

Controlling RADIAN RX Reference Standards with SCPI

Application Note AN-RX-SCPI-01 · Revision 2.0 · October 2026

Applies to: All RADIAN RX reference standards (RX-10/11/15, RX-20/21/23, and RX-30/31/33). The examples use a three-phase RX-30, RX-31, or RX-33; on a single-phase RX, use input 1 only.

Replaces: “SCPI Console Overview for Use with the Radian RX Xytronic Standard” (10/31/18) and “Changing the Pulse Ports on a RX Standard Using SCPI”

Overview

Radian Research designed the RADIAN RX family of reference standards around SCPI (Standard Commands for Programmable Instruments), the standard command language for test and measurement instruments. SCPI builds on the IEEE 488.2 standard, so the commands follow familiar rules and work from almost any software that can open a network connection.

Over its Ethernet port, an RX standard accepts SCPI commands that let you:

- read live measurements (volts, amps, watts, VARs, VA, power factor, phase angle, and frequency) on each input,
- accumulate energy (Wh, VARh, VAh, and more) over a set time or pulse count,
- set up the four pulse output ports, including settings the touchscreen does not offer,
- run meter tests and standards tests, and read the results, and
- check status and errors, so an automated test system knows the standard is healthy.

This note shows how to connect, how the commands are built, and how to complete the most common tasks, first with the RADIAN SCPI Console and then from a Python script. The RX also keeps an RS-232 port for the legacy RADIAN RD command set; this note covers the Ethernet SCPI interface only.

What you need

Item	Notes
A RADIAN RX reference standard	Powered on, with its Ethernet port available. The examples use a three-phase RX-30, RX-31, or RX-33.
Ethernet cable (RJ-45)	A standard patch cable. Connect directly to a PC or through a network switch.
Windows PC	For the SCPI Console. A USB-to-Ethernet adapter works if the PC has no network port.
RADIAN SCPI Console	Free Windows program. To request it, email technicalsupport@radianresearch.com with your RX model and serial number.
Python 3.8 or later (optional)	For the automation example in Section 7. Uses only the standard library.

Connect the RX to your PC

1. Plug in the cable

Connect an Ethernet cable to the Ethernet port on the back of the RX and plug the other end into your PC or network switch.

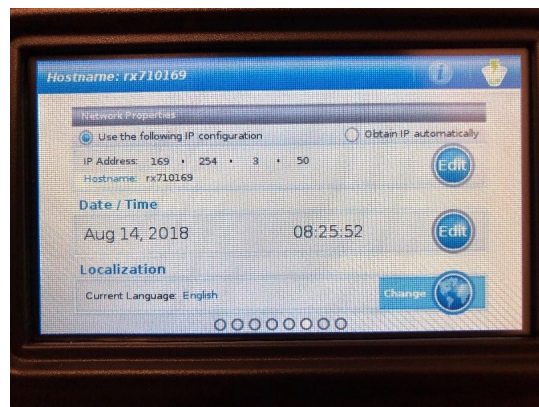


Ethernet port on the back of an RX standard.

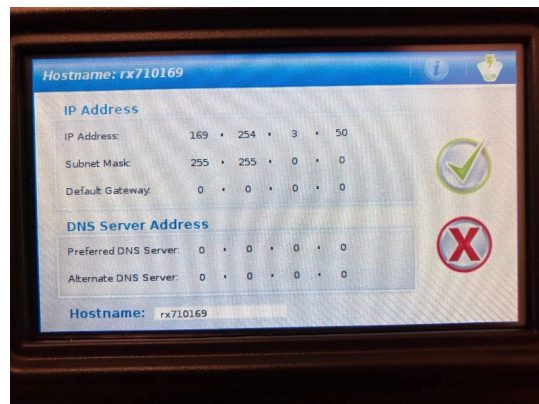
2. Set the RX network address

The IP address of the RX is shown, and can be changed, on the touchscreen under **Network Properties**. You can either use a fixed (static) address or let your network assign one automatically (DHCP).

For a direct PC-to-RX connection, a fixed address is simplest. In this example the RX is set to **169.254.3.50** with a subnet mask of **255.255.0.0**.



Network Properties on the RX touchscreen.

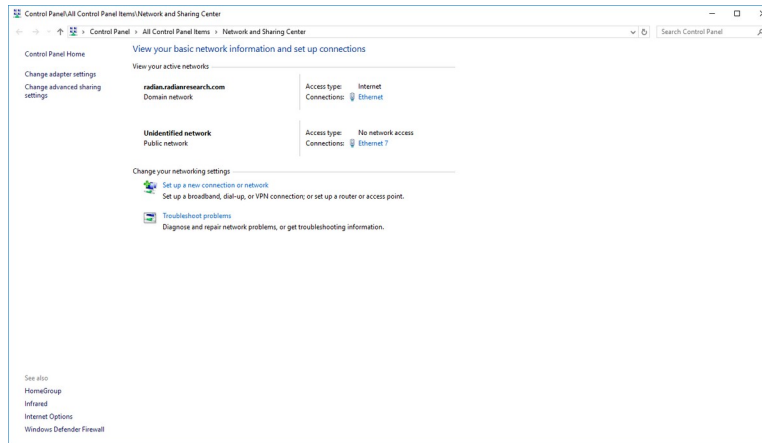


Editing the IP address and subnet mask.

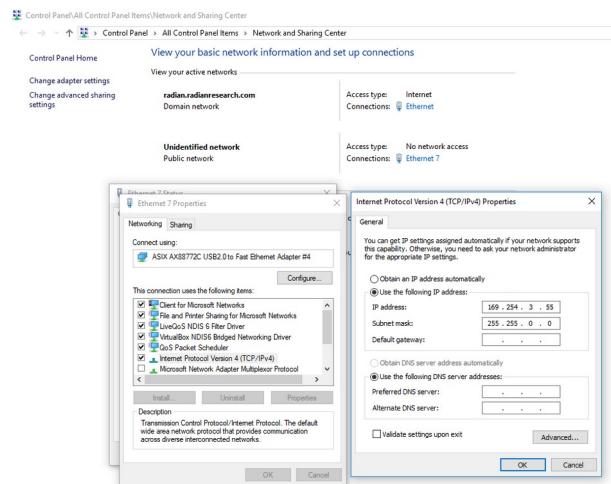
3. Set the PC network address (direct connection only)

If the RX is connected straight to your PC, give the PC's network adapter a fixed address on the same network. In Windows, open **Network and Sharing Center**, select the adapter, then choose **Properties** →

Internet Protocol Version 4 (TCP/IPv4) → Properties. Use an address that matches the RX in the first two numbers but differs in the last, such as **169.254.3.55**, with the same subnet mask, **255.255.0.0**.



Choosing the network adapter in Windows.



Setting a fixed IPv4 address for the adapter.

4. Know the ports

Network port	Purpose
5025 (TCP)	SCPI commands and responses
5026 (TCP)	Device Clear. Sending the text DCL here clears a stuck connection or a query that is waiting too long.

One client at a time. The RX accepts one SCPI connection at a time. Close the SCPI Console before running a script, and close the script before opening the Console.

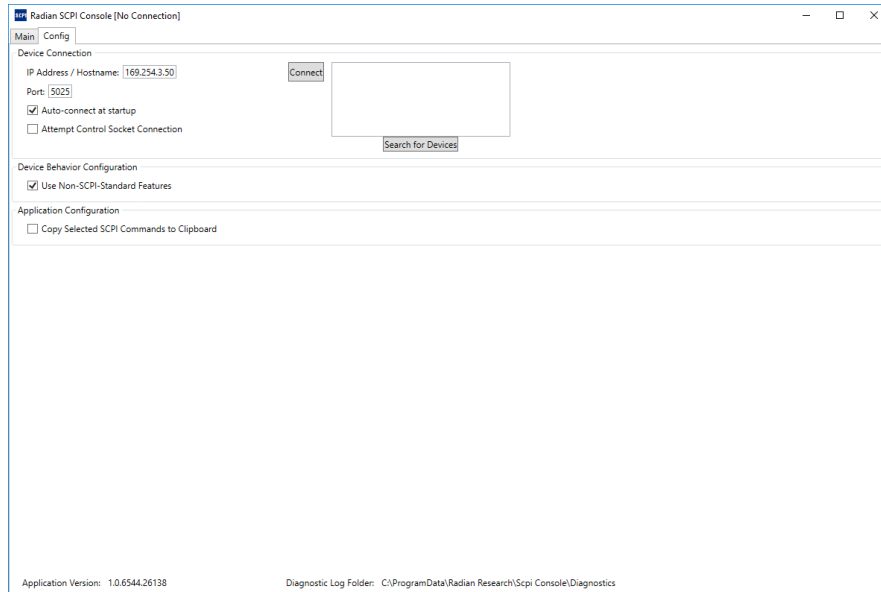
Finding an RX on your network. The SCPI Console's **Search for Devices** button lists every RX on the local network. The RX also advertises itself as a **_scpi-raw._tcp** network service (mDNS/Bonjour, the convention used by LXI instruments), so discovery tools and many test frameworks can find it automatically.

The RADIAN SCPI Console

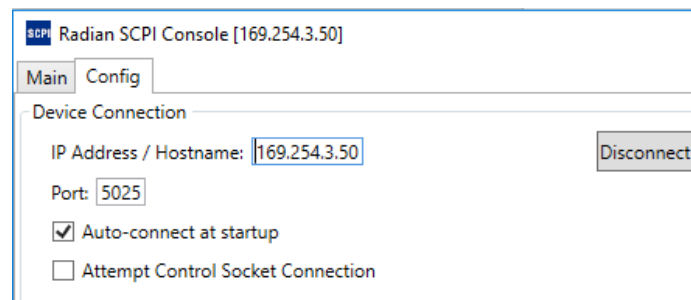
The SCPI Console is a Windows program for learning and testing the RX command set. It lists every command the connected RX supports, shows the syntax and help for each, lets you write and send a sequence of commands, and shows the responses.

Connect

1. Open the **Config** tab.
2. Enter the RX IP address (for example, 169.254.3.50).
3. Set **Port** to **5025**.
4. Check **Use Non-SCPI-Standard Features** and **Attempt Control Socket Connection**.
5. Click **Connect**. The RX address appears in the title bar and the button changes to **Disconnect**.

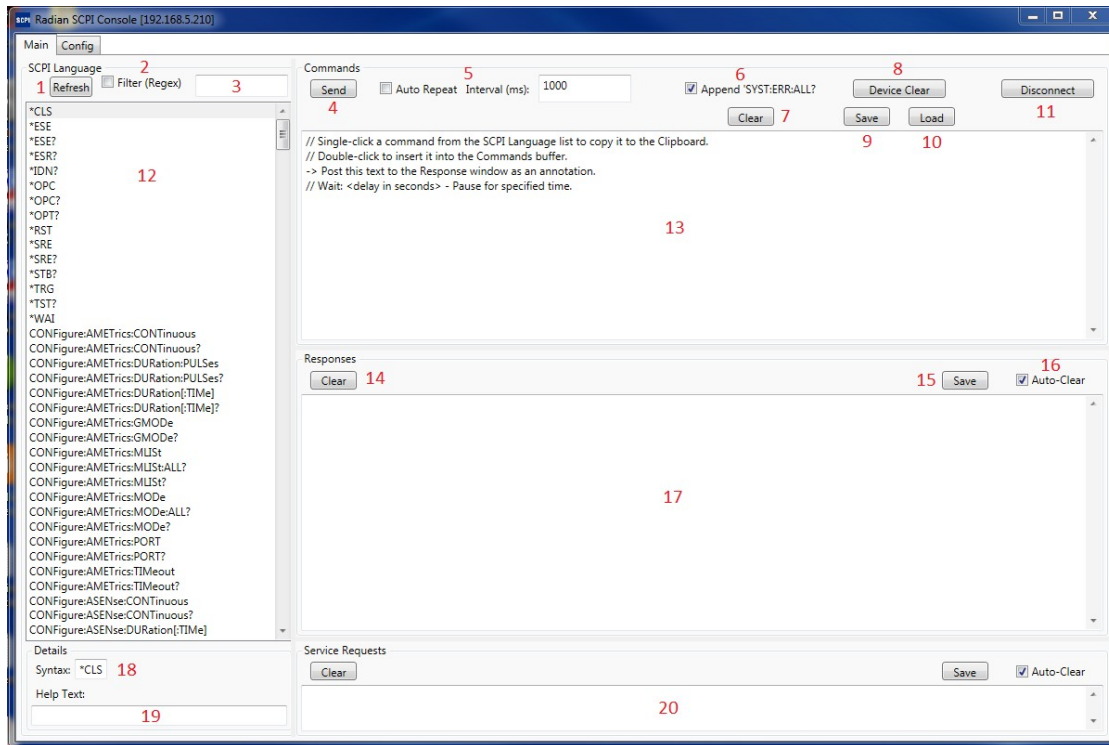


Config tab settings for an RX at 169.254.3.50.



Connected: the RX address in the title bar and the Disconnect button.

Main tab

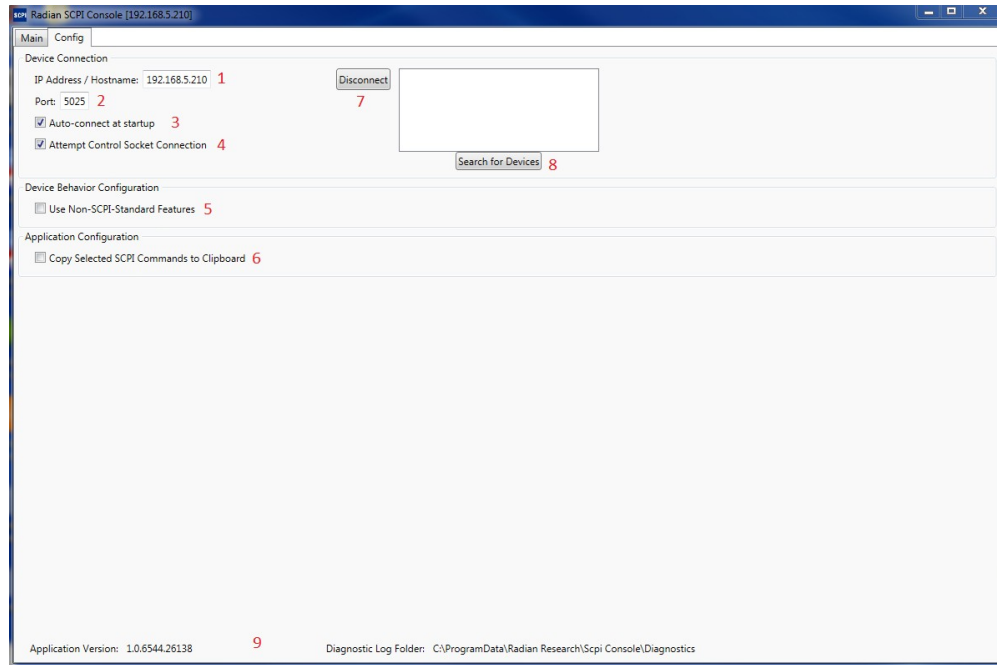


SCPI Console Main tab.

#	Control	What it does
1	Refresh	Reloads the command list (12).
2	Filter (Regex)	When checked, the command list shows only commands that match the filter text (3).
3	Filter text	The text pattern to match, written as a regular expression.
4	Send	Sends everything in the Commands window (13) to the RX.
5	Auto Repeat	Re-sends the commands at the interval shown, in milliseconds.
6	Append SYST:ERR:ALL?	Adds an error-queue query after your commands, so any errors are reported.
7	Clear (Commands)	Clears the Commands window.
8	Device Clear	Sends a Device Clear to the RX. Use it to cancel a query that is waiting too long, such as a FETCH for a measurement that was never started.
9	Save (Commands)	Saves the Commands window to a text file.
10	Load (Commands)	Loads commands from a text file.
11	Connect / Disconnect	Connects to or disconnects from the RX set on the Config tab.
12	Command list	Every command the RX accepts. Double-click a command to insert it into the Commands window.
13	Commands window	The commands to send.
14	Clear (Responses)	Clears the Responses window.
15	Save (Responses)	Saves the responses to a text file.
16	Auto-Clear	Clears the Responses and Service Requests windows each time you send.
17	Responses window	Replies from the RX.
18	Syntax	The full syntax of the selected command.
19	Help Text	A description of the selected command. Requires Use Non-SCPI-Standard Features on the Config tab.

#	Control	What it does
20	Service Requests	Service requests (SRQs) from the RX and Device Clear events.

Config tab



SCPI Console Config tab.

#	Control	What it does
1	IP Address / Hostname	The RX to connect to.
2	Port	Must be 5025 for the RX.
3	Auto-Connect at Startup	Connects to this RX each time the Console opens.
4	Attempt Control Socket Connection	Opens a second connection for service requests and Device Clear. Keep this checked.
5	Use Non-SCPI-Standard Features	Turns on RX-specific features, such as the Help Text box. Keep this checked.
6	Copy Selected SCPI Commands to Clipboard	Copies a command to the clipboard when you select it.
7	Connect / Disconnect	Connects to or disconnects from the RX.
8	Search for Devices	Lists the RX standards found on the local network.
9	Additional Information	The Console version and the location of its diagnostic logs.

Console script features

These work in the Console's Commands window only, not in your own software:

- `// text` is a comment and is not sent.
- `-> text` prints the text in the Responses window, which is useful for labeling results.
- Double-click a command in the list to insert it; single-click to copy it (if option 6 is on).

How RX commands work

Reading the command list

- **Long and short forms.** Each command word has a long form and a short form, shown by the capital letters. `CONFigure:AMETrics:DURation:PULSes?` and `conf:amet:dur:puls?` are the same command.
- **Not case-sensitive.** `SYST:PORT4:KH?` and `syst:port4:kh?` are the same.
- **Queries end in ?.** A command ending in ? returns a reply. Commands without ? change a setting and return nothing.
- **Common commands start with *.** These are the IEEE 488.2 commands every SCPI instrument supports, such as `*IDN?` (identify) and `*CLS` (clear status). Everything else is specific to the RX.
- **Several commands on one line.** Separate them with a semicolon (;).
- **Numbered parts.** Many commands include a number that picks the input, port, or test:

Subsystem	Number range	Example
Instantaneous metrics (<code>IMETrics</code>), accumulated metrics (<code>AMETrics</code>), harmonics, waveform	1 to 3 (inputs on a three-phase RX)	<code>FETCh:IMETrics2:ALL?</code>
Pulse output ports (<code>SYSTem:PORT</code>)	1 to 4	<code>SYSTem:PORT4:KH?</code>
Meter test (<code>MTEST</code>)	1 to 5	<code>INITiate:MTEST1</code>
Standards test (<code>STEST</code>)	1 to 4	<code>IRESult:STEST1:ALL?</code>
Burden test (<code>BTEST</code>)	1 to 3	<code>READ:BTEST1?</code>

- **“No reading” value.** A value of `9.91E+37` means Not a Number: there is no valid reading yet, for example before a measurement has run or when no signal is applied.

The measurement pattern

Every measurement on the RX follows the same four steps:

Step	Command	What happens
1. Configure	<code>CONFigure:<subsystem>:...</code>	Choose what to measure and how: metrics, duration, Kh, and so on.
2. Start	<code>INITiate:<subsystem></code>	The RX starts measuring.
3. Wait	<code>INITiate:<subsystem>:STATe?</code>	Replies <code>MEASuring</code> while running and <code>RAVailable</code> (results available) when finished.
4. Read	<code>FETCh:<subsystem>...?</code> or <code>IRESult:<subsystem>...?</code>	Returns the results.

`READ:<subsystem>?` combines steps 2 to 4 in one query. To stop a run early, send `INITiate:<subsystem>:STOP`.

Two kinds of reset

- `*RST` puts the RX in a state suited to programmatic control. Use it at the start of an automated test.
- `SYSTem:PRESet` puts the RX in a state suited to manual use from the touchscreen, with instantaneous metrics measuring continuously. Use it before handing the RX back to an operator.

Checking for errors

`SYSTem:ERRor:ALL?` returns every error in the queue and empties it; `SYSTem:ERRor:COUNt?` returns how many are waiting. An automated system should check the queue after each configuration step, because a command the RX did not accept does not return a reply on its own.

Asking the RX what it supports

The command set can grow with firmware updates, so ask the RX rather than relying on a fixed list:

- `SYSTem:HELP:HEADers?` lists every command the RX accepts.
- `SYSTem:PORT:AMETrics:ALL?` lists the metrics a pulse port can output.
- `CONFigure:MTEST:ELEMEnts:ALL?` and `CONFigure:MTEST:METRic:ALL?` list the meter test choices.

Common tasks with the SCPI Console

Paste each script into the Commands window and click **Send**. Lines starting with `//` are comments.

Identify the RX and check for errors

```
// Clear status, identify the RX, and check the error queue
*CLS
*IDN?
SYSTem:VERSion?
SYSTem:ERRor:ALL?
```

`*IDN?` returns the manufacturer, model, serial number, and firmware version.

Read live measurements

```
// Choose the metrics, a 1-second integration time, and continuous mode
CONFigure:IMETrics:MLISt (VOLTS,AMPS,WATTs,VAR,VA,PFACTOR,PHASe,FREQUENCY)
CONFigure:IMETrics:ITIME 1
CONFigure:IMETrics:CONTinuous ON
INITiate:IMETrics
// Read input 1, then the totalized values
FETCh:IMETrics1:ALL?
FETCh:IMETrics:TOTalized:ALL?
```

The first value in each reply is an integrity code (0 means good); the readings follow in the order of the metric list. Replace `IMETrics1` with `IMETrics2` or `IMETrics3` for the other inputs. `FETCh:IMETrics1:MINimum:ALL?` and `FETCh:IMETrics1:MAXimum:ALL?` return the lowest and highest values since the run started, and `CONFigure:IMETrics:CEXTrema` clears them.

Set up a pulse output port

The touchscreen shows the pulse port settings but cannot change all of them. This script sets **port 4** to output **totalized Wh** with a **Kh of 0.00001**, then reads each setting back to confirm.

```
// Port 4 outputs a metric (rather than off or a fixed frequency)
SYSTem:PORT4:OUTPut METRic
SYSTem:PORT4:OUTPut?
// The metric: watthours
SYSTem:PORT4:METRic WH
SYSTem:PORT4:METRic?
// The source: the total of all three inputs
SYSTem:PORT4:SOURce TOTalized
SYSTem:PORT4:SOURce?
// The pulse constant (Kh), in Wh per pulse
SYSTem:PORT4:KH 0.00001
SYSTem:PORT4:KH?
```

Commands ☐ Auto Repeat Interval (ms): 1000 ☒ Append SYSTERRALL?

```
// Reference for available metrics
SYSTem:PORT4METrics:ALL?

// Set the port to output metric (as opposed to off or frequency)
SYSTem:PORT4:OUTPut metric

// Confirms the port is set to Metric
SYSTem:PORT4:OUTPut?

// Set the port to WH
SYSTem:PORT4:METric WH

// Confirms the port is set to WH
SYSTem:PORT4:METric?

//Set the port to Totalized
SYSTem:PORT4:SOURCE Totalized

//Confirms that the port is Totalized
SYSTem:PORT4:SOURCE?
```

Commands pasted into the Commands window.

Commands ☐ Auto Repeat Interval (ms): 1000 ☒ Append SYSTERRALL?

```
// Reference for available metrics
SYSTem:PORT4METrics:ALL?

// Set the port to output metric (as opposed to off or frequency)
SYSTem:PORT4:OUTPut metric

// Confirms the port is set to Metric
SYSTem:PORT4:OUTPut?

// Set the port to WH
SYSTem:PORT4:METric WH

// Confirms the port is set to WH
SYSTem:PORT4:METric?

//Set the port to Totalized
SYSTem:PORT4:SOURCE Totalized

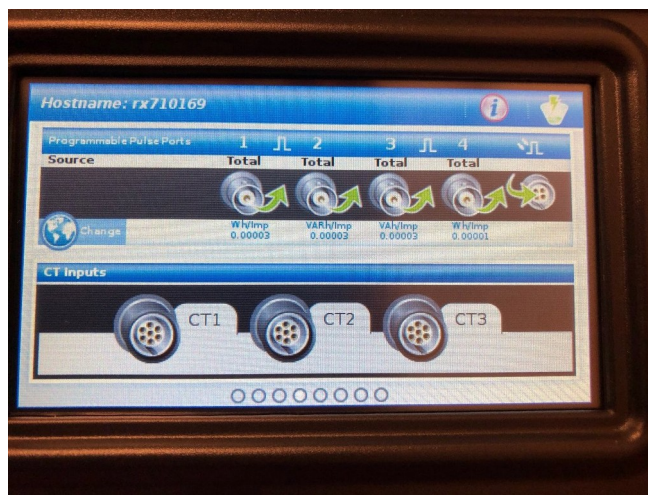
//Confirms that the port is Totalized
SYSTem:PORT4:SOURCE?
```

Responses ☒ Auto-Clear

```
WH,VARH,VAH,VH,AH,VZH,AZH,WHPos,WHNeg,VARHPos,VARHNeg,WHDelta,TIME,VAHDelta,VARHDelta,VARHDX,VARHX,WHDPos,WHDNeg,VARHDPos,VARHDNeg,VARHDXPos,VARHDXNeg,VARHXPos,VARHDXNeg,WHFund,VARHFund,VAHFund,VHFund,AHFund,VZH
METric
WH
Totalized
1.00000000000000E-05
0,"No Error"
```

Responses confirming the new settings.

The touchscreen's **Programmable Pulse Ports** screen now shows port 4 as Total Wh at 0.00001 Wh per pulse.



Pulse port 4 set to Total Wh on the RX touchscreen.

To set up a different port or metric, change these parts of the script:

Setting	Command	Choices
Port	SYSTem:PORT <i>n</i>	1, 2, 3, or 4
Output mode	:OUTPut	METRic (pulses proportional to a

Setting	Command	Choices
		metric), FREQUENCY (fixed frequency, set with :FREQUENCY), SYNCOUt (synchronization output), or OFF
Metric	:METRIC	Common choices are WH , VARH , VAH , QH , VH , AH , V2H , and A2H , plus signed and delta versions such as WHPos , WHNeg , and WHDelta . Send SYSTem:PORT:AMETrics:ALL? for the full list your firmware supports.
Source	:SOURce	TOTalized (sum of all inputs), or INPut1 , INPut2 , or INPut3 for a single input
Pulse constant	:KH	A number, or MINimum , MAXimum , or DEFault
Polarity	:POLarity	POSitive or NEGative
Pull-up impedance	:PULLup:IMPedance	HIGH or LOW

[SYSTem:PORT:RESet](#) returns every port to its factory settings.

Accumulate energy over a set time

```
// Accumulate Wh, VARh, and VAh for 60 seconds
CONFIGure:AMETrics:MLISt (WH,VARH,VAH)
CONFIGure:AMETrics:DURation 60
INITiate:AMETrics
// Repeat this query until it no longer replies MEASuring
INITiate:AMETrics:STATe?
// Read the totals for all inputs, then input 1 alone
FETCh:AMETrics:TOTalized:ALL?
FETCh:AMETrics1:ALL?
```

To stop after a pulse count instead of a time, use [CONFIGure:AMETrics:DURation:PULSes](#).
[INITiate:AMETrics:STOP](#) ends a run early.

Run a meter test

The RX compares the meter's pulses against its own measurement. This example tests a single-element meter on input 1 for 10 pulses at a Kh of 1.0.

```
// Configure meter test 1
CONFIGure:MTEST1:METRIC WH
CONFIGure:MTEST1:KH 1.0
CONFIGure:MTEST1:ELEMents ELEM1
CONFIGure:MTEST1:PHASe P1
CONFIGure:MTEST1:DURation:PULSes 10
// Start, wait, and read the result
INITiate:MTEST1
INITiate:MTEST1:STATe?
IRESult:MTEST1:ALL?
```

Element choices are [ELEM1](#), [ELEM2](#), [ELEM2_5](#), and [ELEM3](#); phase choices are [P1](#), [P2](#), [P3](#), and [TOTalized](#). A test can also stop after a time ([:DURation:TIME](#)) or a number of revolutions ([:DURation:REVolutions](#)). Up to five meter tests ([MTEST1](#) to [MTEST5](#)) can be set up at once.

Automating with Python

Any software that can open a TCP connection can control the RX: Python, LabVIEW, C#, MATLAB, or a test executive. Each command is one line of plain text ending with a newline character, and each query returns one line.

The script below, **rx_scpi_example.py**, is supplied with this note. It connects, identifies the RX, reads live measurements on all three inputs, sets pulse port 4, accumulates energy for a set time, and prints the error queue. It uses only the Python standard library.

To run it:

1. Close the SCPI Console (the RX accepts one connection at a time).
2. Open a command prompt in the folder that holds the script.
3. Run `python3 rx_scpi_example.py --host 169.254.3.50`, using your RX's address. Add `--skip-port-setup` to leave the pulse ports unchanged.

Sample output:

```
Connected to: Radian Research,RX-31,<serial number>,<firmware version>
```

Live readings

Input	VOLTs	AMPS	WATTS	PFACTOR	FREQUENCY
1	120	5	600	1	60
2	120	5	600	1	60
3	120	5	600	1	60

Pulse port 4 settings

```
OUTPut METRic
METRic WH
SOURce TOTalized
KH 0.00001
```

```
Accumulating for 10 s...
```

The core idea, in a few lines:

```
import socket

rx = socket.create_connection(("169.254.3.50", 5025), timeout=5)
reader = rx.makefile("r", encoding="ascii", newline="\n")

def send(command):
    """Sends one command line (no reply expected)."""
    rx.sendall((command + "\n").encode("ascii"))

def query(command):
    """Sends a query and returns its one-line reply."""
    send(command)
    return reader.readline().strip()

send("*CLS")
print(query("*IDN?"))          # Radian Research,RX-31,...
send("SYSTem:PORT4:METRic WH") # set pulse port 4 to Wh
print(query("SYSTem:PORT4:METRic?")) # WH
print(query("SYSTem:ERRor:ALL?")) # error queue
rx.close()
```

The full script in **Appendix A** adds what a production system needs: clearing old replies before each query, retrying the identity check, polling for completion, parsing numeric replies, and reporting errors.

Good practice for automated systems

- **Clear old replies before each query.** The RX keeps unread replies across connections. If an earlier program disconnected without reading its answers, they arrive first and every answer after them is one line out of step. The script's `drain()` function handles this.
- **Confirm the connection with `*IDN?`.** The reply always begins with "Radian". If it doesn't, discard it and ask again.
- **Poll `STATE?` rather than waiting a fixed time.** It tells you exactly when results are ready.
- **Check `SYSTEM:ERROR:ALL?` after configuring.** A command the RX did not accept is reported only in the error queue.
- **Stop what you start.** Send `INITiate:<subsystem>:STOP` when you no longer need a continuous measurement, so it does not affect the next test.
- **Close the connection when finished,** so the SCPI Console or another program can connect.

Troubleshooting

Symptom	Likely cause	What to do
Connection refused on port 5025	Another program is still connected, or an earlier session did not close cleanly.	Close the other program. If no other program is open, send <code>DCL</code> to port 5026 (or click Device Clear in the Console), wait a moment, and try again. Power-cycling the RX also clears it.
Can't connect at all	The PC and RX are on different networks.	Check that the first numbers of both IP addresses match and the subnet masks are the same. Try Search for Devices in the Console.
Replies don't match the questions	Old replies from an earlier session are queued.	Send <code>*CLS</code> , then send <code>*IDN?</code> until the reply begins with "Radian". In software, clear the receive buffer before each query.
A query never returns	It is waiting for a measurement that was never started.	Use Device Clear (or send <code>DCL</code> to port 5026). Make sure <code>INITiate</code> was sent first.
Values of 9.91E+37	No valid reading yet.	Check that a measurement has run and a signal is applied.
A setting didn't change	The RX did not accept the command or value.	Send <code>SYSTEM:ERROR:ALL?</code> to see why, and check the choices with the matching <code>...:ALL?</code> query.
Help text is blank in the Console	RX-specific features are off.	Check Use Non-SCPI-Standard Features on the Config tab.

Command quick reference

Subsystem	Use it to	Key commands
IEEE 488.2 common	Identify, reset, and clear status	<code>*IDN?</code> , <code>*RST</code> , <code>*CLS</code> , <code>*OPC?</code> , <code>*TST?</code>
<code>CONFigure:IMETrics</code>	Set up live measurements	<code>:MLISt</code> , <code>:ITIME</code> , <code>:CONTinuous</code> , <code>:CEXTrema</code>
<code>CONFigure:AMETrics</code>	Set up energy accumulation	<code>:MLISt</code> , <code>:DURation</code> , <code>:DURation:PULSes</code>
<code>CONFigure:MTES tn</code>	Set up a meter test	<code>:METRic</code> , <code>:KH</code> , <code>:ELEMEnts</code> , <code>:PHASe</code> , <code>:DURation:PULSes</code>
<code>CONFigure:STES tn</code>	Set up a standards test	<code>:METRic</code> , <code>:KH</code> , <code>:PHASe</code> , <code>:DURation:PULSes</code>
<code>INITiate</code>	Start, check, or stop a run	<code>INITiate:<subsystem></code> , <code>:STATE?</code> , <code>:STOP</code>

Subsystem	Use it to	Key commands
FETCH / IRESult	Read results	FETCH:IMETricsn:ALL?, FETCH:AMETrics:TOTalize:ALL?, IRESult:MTESTn:ALL?
READ	Start and read in one step	READ:IMETricsn?, READ:MTESTn?
SYSTem:PORT	Set up the pulse output ports	:OUTPut, :METRic, :SOURce, :KH, : POLarity, :RESet
SYSTem:ERRor	Read the error queue	:ALL?, :COUNT?
SYSTem:COMMunicate:LAN	View or change network settings	:IPADdress, :SUBNet, :DHCP, :HOS Tname, :APPLY
SYSTem:HELP	List the supported commands	:HEADers?
SYSTem	Version, date, and presets	:VERSion?, :DATetime, :PRESet

The SCPI-99 standard is published by the IVI Foundation at ivifoundation.org.

Support

For the SCPI Console, help with automation, or questions about this note, contact RADIANT Technical Support:

- **Email:** technicalsupport@radianresearch.com
- **Phone:** (765) 449-5500
- **Web:** radianresearch.com/support

Appendix A: rx_scpi_example.py

```
#!/usr/bin/env python3
"""
rx_scpi_example.py - Control a RADIANT RX reference standard over SCPI.

Companion script for the Radian Research application note
"Controlling RADIANT RX Reference Standards with SCPI".

What it does, in order:
1. Connects to the RX on TCP port 5025 and confirms its identity (*IDN?).
2. Reads live volts, amps, watts, power factor, and frequency on all
   three inputs.
3. Sets pulse output port 4 to totalized Wh at Kh 0.00001 and reads the
   settings back.
4. Accumulates energy for a fixed time and prints the totals.
5. Prints anything left in the RX error queue.

Requirements: Python 3.8 or later. Uses only the standard library.

Usage:
python3 rx_scpi_example.py --host 169.254.3.50
python3 rx_scpi_example.py --host 169.254.3.50 --skip-port-setup

Notes:
* The RX accepts one SCPI client at a time. Close the SCPI Console (or any
  other program connected to the RX) before running this script.
* Commands and responses are plain ASCII text, one line each, ending in a
  newline character.
"""

import argparse
import socket
import sys
import time
```

```
# The RX reports "Not a Number" (no valid reading) as 9.91E+37, per the
# SCPI standard. Any value at or above this is treated as "no reading".
SCPI_NAN = 9.91e37
```

```
class RxScpi:
    """A minimal SCPI client for a RADIAN RX standard over TCP.

    Keeps one socket open and provides send() for commands that return
    nothing and query() for commands that return one line of text.
    """

    def __init__(self, host, port=5025, timeout=5.0):
        self.host = host
        self.port = port
        self.timeout = timeout
        self.sock = socket.create_connection((host, port), timeout=timeout)
        self._buffer = b""

    def close(self):
        """Closes the connection so another program can connect."""
        self.sock.close()

    def send(self, command):
        """Sends one command line. Use for commands that don't end in '?'. """
        self.sock.sendall((command + "\n").encode("ascii"))

    def _read_line(self):
        """Reads one newline-terminated response line from the RX. """
        while b"\n" not in self._buffer:
            chunk = self.sock.recv(4096)
            if not chunk:
                raise ConnectionError("The RX closed the connection.")
            self._buffer += chunk
        line, self._buffer = self._buffer.split(b"\n", 1)
        return line.decode("ascii", errors="replace").strip()

    def drain(self):
        """Discards any responses already waiting on the connection.

        The RX keeps unread responses across connections. If an earlier
        program disconnected without reading its replies, they arrive first
        and shift every answer by one line. Draining before each query keeps
        questions and answers paired.
        """
        self._buffer = b""
        self.sock.setblocking(False)
        try:
            while True:
                if not self.sock.recv(4096):
                    break
        except (BlockingIOError, socket.timeout):
            pass
        finally:
            self.sock.setblocking(True)
            self.sock.settimeout(self.timeout)

    def query(self, command):
        """Sends a query (a command ending in '?') and returns its reply. """
        self.drain()
        self.send(command)
        return self._read_line()

    def parse_numbers(reply):
        """Turns a comma-separated reply into a list of floats.
```

```

Values of 9.91E+37 (SCPI "Not a Number") become None.
"""
values = []
for field in reply.replace("(", "").replace(")", "").split(","):
    try:
        number = float(field)
    except ValueError:
        continue # Skip text fields such as status words.
    values.append(None if abs(number) >= SCPI_NAN else number)
return values

def wait_until_done(rx, subsystem, timeout_s=120.0, poll_s=0.25):
    """Polls INITiate:<subsystem>:STATE? until the run is no longer measuring.

    The RX answers MEASuring while a run is in progress and RAVailable
    ("results available") when it has finished.
    """
    deadline = time.monotonic() + timeout_s
    while time.monotonic() < deadline:
        state = rx.query(f"INITiate:{subsystem}:STATE?").upper()
        if not state.startswith("MEAS"):
            return state
        time.sleep(poll_s)
    raise TimeoutError(f"{subsystem} run did not finish within {timeout_s} s.")

def identify(rx):
    """Clears the status registers and confirms the RX is answering."""
    rx.send("*CLS")
    # Retry a few times in case old replies from a previous client arrive first.
    for _ in range(3):
        idn = rx.query("*IDN?")
        if idn.lower().startswith("radian"):
            return idn
    raise RuntimeError(f"Unexpected *IDN? reply: {idn!r}")

def read_live_metrics(rx, metrics=("VOLTS", "AMPS", "WATTS", "PFACTOR", "FREQUENCY")):
    """Configures and reads instantaneous metrics on inputs 1 to 3."""
    rx.send(f"CONFIGure:IMETrics:MLIST ({','.join(metrics)})")
    rx.send("CONFIGure:IMETrics:ITime 1") # 1-second integration time
    rx.send("CONFIGure:IMETrics:CONTinuous ON") # keep measuring
    rx.send("INITiate:IMETrics")
    time.sleep(1.5) # allow the first integration period to finish

    print("\nLive readings")
    print("Input " + " ".join(f"{m:>12}" for m in metrics))
    for channel in (1, 2, 3):
        values = parse_numbers(rx.query(f"FETCH:IMETrics{channel}:ALL?"))
        # The first field is an integrity code (0 = good); readings follow.
        readings = values[1:1 + len(metrics)]
        cells = ["-" if v is None else f"{v:.6g}" for v in readings]
        print(f"{channel:>5} " + " ".join(f"{c:>12}" for c in cells))

def set_pulse_port(rx, port=4, metric="WH", source="TOTalized", kh="0.00001"):
    """Sets a pulse output port and reads each setting back."""
    rx.send(f"SYSTem:PORT{port}:OUTPut METRic")
    rx.send(f"SYSTem:PORT{port}:METRic {metric}")
    rx.send(f"SYSTem:PORT{port}:SOURce {source}")
    rx.send(f"SYSTem:PORT{port}:KH {kh}")

    print(f"\nPulse port {port} settings")
    for setting in ("OUTPut", "METRic", "SOURce", "KH"):

```

```

print(f" {setting:<7} {rx.query(f'SYSTEM:PORT{port}:{setting}?')}")

def accumulate_energy(rx, seconds=10, metrics=("WH", "VARH", "VAH")):
    """Accumulates energy for a fixed time and prints the totalized results."""
    rx.send(f"CONFIGure:AMETrics:MLIST ({','.join(metrics)})")
    rx.send(f"CONFIGure:AMETrics:DURAtion {seconds}")
    rx.send("INITiate:AMETrics")
    print(f"\nAccumulating for {seconds} s...")
    state = wait_until_done(rx, "AMETrics", timeout_s=seconds + 30)
    values = parse_numbers(rx.query("FETCh:AMETrics:TOTalized:ALL?"))
    print(f" Run state: {state}")
    print(f" Totalized reply values: {values}")

def print_errors(rx):
    """Reads and prints the RX error queue (empty means no errors)."""
    print(f"\nError queue: {rx.query('SYSTEM:ERRor:ALL?')}")

def main():
    parser = argparse.ArgumentParser(description=__doc__.split("\n")[1])
    parser.add_argument("--host", required=True, help="IP address or hostname of the RX")
    parser.add_argument("--port", type=int, default=5025, help="SCPI port (default 5025)")
    parser.add_argument("--seconds", type=int, default=10, help="accumulation time in seconds")
    parser.add_argument("--skip-port-setup", action="store_true",
                        help="don't change pulse port 4")
    args = parser.parse_args()

    try:
        rx = RxScpi(args.host, args.port)
    except OSError as err:
        sys.exit(f"Could not connect to {args.host}:{args.port} ({err}). "
                "Check the IP address and close any other program using the RX.")

    try:
        print(f"Connected to: {identify(rx)}")
        read_live_metrics(rx)
        if not args.skip_port_setup:
            set_pulse_port(rx)
        accumulate_energy(rx, seconds=args.seconds)
        print_errors(rx)
    finally:
        rx.close()

if __name__ == "__main__":
    main()

```