Tip/Ring Signal Simulator

TRsSim - Remote Control

User Guide & Reference

Advent Instruments Inc.

Release 1.2 - April 2010

Printed in Canada

Advent Instruments Inc.
111 - 1515 Broadway Street
Port Coquitlam, BC, V3C6M2
Canada

Internet: techsupport@adventinst.com
sales@adventinst.com

Web Site: http://www.adventinstruments.com

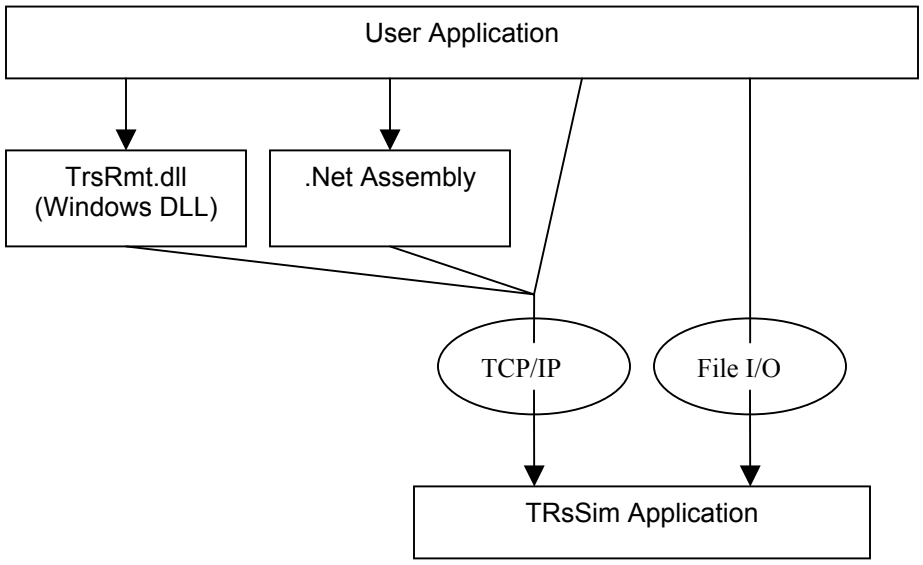
Telephone: (604) 944-4298
Fax: (604) 944-7488

Contents

1. Introduction

This document describes how the TRsSim software can be controlled remotely from another application. Commands from another application may be used to perform various actions or return data from the TRsSim software. This ability becomes useful when integrating the features of TRsSim into a larger testing environment.

The following figure shows the primary methods for sending commands and receiving data.



The TRsSim application supports two different methods for accepting commands. They are via a TCP/IP connection and via simple text files. By default, no commands are accepted by either means unless enabled from the **Preferences** settings window.

An application wishing to control TRsSim can use any of the following approaches:

- **TrsRmt.dll** – This Windows DLL, supplied with the TRsSim software, abstracts the details of establishing a TCP/IP connection. As most Windows programming environments provide mechanisms for DLL access, using the TrsRmt.dll provides a simple means for controlling TRsSim.
- **.Net Assembly** – A .NET 3.5 assembly is available that also abstracts the TCP/IP connection details. This assembly integrates much easier into a .NET project than the TrsRmt.dll and is provided with source code and an example project.

- **Direct TCP/IP Connection** – The user application can directly open a TCP/IP connection with TRsSim. Once open, it can then send commands and receive responses.
- **File I/O Connection** – The user application controls TRsSim directly by writing commands into text files and reading responses from text files.

The TrsRmt DLL, .Net Assembly, examples, and documentation can be downloaded from our web site at:

www.adventinstruments.com

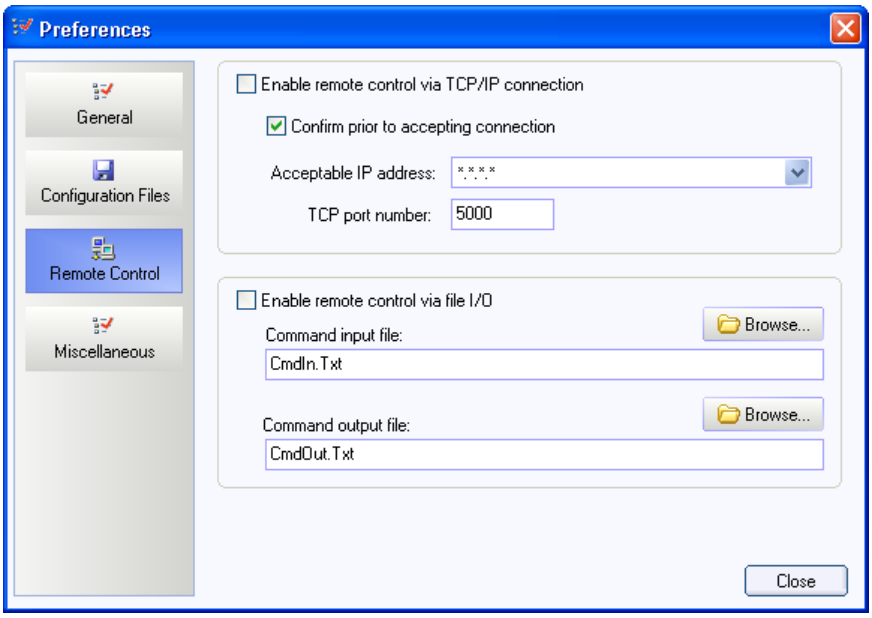
The command reference section of this document focuses on the structure of the commands and responses used by TRsSim. If either the TrsRmt DLL or the .Net Assembly is being used, the applicable documentation should be downloaded from the web site URL shown above.

The next two sections provide an overview on how the TCP/IP and file I/O remote control operates and is configured within the TRsSim software.

2. Tcp/Ip Operation

2.1 Configuring TRsSim

By default the TRsSim software does not accept any TCP/IP connections regardless of the source IP address or port. In order to enable a connection, the default TCP/IP settings must be changed. To view the current settings, select the "**Preferences**" command from the "**Configuration**" menu. This displays a window similar to the following. Once visible, click the "Remote Control" category button on the left side. The following figure shows the default settings.

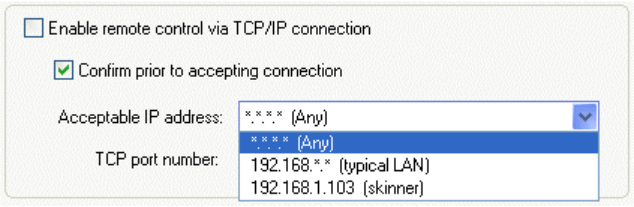


The top most check box enables or disables TCP/IP connections. Unless it is checked, connections from other applications are not allowed. The next setting determines if a confirmation message is displayed before allowing a TCP/IP connection. If checked (default), the user must click a "Yes" button when the confirmation message is displayed. If "No" is clicked, then the TCP/IP connection is refused.

The following two settings are used to control the acceptable IP address of the controlling device, and the TCP port number.

By default, any IP address is allowed as the controlling device. The drop down list box includes two other settings. The second selection represents common LAN IP addresses, while the third selection is the IP address of the PC running the TRsSim software. If

available, the PC name is displayed in brackets following the IP address. In the example figure below, the PC called "skinner" is located at address 192.168.1.103.



Any IP address can be entered into the above text box. The '*' character indicates any value from 0 to 255 is acceptable.

The default TCP Port used is 5000. However this value can be changed if a conflicts occur with other applications. The maximum allowable port value is 65535. Normally port values should be above 1024 to avoid conflicts with common applications.

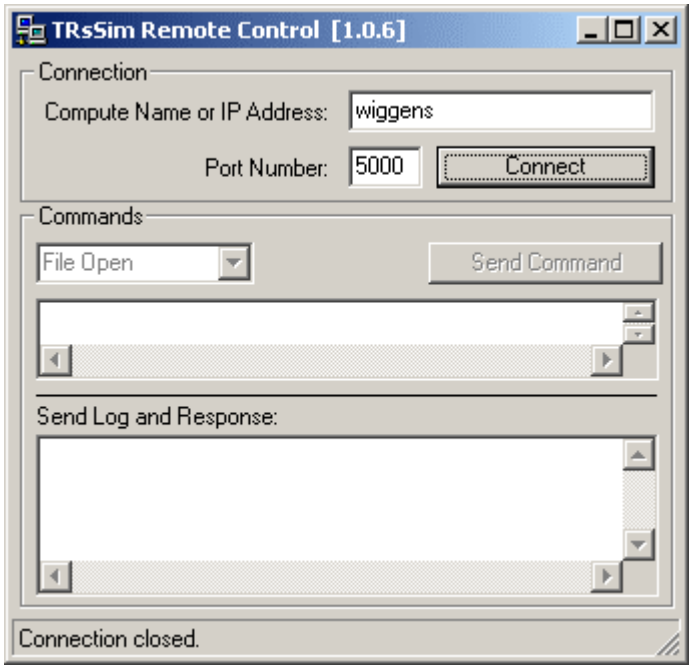
All of the above settings are saved when the TRsSim program shuts down, and reloaded when started again. Loading configuration files or selecting the "Restore Default Settings" menu command has no effect on the TCP/IP settings.

If TCP/IP connections are enabled then the lower right corner of the TRsSim software displays a small icon. The icon indicates that it is listening on the specified TCP port for a controlling application to initiate a connection. It is important to note that only a single connection is accepted by the TRsSim software. Once a connection is established, requests by other devices for a connection will be refused.

2.2 Establishing a Connection

For demonstration purposes, a special application is included with the TRsSim software package. This program, "TRsSimRC.exe" is located in the same application directory as the main TRsSim software. It can be used to setup a TCP/IP connection and send commands to the TRsSim software.

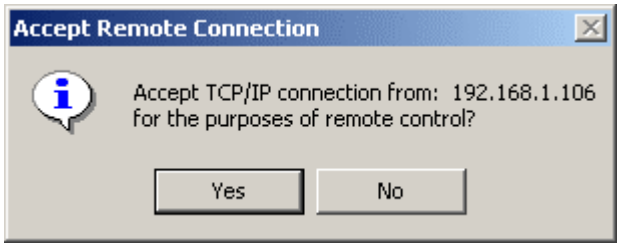
Upon starting the program, a single window should appear, as shown in the following figure. To setup a connection to the TRsSim software, the IP address or PC name must be entered in the appropriate text box. By default, the name of the PC is automatically entered. In addition, the port number setting must match that specified in the TRsSim software.



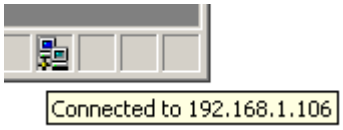
The following conditions must be met, before a connection can be established.

- TRsSim software is running.
- TRsSim software settings have enabled TCP/IP connections.
- No connection is currently established with another program.
- The acceptable IP address setting matches the IP address of the demonstration program.
- The port values match in both the TRsSim software and the demonstration program.

Clicking the "Connect" button on the demonstration program results in a connection attempt. If the TRsSim software is configured to require a confirmation of acceptance, then the following message is displayed.

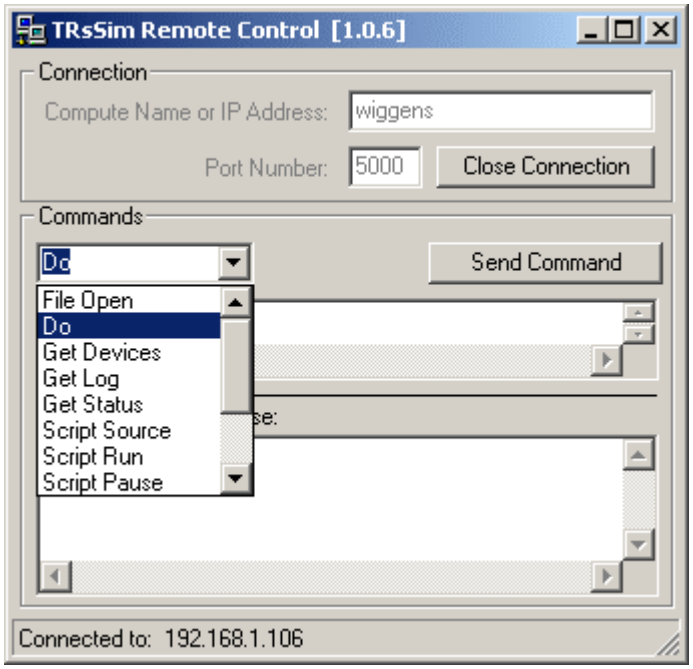


Choosing the "Yes" button allows the connection, while the "No" button prevents a connection. Assuming the "Yes" button is clicked, a TCP/IP connection is then established between the demonstration and TRsSim program. The icon displayed in the lower right corner of the TRsSim software changes to reflect the connection. If the mouse if moved over the icon, the IP address of the remote computer is displayed.



2.3 Sending Commands

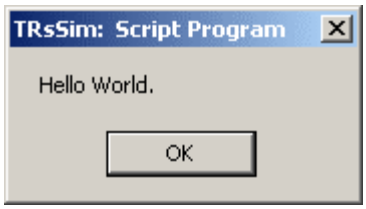
Once a TCP/IP connection is established, commands can be sent to the TRsSim software. The demonstration program included with the TRsSim software provides a simple way to send various commands. The figure below shows the demonstration program after a connection has been established.



Commands are selected from the drop down list box. Some commands may include additional data, which is entered in the text box below the drop down list. In this example, the command DO was selected. The DO command tells the TRsSim software to execute script language statements. In the text box below the drop down list, the following script language statement was entered.

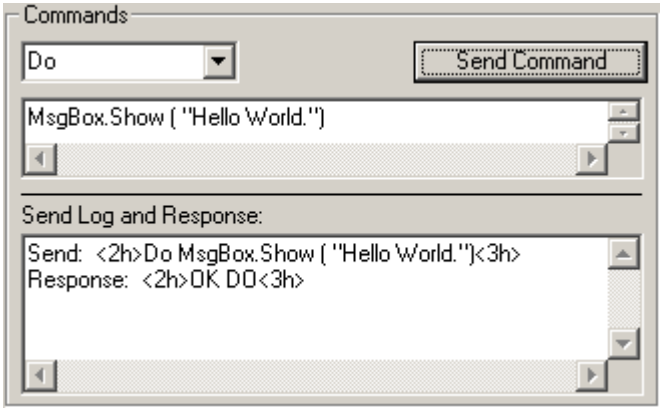
```
MsgBox.Show ( "Hello World")
```

This script statement tells the TRsSim program to display the message "Hello World" to the user. Once the "Send Command" button is clicked on the demonstration program window, the command is sent and the TRsSim software displays the message.



Click the "OK" button to remove the message window from the TRsSim software.

The lower text box in the demonstration programs displays the commands sent to the TRsSim software and the responses returned by the TRsSim software. The figure below shows an example of the command sent.



It is important to note that the ASCII character STX (02h) is used to mark the start of all messages, while ETX (03h) marks the end of all messages. The TRsSim software will ignore any data bytes sent that is not delimited by the STX and ETX characters.

The TRsSim software will always send a response to every command. If the command was free of syntax errors, the first word is always "OK". In case syntax or other errors are detected, the first word returned is "ERROR". In the example above, the message "OK DO" was returned. This indicates that the DO command was executed successfully.

2.4 Message Log

The TRsSim software maintains a log of all the TCP/IP messages received from and sent to the controlling application. This log can become useful in troubleshooting problems encountered when attempting to control the TRsSim software.

The log contents are stored in the file "LogTcpIp.txt" located in the following directory:

C:\Documents and Settings\All Users\Application Data\Advent\TRsSim\data\<dircode>

Where <dircode> is a five digit number based on where the TRsSim software was installed on the PC.

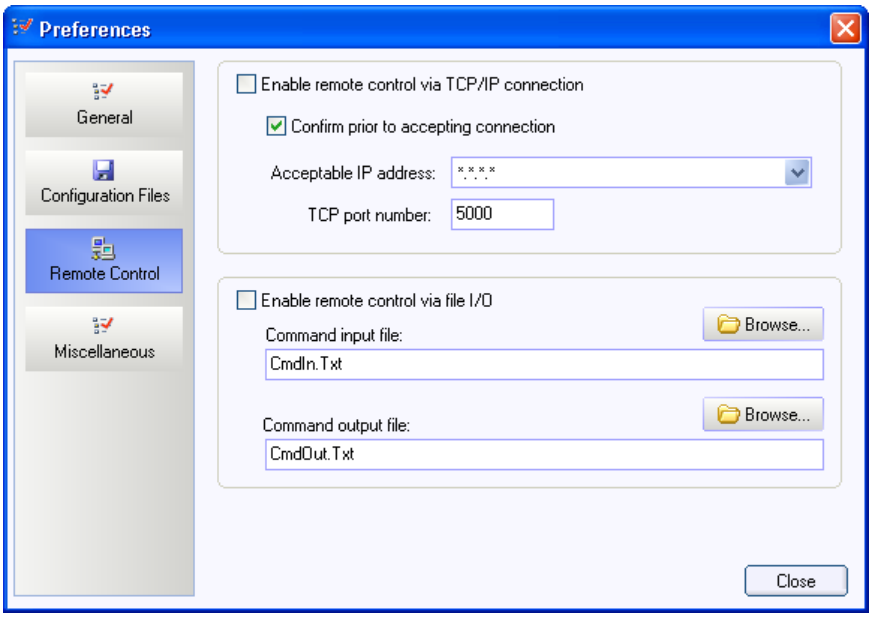
When the TRsSim programs starts, it erases any existing log file and starts a new one.

The following log is the result of the example shown in the previous two topics. The first three lines are written upon program startup and when TCP/IP connections are enabled, while the last two lines are written when the TRsSim program is shutdown.

```
TCP/IP Remote Access Log
***Start Up  [Oct-26-2004,  17:17:10]
***Listen on port:  5000  [Oct-26-2004,  17:17:10]
***Request connection from:  192.168.1.106  [Oct-26-2004,  17:17:22]
***Request accepted from:  192.168.1.106  [Oct-26-2004,  17:17:23]
***Rx Data:  34 bytes.
***Rx Message:  Do MsgBox.Show ( "Hello World.")  [Oct-26-2004,  17:17:28]
***Response:  OK DO  [Oct-26-2004,  17:17:28]
***Close connection  [Oct-26-2004,  17:17:33]
***Listen on port:  5000  [Oct-26-2004,  17:17:33]
***Close connection  [Oct-26-2004,  17:19:26]
***Shut Down  [Oct-26-2004,  17:19:26]
```

3. File I/O Operation

As an alternative to a TCP/IP connection, the TRsSim software can accept commands from a text file and respond with data by writing to a text file. By default, this method of remote control is disabled. In order to enable file I/O, select the "**Preferences**" command from the "**Configuration**" menu. This displays a window similar to the following. Once visible, click the "Remote Control" category button on the left side. The following figure shows the default settings.



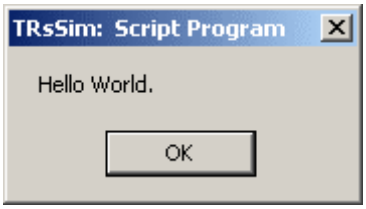
If file I/O remote control is enabled, the TRsSim software continuously monitors the specified command input file for either its existence or a change in its date/time. Once the file is newly created, or its date/time changes, the file is read by TRsSim and the command contained within is executed.

As each command returns a response, an output file is created with the response of the command. By default the names of the input and output files are "CmdIn.Txt" and "CmdOut.Txt" respectively. Alternate file names may be used by entering them in the text field provided. If the files are specified without a complete path, they are assumed to be located in the following directory:

C:\Documents and Settings\All Users\Application Data\Advent\TRsSim

As an example, creating the "CmdIn.txt" file with the following single command, and then enabling remote control via file I/O, causes the TRsSim software to execute the script command and display the following window.

```
Do MsgBox.Show ( "Hello World")
```



Click the "OK" button to remove the message window from the TRsSim software.

The resulting "CmdOut.Txt" file created by TRsSim contains a successful acknowledgement of the command:

```
OK DO
```

4. Command Reference

Every command sent to the TRsSim software must begin with one of seven keywords. They are: DO, EXIT, FILE, GET, RESET, SCRIPT, and DEV. Any other command causes the return of an ERROR message. Note that the commands are not case sensitive.

The following sections describe the syntax and operation of each keyword.

4.1 DO Command

The DO command is used to immediately execute script statements. Any valid script statement can follow the DO command. The syntax is:

DO <script language statements>

Reponses:

OK DO (no errors)

ERROR <message> (error detected in script statements)

When the TRsSim software receives this command, it checks the script statements for syntax errors. If no errors are detected, it then immediately executes the script statements. However if errors are detected, it then returns an error response.

The specified script statements execute independently from any other script program. They can be used to perform various tasks at any time. One script statement is sent per line, so multiple script statements require more than one line. If multiple script statements are sent in the DO command, they must include the declaration for a subroutine called Main(). See the examples below.

The amount of time required to execute the script statements is dependent on the script statements themselves. For example, if sending a Caller ID transmission, normally the statement does not finish until the transmission is complete. The "DO OK" response is sent before the script statement starts to execute. In order to determine if the script statements have finished execution, use the GET STATUS command (see GET Command). Sending another "DO" command while any script statements are still running causes the their execution to be stopped, and the new statements to execute instead.

Examples:

1. Send multiple script statements that configure the TRsSim Caller ID type, signaling type, and message to send. Note that for multiple script statements, the first statement is "Sub Main" and the last is "End Sub". The Main() subroutine is the entry point for any

script language execution. It is possible to declare multiple subroutines or functions and call these routine from the Main() subroutine.

```
DO  Sub MAIN
    Let Cid.Type = ctFskEtsi
    Let Cid.SignalingOnHook = "RP-AS Before Data (Ring)"
    Let Cid.SignalingOffHook = "DT-AS Wait For ACK"
    Let Cid.Message = "Calling Name and Number"
End Sub
```

2. Once the above example has selected the Caller ID type, signaling, and message, this single statement command starts the Caller ID transmission. Note that when sending a single script statement, the "Sub Main" and "End Sub" statements are implied and do not need to be included.

```
DO  Call Action.SendCID
```

4.2 EXIT Command

The EXIT command is used to exit the TRsSim software. The syntax is:

EXIT

Response:

OK EXIT

After the TRsSim software sends the response, it starts the process of shutting itself down. Please note that if the software is currently displaying a "modal" window, then the shutdown process is delayed until the window is closed. Modal windows stop all background tasks while they are displayed. These are used in various locations within the TRsSim software. Usually for either displaying various program settings, or reporting error messages.

Once the TRsSim software completes it shutdown process, the TCP/IP connection is lost.

4.3 FILE Command

The FILE command allows configuration files or script project files to be loaded into the TRsSim software. The syntax is:

FILE OPEN <file path & name>

Reponses:

OK FILE (no errors)

ERROR <message> (error detected during file load)

WARNNG <message> (warning generated during file load)

One of three different responses is returned by this command. If no errors are detected, then "OK FILE" is returned by the TRsSim software. In the case file errors are encountered, an error message is returned. A warning message may be returned if the file

opened contained settings that are not applicable to the current configuration of the TRsSim software.

4.4 GET Command

The GET command is used to return general information from the TRsSim software. Three variants of this command may be used. They are:

GET DEVICES

GET LOG [<name>]

GET STATUS

GET DEVICES:

This command returns information on what devices are currently connected to the TRsSim software. The format of the response is as follows:

```
OK  GET  COUNT=<n>
DEV1=<status>:<identifier>:<serial number>:<name>:<soft vrs>:<hard vrs>
DEV2=<status>:<identifier>:<serial number>:<name>:<soft vrs>:<hard vrs>
...
DEVn=<status>:<identifier>:<serial number>:<name>:<soft vrs>:<hard vrs>
```

Following the OK keyword, the number of devices is returned in the form of "COUNT=<n>". Details of each device are included in a separate line. Each line contains multiple fields separated by the colon ':' character. Each field is defined as follows:

<status>	Current connection status: Connected, NotPresent, NoKey, Demo
<identifier>	Single letter representing the device identifier
<serial number>	Device serial number
<name>	Device name
<soft vrs>	Device software version
<hard vrs>	Device hardware version

An key field is the <identifier>, since it returns what letter is used to represent each device. When using script statements with multiple connected devices, the identifier determines which device the statement is directed to. For example, if an AI-7280 is connected as device 'A' and an AI-80 as device 'B', then the following DO command starts a Caller ID transmission from both devices.

```
DO  Sub MAIN
    Call DevA.Action.SendCID
    Call DevB.Action.SendCID
End Sub
```

GET LOG [<name>]:

Returns the contents of one or all of the script output logs. If no name is specified, then all script output logs are returned by this command. The response format to this command is as follows:

```
OK  GET  COUNT=<n>
LOG1=<name of script output log #1>
> (line 1 of log)
> (line 2 of log)
...
> (last line of log)
LOG2=<name of script output log #2>
> (line 1 of log)
> (line 2 of log)
...
> (last line of log)
LOGn=<name of script output log #n>
> (line 1 of log)\
> (line 2 of log)
...
> (last line of log)
```

Following the OK keyword, the number of script logs is returned in the form of "COUNT=<n>". If the value <n> is one or greater, then the following line outputs the name of the first script log. Following the name, each line in the log is returned. However the character '>' is prefixed to each line. After the last line of the log contents, any additional script logs are returned.

The use of the script output logs and this command is meant to be the primary means for a remote application to acquire data from the TRsSim software. By using either the DO or SCRIPT commands, output logs can be created containing virtually any type of output data. The GET LOG command is then used to return the data to the remote application.

Example:

Use the DO statement to copy the hook detect state, DC line voltage, and DC loop current of an AI-7280 to a script output log. Then return this information with the GET LOG command.

```
DO  Sub MAIN
    Log.Add("RemoteData")
    Log.Clear
    Log.AddLine = "OffHook=" + Str(TelInt.HookDetect)
    Log.AddLine = "Voltage=" + Str(Meter.DcVoltage)
    Log.AddLine = "Current=" + Str(Meter.DcCurrent)
End Sub
```

GET LOG RemoteData

The data returned in this example is as follows:

```
OK COUNT=1
LOG1=RemoteData
>OffHook=1
>Voltage=-7.81162023544312
>Current=26.2322692871094
```

GET STATUS:

This command returns the status of script statement execution. Both information on any script program running and DO script statements is returned in the following format:

OK GET SCRIPT=<script state> DO=<do state>

Where:

<script state> State of any script program executing:
- STOP no script execution
- RUN script running
- PAUSE script in pause mode
- IDLE script waiting for events
- ERROR error detected in script program

<do state> State of any DO script statements
- STOP no execution
- RUN script statements running
- ERROR error detected in script statements

4.5 RESET Command

The RESET command is used to return the TRsSim program settings back to their default state. Sending this command has the same effect as the "Restore Default Settings" menu command. The syntax is:

RESET

Response:

OK RESET

4.6 SCRIPT Command

The SCRIPT command is used to control script programs. This includes setting the source code for a script project along with controlling its execution. Five different variants of this command may be used. They are:

SCRIPT END
SCRIPT PAUSE
SCRIPT RESUME
SCRIPT RUN
SCRIPT SOURCE <script program>

SCRIPT END:

This command stops execution of any script program regardless of what state it is current in (running, paused, idle). The response to this command is as follows:

OK SCRIPT

SCRIPT PAUSE:

This command halts any script programs from running. If the script program is not running, then this command has no effect. Once halted, use the RESUME command to return to the running state. The response to this command is as follows:

OK SCRIPT

SCRIPT RESUME:

This command resumes script program execution if the program was halted by using the PAUSE command. The response to this command is as follows:

OK SCRIPT

SCRIPT RUN:

This commands starts execution of the current script program. If the user has made changes to the script program, then it is first compiled. Any errors detected prevent the program from running. In addition an ERROR response is returned. If no script errors are detected, then a OK response is returned. The format of the possible responses are as follows:

OK SCRIPT	(no errors)
ERROR <message>	(compiler detected error in script)

SCRIPT SOURCE <script program>:

This commands performs the following tasks:

a) Stop any executing script program

b) Erase the existing script program

c) Create a new script program using the statements passed in the command

d) Compile the new script program to check for errors.

If a compiler error is detected, an ERROR response is returned. Otherwise an OK response is returned. The format of the two possible responses are as follows:

OK SCRIPT	(no errors)
ERROR <message>	(compiler detected error in script)

Example:

Use the SCRIPT commands to continually send Caller ID when connected to either an AI-7280 or AI-80. First set and compile the script program with the following command:

```
SCRIPT  SOURCE  Sub Main
```

```
    Loop
        Action.SendCID
    End Loop
End Sub
```

Next start the script program by:

SCRIPT RUN

Pause, resume or stop the script program with the following commands:

SCRIPT PAUSE

SCRIPT RESUME

SCRIPT END

4.7 DEV Command

The DEV command is used to issue a device specific command. The TRsSim software currently supports three different classes of devices. They are:

- AI-80 or AI-7280 for emulating a PSTN central office line, or
- AI-5120 for monitoring the signals on a telephone line
- AI-5620 for simulating a terminal equipment device

The syntax of the DEV command is as follows:

DEV <identifier> <command data>

Where the <identifier> field represents the device the command is being sent to. The TRsSim software represents each device it is connected to by a letter ranging from 'A' to 'D'. The GET DEVICES command can be used to obtain a listing of all the devices attached to the TRsSim software.

The response to the DEV command can take one of two different forms. They are:

OK DEV <returned data> (no errors)

ERROR <message> (syntax error or invalid data passed)

The 'OK' response indicates that the command was successfully executed. Depending on the command, data may be returned following the 'OK' characters. If the command syntax or data passed was invalid, the 'ERROR' response is returned.

Currently only the AI-5120 and AI-5620 use the DEV commands. For more information on the capabilities and syntax of the DEV commands see the sections: Device Commands: AI-5120 (page 20) and Device Commands: AI-5620 (page 25).

5. Device Commands: AI-5120

The AI-5120 currently supports three device related commands. They are used in conjunction with the DEV command to return information about detected events and waveform data.

5.1 Get Event Command (AI-5120)

The "Get Event" command returns information about events captured by the AI-5120. The syntax for this command is as follows:

DEV <identifier> GET EVENT <selector>

Where:

<identifier> Letter representing the AI-5120 device
(normally 'A')
(use GET DEVICES to determine the identifier)

<selector> Selects which event to return information on:
FIRST first event (earliest in time)
LAST last event (latest in time)
NEXT event following current event
PREV event previous to current event
CURRENT currently selected event

The first time this command is called, the <selector> should be either FIRST or LAST. This establishes an index into the list of events maintained by the AI-5120. Subsequent calls can then use the NEXT, PREV, or CURRENT keywords to scan the event list.

Note that only the events marked for data export and logging are accessible by the GET EVENT command. This allows for filtering out unwanted events by disabling them in the export and data log settings window.

The response to this command can take one of two different forms. They are:

OK DEV EVENT=<index> [<CRLF> <info>] (no errors detected)

ERROR <message> (syntax error or invalid data passed)

If no error is detected, the response is the keyword 'OK' followed by 'EVENT=' and a single integer. The integer number represents an index into the data lists stored in the TRsSim software. If the number is greater than zero, then the response is followed by at least three text lines containing information on the selected event. Each text line is

separated by a carriage return (0Dh) and line feed (0Ah) character. If no events are detected or none pass the current filter settings, the value zero is returned.

The format of the event information is as follows:

TYPE = <event type code> : <event type name>
TIME = <yr> : <mn> : <day> : <hr> : <min> : <sec> : <time from start>
WAVE= <AC coupled waveform ref #> : <DC coupled waveform ref #>
> [event details]

The first three text lines provide the type of event, time of the event, and reference numbers to any captured waveforms. For all events, the format of the first three text lines is the same. Depending on the event type, additional text lines may be returned as well. Some events have no additional information, while others return measurement information. Each line of additional data is preceded by the '>' character.

For example, the following command requests information on the first event that passes the export and log settings filter.

Send:

```
Dev A Get Event First
```

Response:

```
OK DEV EVENT=1
TYPE=9:DTMF Digits
TIME=2005:2:1:14:0:12:17.551
WAVE=2:3
>NUMBERDIGITS=3
>DIGIT1=2:696.9:1335.8:-8.6:-5.8:dBm:83:-1:ms
>DIGIT2=3:697.0:1476.7:-8.6:-6.0:dBm:83:2460:ms
>DIGIT3=6:770.1:1477.3:-8.6:-6.0:dBm:83:374:ms
```

The DTMF digits occurred at a date of Feb. 1st, 2005, at 2:00 pm and 12 seconds. The time delay from the start of event recording was 17.551 seconds. An AC and DC coupled waveform was recorded with reference numbers of 2 and 3 respectively. The reference number can be used to retrieve the waveform samples. The last three text lines show the measurements taken for each digit. For each digit, the digit code, low tone frequency, high tone frequency, low tone level, high tone level, level units, tone duration, inter-digit interval are listed. The last field contains the units used for the duration and inter-digit times. As the first digit has no inter-digit time, the value -1 is returned.

5.2 Get WaveInfo Command (AI-5120)

The "Get WaveInfo" command returns information regarding a specific waveform recorded by the AI-5120. The syntax for this command is as follows:

DEV <identifier> GET WAVEINFO <ref#>

Where:

<identifier>	Letter representing the AI-5120 device (normally 'A') (use GET DEVICES to determine the identifier)

<ref#> An integer representing the desired waveform
Use "Get Event" to return this value

The "Get Event" command returns two waveform reference numbers for each event. The first number is for AC coupled waveforms, while the second is for DC coupled waveforms. By using these reference numbers with the "Get WaveInfo" command, the TRsSim software returns information regarding the specific waveform. This includes the number of samples, sample rate, start time, and voltage scaling factor.

The response to this command can take one of two different forms. They are:

OK DEV WAVEINFO=<ref#> <CRLF> <info> (no errors detected)

ERROR <message> (syntax error or invalid data passed)

If no error is detected, the response is the keyword 'OK' followed by 'WAVEINFO=' and the passed waveform reference number. An additional two text lines return the waveform information. If an invalid reference number is sent, then the error response is returned.

The format for the waveform information is as follows:

INFO = <start time> : <# samples> : <sample rate> : <scale factor> ...

: <maximum block size> : <bits per sample>

BLOCKS = <number of data blocks required to the read the waveform>

The text line starting with 'INFO=', returns the start time of the waveform, number of samples recorded, sample rate, and scale factor. As each sample is represented by a 16 bit signed value, the scale factor is used to convert this signed integer value to volts.

The second line starting with "BLOCKS=" returns the number of data blocks needed to transfer all the samples contained within the waveform. Each block represents 2048 waveform samples. To read the waveform blocks, use the 'GET WAVEBLOCK' command.

For example, the following command requests information on a waveform with a reference number of 2.

Send:

```
Dev A Get WaveInfo 2
```

Response:

```
OK DEV WAVEINFO=2
INFO=17.381:15063:19531.250:0.00017261:2048:16
BLOCKS=8
```

This waveform starts at a time of 17.381 seconds and contains 15063 samples. The sample rate used to store the data is 19531.25 Hz and the scale factor needed to convert the sample values back to volts is 0.00017261. At most 2048 samples are returned for each GET WAVEBLOCK command, and each sample is composed of 16 bits.

To read all of the waveform samples, 8 calls with 'GET WAVEBLOCK' are needed. Each command will return a single block of 2048 samples, with the possible exception of the last sample block.

5.3 Get WaveBlock Command (AI-5120)

The "Get WaveBlock" command returns a single block of samples from a specific waveform recorded by the AI-5120. The syntax for this command is as follows:

DEV <identifier> GET WAVEBLOCK <ref#> <block#>

Where:

<identifier>	Letter representing the AI-5120 device (normally 'A') (use GET DEVICES to determine the identifier)
<ref#>	An integer representing the desired waveform Use "Get Event" to return this value
<block#>	Specifies which block of samples to return. (first block is 1)

The response to this command can take one of two different forms. They are:

OK DEV WAVEBLOCK=<ref#> : <block#> <CRLF> <info>

(no errors detected)

ERROR <message> (syntax error or invalid data passed)

If no error is detected, the response is the keyword 'OK' followed by 'WAVEBLOCK' and the passed waveform reference and block number. Additional text lines return the waveform samples. If an invalid reference number is sent, then the error response is returned.

The format for the waveform sample information is as follows:

BLOCK = <#sample returned> : <#bits per sample> : <checksum>
> (first line of 32 samples in hexadecimal format)
> (second line of 32 samples in hexadecimal format)
...
> (last line of up to 32 samples in hexadecimal format>

The first additional line starts with 'BLOCK=', and returns the total number of samples sent, number of bits per sample, and a 16 bit checksum value representing the sum all of the sample values. Each subsequent line starts with the '>' character and contains up to 32 samples in a hexadecimal format. Each hexadecimal value represents a signed integer value.

Currently all waveforms returned use 16 bits per sample.

For example, the following command requests the first sample block from waveform #2.

Send:

```
Dev A Get WaveBlock 2 1
```

Response:

```
OK DEV WAVEBLOCK=2:1
BLOCK=2048:16:F148
>0002FFFF0001000100020003...
>FFFF00010004FFFEFFFE0001...
...
>0006FFFE0001FFFDFFEFFFF3...
```

The block of samples returned contains 2048 samples at 16 bits per sample. The checksum value for all the samples returned is F148h. 64 lines of hexadecimal samples follow. Each line contains 32 sample values represented by a 4 digit hexadecimal value.

6. Device Commands: AI-5620

The AI-5620 currently supports three device related commands. They are used in conjunction with the DEV command to return information about detected events and waveform data.

6.1 Get Event Command (AI-5620)

The "Get Event" command returns information about events captured by the AI-5620. The syntax for this command is as follows:

DEV <identifier> GET EVENT <selector>

Where:

<identifier>	Letter representing the AI-5620 device (normally 'A') (use GET DEVICES to determine the identifier)
<selector>	Selects which event to return information on: FIRST first event (earliest in time) LAST last event (latest in time) NEXT event following current event PREV event previous to current event CURRENT currently selected event

The first time this command is called, the <selector> should be either FIRST or LAST. This establishes an index into the list of events maintained by the AI-5620. Subsequent calls can then use the NEXT, PREV, or CURRENT keywords to scan the event collection.

The scripting event filter determines which events are retrieved by this command. This provides a mechanism for rejecting unwanted event types and only accessing desired event types. For example, to configure the filter for only ringing events use the following scripting statements:

```
Const cRingingEventType = 11
With Event.Filter
    Call .ShowNone
    Call .TypeAdd(cRingingEventType, cWarn_Any)
    Call .SubmitQuery
End With
```

The above scripting statements can be passed to TRsSim for execution by using the 'DO' command.

The response to the GET EVENT command can take one of two different forms. They are:

OK DEV EVENT=<position>:<count> [<CRLF> <info>]

ERROR <message> (syntax error or invalid data passed)

If no error is detected, the response will be one or more text lines containing event information. Each text line returned is separated by a carriage return (0Dh) and line feed (0Ah) character. The first text line always starts with 'OK' and the command keyword. Additionally it provides the position of the event in the filter's collection along with the number of events present in the collection. The first event always has a position value of 1 while the last event's position value will be equal to the returned count value. If no events pass through the script filter, then both the position and count values are returned as zero.

Additional text lines provide details on the selected event. The format of the event information is as follows:

TYPE = <event type code> : <event type name>
TIME = <yr> : <mn> : <day> : <hr> : <min> : <sec> : <time from start>
> [event details]

The first two lines provide the type of event and timing information on the event. For all events, the format of the first two text lines is always the same. However depending on the event type, additional text lines may be returned. Some events have no additional information, while others return measurement information. Each line of additional data is prefixed by the '>' character.

For example, the following command requests information on the first event that passes through the script filter. The first event in this case represents the detection of ringing.

Send:

```
Dev A Get Event First
```

Response:

```
OK DEV EVENT=1:4
TYPE=11:Ringing
TIME=2010:4:20:16:44:42:24.1138
>DURATION=1.977:s
>FREQ=22.0:Hz
>LEVEL=80.2:Vrms
>DCOFFSET=-47.9:V
>CRESTFACTOR=1.42
>VOLTPP=227:V
>VOLTMIN=-162:V
>VOLTMAX=66:V
```

The ringing signal occurred at a date of April. 20th, 2010, at 4:44 pm and 42 seconds. The time delay from the start of event recording was 24.1138 seconds. The text lines prefixed with the '>' character provide measurement information on the ringing signal. While each event type has different measurements, the format is generally similar to above. Each measurement consists of a name followed by an equal '=' character, which is

then followed by its measurement value. Additionally, if applicable, units are provided after the measurement value.

6.2 Get WaveInfo Command (AI-5620)

The "Get WaveInfo" command is used to obtain waveform information available within a specified time range. Three fields are used to pass along the type of waveform and the start and end times representing the desired range. The syntax for this command is as follows:

DEV <identifier> GET WAVEINFO <type> <start time> <end time>

Where:

<identifier> Letter representing the AI-5620 device
(normally 'A')
(use GET DEVICES to determine the identifier)

<type> Type of waveform. Valid types are:

AcV	Ac coupled line voltage
LineV	Dc coupled line voltage
LoopI	Dc coupled loop current
CmV	Dc coupled common mode voltage

<start time> Start time in seconds
<end time> Ending time in seconds

The response to this command can take one of two different forms. They are:

OK DEV WAVEINFO=<ref#> <CRLF> <info> (no errors detected)

ERROR <message> (syntax error or invalid data passed)

If no error is detected, the response is the keyword 'OK' followed by 'WAVEINFO=' and a positive integer representing a waveform reference number. If no waveform of the specified type is present in the specified time range the reference number is always zero. However if a waveform does exist, the reference number will be a random positive integer value. This same reference number is needed if the waveform is to be downloaded with the 'WaveBlock' command.

An additional two text lines return the waveform information.

The format for the waveform information is as follows:

INFO = <start time> : <# samples> : <sample rate> : <scale factor> ...

: <maximum block size> : <bits per sample>

BLOCKS = <number of data blocks required to the read the waveform>

The text line starting with 'INFO=', returns the time of the first sample of the waveform selected, the number of samples that can be returned, sample rate, and scaling factor. As each sample is represented by a 16 bit signed value, the scale factor is used to convert this signed integer value to volts or mA.

The second line starting with "BLOCKS=" returns the number of data blocks needed to transfer all the samples contained within the waveform. The maximum number of samples in a block is returned in the <maximum block size> file of the "INFO=" line.

To read the waveform blocks, use the 'GET WAVEBLOCK' command.

For example, the following command requests information on the line voltage waveform captured during the time range of 10 to 20 seconds.

Send:

```
Dev A Get WaveInfo LineV 10.0 20.0
```

Response:

```
OK DEV WAVEINFO=533425
INFO=11.675000:8325:1000.00:0.010071103:2048:16
BLOCKS=5
```

The waveform selected is given a reference number of 533425 and begins at a time of 11.675 seconds and contains 8325 samples. The sample rate used to store the data is 1000 Hz and the scale factor needed to convert the sample values back to volts is 0.010071103. At most 2048 samples are returned for each GET WAVEBLOCK command, and each sample is returned as a 16 bit signed integer.

To read all of the waveform selected, 5 calls with 'GET WAVEBLOCK' are needed. Each command will return a single block of up to 2048 samples.

6.3 Get WaveBlock Command (AI-5620)

The "Get WaveBlock" command returns a single block of samples from a selected waveform recorded by the AI-5620. The waveform range must be first selected with the "Get WaveInfo" command prior to sending this command. The syntax for this command is as follows:

DEV <identifier> GET WAVEBLOCK <ref#> <block#>

Where:

<identifier>	Letter representing the AI-5620 device (normally 'A') (use GET DEVICES to determine the identifier)
<ref#>	An integer representing the selected waveform which was returned by the "WaveInfo" command.
<block#>	Specifies which block of samples to return. (first block is 1)

The response to this command can take one of two different forms. They are:

OK DEV WAVEBLOCK=<ref#> : <block#> <CRLF> <info>

ERROR <message> (syntax error or invalid data passed)

If no error is detected, the response is the keyword 'OK' followed by 'WAVEBLOCK' and the passed waveform reference and block number. Additional text lines return the waveform samples. If an invalid reference number or block number is sent, then an error message is returned.

The format for the waveform sample information is as follows:

BLOCK = <#sample returned> : <#bits per sample> : <checksum>
: <first sample index>
> (first line of 32 samples in hexadecimal format)
> (second line of 32 samples in hexadecimal format)
...
> (last line of up to 32 samples in hexadecimal format>

The first additional line starts with 'BLOCK=', and returns the total number of samples to follow, number of bits per sample, a 16 bit checksum value representing the sum all of the sample values, and finally the index representing the first sample returned. Each subsequent line starts with the '>' character and contains up to 32 samples in a hexadecimal format. Each hexadecimal value represents a signed integer value.

It is important to note that the waveform time range selected may not be continuous. It is possible that portions of the time range may not contain waveform samples because they have been deleted or none where recorded during that time interval. Each block of samples returned will always represent continuous waveform samples; however, the first sample of a block may not follow the last sample of the previous block. The field '<first sample index>' can be used to determine if this block is continuous with the prior block. It represents the index to the first sample in this block with respect to the entire waveform time range selected.

For example, if the waveform is continuous over the time range selected, the first sample of the first block has an index of zero. The first sample of the next block has an index equal to the maximum block size (normally 2048). The first sample of the third block has an index equal to two times the block size (2 x 2048), and so on. However if a gap exists between the second and third blocks, the first sample index of the third block will not be equal to the first sample index of the second block plus the number of sample in the second block. Rather it will be a value greater than this. The duration of the gap in the waveform can be calculated from this difference in index values and the waveform's sample rate.

Currently all waveforms returned use 16 bits per sample.

For example, the following command requests the first sample block from the waveform selected by the "WaveInfo" command example.

Send:

```
Dev A Get WaveBlock 533425 1
```

Response:

```
OK DEV WAVEBLOCK=533425:1
BLOCK=2048:16:38CF:0
>ED72ED61ED61ED61ED61ED61...
...
>0006FFFE0001FFFDFFEFFFF3...
```

The block of samples returned contains 2048 samples at 16 bits per sample. The checksum value for all the samples returned is 38CFh. 64 lines of hexadecimal samples follow. Each line contains 32 sample values represented by a 4 digit hexadecimal value. The last field value of zero indicates that the first sample of this block is the first sample of the selected waveform (sample index equal to zero). This is always the case when reading the first block of samples.