Supplement to Operating Manual
R&S Signal Generator

Firmware Version 2.05.200 and higher

The new firmware version for the R&S Signal Generator offers Remote Control via LAN Interface using Telnet protocol as a new functionality that could not be reported yet in the current operating manual.The following description is to provide you with comprehensive information about these new features.

Table of contents

Remote Control via LAN Interface using Telnet protocol

The instrument is equipped with the following interfaces for remote control:

- IEC/IEEE bus interface according to standard IEC 625.1/IEEE 488.2.
- LAN interface: the network card uses 10/100/1000Mbps Ethernet IEEE 802.3u; two protocols are supported:
  - the VXI-11 standard, using VISA
  - a simple telnet protocol, also called "Raw Ethernet".

The connectors are located at the rear of the instrument and permit a connection to a controller for remote control either via GPIB or via a local area network (LAN).
A VISA installation on the controller is a prerequisite for remote control over LAN (when using VXI-11 protocol).

Another alternative way to remote control the instrument is using a simple telnet protocol (port 5025). Unlike using the VXI-11 protocol, no VISA installation is necessary on the remote controller side.

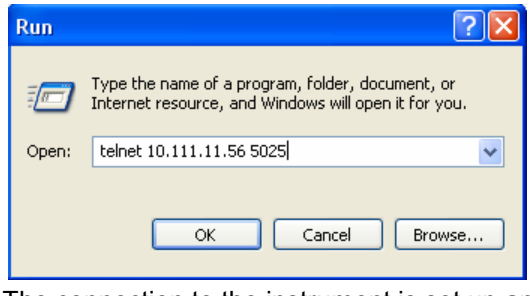
This protocol is sometimes also referred as "socket communication" or "Raw Ethernet mode".

To control the instrument manually, only a Telnet program is required. The Telnet program is part of every operating system.

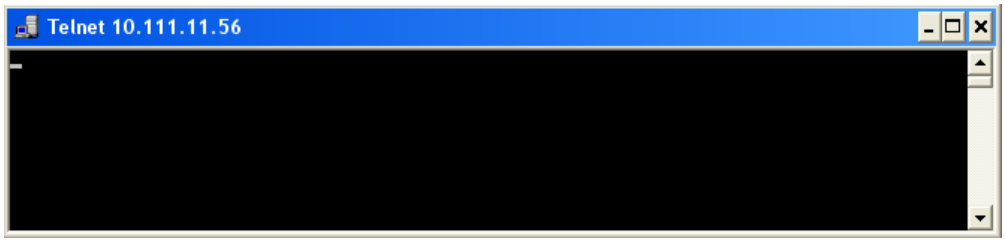
Setting up a Telnet Connection

1. To establish a Telnet connection with an instrument, start the Telnet program, enter the IP address of the R&S Signal Generator and the number of the port configured for remote-control via Telnet.

Note:
The R&S Signal Generator uses the port number 5025 for remote connection via Telnet.



The connection to the instrument is set up and remote-control commands can be sent.



2. Even if the cursor is not visible on the screan, enter blind a remote-control command and confirm with Enter.

After the fisrt remote-control command had been send, the R&S Signal Generator is in the REMOTE state, i.e. instrument control from the front panel or via mouse and keyboard is disabled and REMOTE is displayed in the status line.

Program examples

To write a program, only a socket communication must be established. The following program example shows a simple TcpClient class and how to use it.

```
#include <string>
//defines structs for socket handling
#include <netinet/in.h>
using namespace std;
typedef struct sockaddr_in SockAddrStruct;
typedef struct hostent     HostInfoStruct;
class TcpClient
{
  public:
    TcpClient();
    ~TcpClient();
    void connectToServer( string &hostname, int port );
    void disconnect( );
    void transmit( string &txString );
    void receive( string &rxString );
    string getCurrentHostName( ) const;
    int    getCurrentPort( ) const;
  private:
    string           currentHostName;
    int              currentPort;
    int              currentSocketDescr;
    SockAddrStruct   serverAddress;
    HostInfoStruct * currentHostInfo;
    bool             clientIsConnected;
    int              receiveBufferSize;
};


#include <netdb.h>
#include <netinet/in.h>
#include <unistd.h>
#include "TcpClient.h"
TcpClient::TcpClient()
: currentHostName( "" )
, currentPort( 0 )
, currentSocketDescr( 0 )
, serverAddress ( )
, currentHostInfo( NULL )
, clientIsConnected( false )
, receiveBufferSize( 1024 )
{
}
TcpClient::~TcpClient()
{
  currentHostInfo = NULL;
}

void TcpClient::connectToServer( string &hostname, int port )
{
  currentHostInfo = gethostbyname( hostname.c_str( ) );
  if( currentHostInfo == NULL )
  {
    currentHostName   = "";
    currentPort       = 0;
    currentHostInfo   = NULL;
    clientIsConnected = false;
    printf("error connecting host\n" );
  }
```

```
  currentHostName = hostname;
  currentPort     = port;
  currentSocketDescr = socket(AF_INET, SOCK_STREAM, 0);
  if( currentSocketDescr == 0 )
  {
    currentHostName   = "";
    currentPort       = 0;
    currentHostInfo   = NULL;
    clientIsConnected = false;
    printf("can't create socket\n" );
  }
  serverAddress.sin_family = currentHostInfo->h_addrtype;
  serverAddress.sin_port   = htons( currentPort );
  memcpy( (char *) &serverAddress.sin_addr.s_addr, currentHostInfo->h_addr_list[0],
currentHostInfo->h_length );
  if( connect( currentSocketDescr, ( struct sockaddr *) &serverAddress, sizeof(
serverAddress ) ) < 0 )
  {
    throw string("can't connect server\n" );
  }
  clientIsConnected = true;
}
void TcpClient::disconnect( )
{
  if( clientIsConnected )
  {
    close( currentSocketDescr );
  }
  currentSocketDescr = 0;
  currentHostName    = "";
  currentPort        = 0;
  currentHostInfo    = NULL;
  clientIsConnected  = false;
}
void TcpClient::transmit( string &txString )
{
  if( !clientIsConnected )
  {
    throw string("connection must be established before any data can be sent\n");
  }
  char * transmitBuffer = new char[txString.length() +1];
  memcpy( transmitBuffer, txString.c_str(), txString.length() );
  transmitBuffer[txString.length()] = '\n'; //newline is needed!
  if( send( currentSocketDescr, transmitBuffer, txString.length() + 1, 0 ) < 0 )
  {
    throw string("can't transmit data\n");
  }
  delete [] transmitBuffer;
}
void TcpClient::receive( string &rxString )
{
  if( !clientIsConnected )
  {
    throw string("connection must be established before any data can be received\n");
  }
  char * receiveBuffer = new char[receiveBufferSize];
  memset( receiveBuffer, 0, receiveBufferSize );
  bool receiving = true;
  while( receiving )
  {
    int receivedByteCount = recv( currentSocketDescr, receiveBuffer,
receiveBufferSize, 0 );
    if( receivedByteCount < 0 )
    {
      throw string("error while receiving data\n");
    }
```

```
    rxString += string( receiveBuffer );
    receiving = ( receivedByteCount == receiveBufferSize );
  }
  delete [] receiveBuffer;
}
string TcpClient::getCurrentHostName( ) const
{
  return currentHostName;
}
int TcpClient::getCurrentPort( ) const
{
  return currentPort;
}

#include <iostream>
#include "TcpClient.h"
void printUsage()
{
  cout<<"usage: EthernetRawCommand <server-ip> [scpi-command]"<<endl;
}
int main( int argc, char *argv[] )
{
  int errorCode         = 0; //no error
  bool useSingleCommand = false;
  string singleCommand  = "";
  string hostname       = "";
  int    port           = 5025;
  string input          = "";
  TcpClient client;
  switch( argc )
  {
    case 3:
      useSingleCommand = true;
      singleCommand    = argv[2];
    case 2:
      hostname         = argv[1];
      break;
    default:
        printUsage();
        return(-1);
  }
  try
  {
    client.connectToServer( hostname, port );
    bool terminate = false;
    while( !terminate )
    {
      char buffer[1024];
      if( useSingleCommand )
      {
        input =  singleCommand; //send string
      }
      else
      {
        cin.getline( buffer, 1024 );
        input = buffer;
        if( input == "end" )
        {
          terminate = true;
        }
      }
      if( !terminate)
      {
        client.transmit( input ); //send string
        int qPos = input.find( "?", 0 );
        //receive string only when needed
```

```
        if( qPos > 0 )
        {
          string rcStr = "";
          client.receive( rcStr );
          cout << rcStr << endl;
        }
      }
      if( useSingleCommand )
      {
        terminate = true;
      }
    }
  }catch( const string errorString )
  {
    cout<<errorString<<endl;
  }
  client.disconnect( );
  return errorCode;
}
```