instrument_name;
float instrument_value = 15;
instrument_name = "heading_compass";
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("adf_compass", instrument_value1);
TRC_SendData("alti_height", instrument_value2);
TRC_SendData("attitude_bank", 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 = "heading_compass";
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("adf_compass", &instrument_value1);
TRC_RecvData("alti_height", &instrument_value2);
TRC_RecvData("attitude_bank", &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 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 CCU 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 1927

Code example

```
int error = 0;
error = TRC_Startup("127.0.0.1", 1927);
```

Code example

```
int error = 0;
char *IPaddress;
unsigned short portnumber = 1927;
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 altimeter, you can send a value between 0 and 100000 (feet), but for the heading indicator, any value greater then 360 is incorrect and will give a wrong reading. A table with valid values for each device is included at chapter 9.
It is not possible to destroy a servo or SimKits device by sending an invalid value to it.

Code example
```
int error = 0;
char *instrument_name;
float instrument_value = 15;
instrument_name = "heading_compass";
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("adf_compass", instrument_value1);
TRC_SendData("alti_height", instrument_value2);
TRC_SendData("attitude_bank", 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 = "heading_compass";
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("adf_compass", &instrument_value1);
TRC_RecvData("alti_height", &instrument_value2);
TRC_RecvData("attitude_bank", &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();
```

17. 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 CCU 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 1927

Code example

```
Call TRC_Startup(Winsock1, "127.0.0.1", 1927)
```

Code example

```
Dim IPaddress As String
IPaddress = "127.0.0.1"
Dim port As Integer
portnumber = 1927
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 altimeter, you can send a value between 0 and 100000 (feet), but for the heading indicator, any value greater then 360 is incorrect and will give a wrong reading. A table with valid values for each device is included at chapter 9.
It is not possible to destroy a servo or SimKits device by sending an invalid value to it.

Code example

```
Dim error As Long
Dim instrument_name As String
Dim Instrument_value As Single
instrument_value = 15
instrument_name = "heading_compass"
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 = 1400
instrument_value3 = 30
' If you want to update all of the devices with a smooth movement, call
these functions at least 20
' times a second
TRC_SendData("adf_compass", instrument_value1)
TRC_SendData("alti_height", instrument_value2)
TRC_SendData("attitude_bank", 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 = "heading_compass"
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("adf_compass", instrument_value1)
TRC_RecvData("alti_height", instrument_value2)
TRC_RecvData("attitude_bank", 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()
```

18. 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 CCU 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 1927

Code example 1:

```
var error : Integer = 0;
error := TRC_Startup('127.0.0.1', 1927);
```

Code example 2:

```
var error : integer = 0;
var IPaddress : String = '127.0.0.1';
var portnumber : Word = 1927;
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 altimeter, you can send a value between 0 and 100000 (feet), but for the heading indicator, any value greater then 360 is incorrect and will give a wrong reading. A table with valid values for each device is included at chapter 9.
It is not possible to destroy a servo or SimKits device by sending an invalid value to it.

Code example

```
var error : integer = 0;
var instrument_name : String = 'heading_compass';
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('adf_compass', instrument_value1);
TRC_SendData('alti_height', instrument_value2);
TRC_SendData('attitude_bank', 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 = 'heading_compass';
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('adf_compass', instrument_value1);
TRC_RecvData('alti_height', instrument_value2);
TRC_RecvData('attitude_bank', 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();
```

SimKits and its accompanying software is a product of:

Company Address:
TRC Development b.v.
Gatwickbaan 9
3045 AP ROTTERDAM AIRPORT
THE NETHERLANDS

Postal Address:
TRC Development b.v.
P.O.Box 12004
3004 GA ROTTERDAM AIRPORT
THE NETHERLANDS

All Products carry a limited warranty of 12 months after purchase.
Please look for warrant 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 reavct on each email within

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 10 439 02 00, (from Europe): 0031 10 439 02 00
Fax: (from USA): 011 31 10 439 02 10, (from Europe): 0031 10 439 02 10

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