

R&S®NRP-Z Power Sensor Programming Guide Application Note

Products:

- | R&S®NRP-Z
Power Sensors

The R&S®NRP-Z power sensors from Rohde & Schwarz represent the latest in power measurement technology. They offer all the functionality of conventional power meters, and more, within the small housing of a power sensor. This application note serves as a coding guide for situations in which the R&S®NRP-Z power sensors are to be used in custom test and measurement software.

Table of Contents

1	Introduction	4
2	Driver Architecture	5
3	Connecting and Disconnecting	6
3.1	Sensor Resource Strings	6
3.2	Connecting	7
3.3	Multiple Sensors	8
4	Detecting Devices	8
4.1	Device Changes	8
4.2	Reading the Device List	9
4.3	Identifying a Sensor	10
5	General Functions	11
5.1	Error Handling	11
5.2	Zeroing	12
5.3	Numeric Results	13
6	Continuous Average Power	14
7	Trace Measurements	17
7.1	Single-Shot Events	21
7.1.1	Peak Trace Data	22
7.1.2	Automatic Pulse Measurement	23
8	Statistics	25
9	Timeslot	28
9.1.1	Peak Timeslot Data	32
10	Peculiarities	33
10.1	Reading Parameter Limits	33
10.2	Reading the Trigger and Measure State	33
10.3	Checking for Level Over-Range	34
10.4	Questionable USB Data Transfer	34

11	Literature	35
12	Ordering Information	36

1 Introduction

The R&S®NRP-Z power sensors from Rohde & Schwarz represent the latest in power measurement technology. They offer all the functionality of conventional power meters, and more, within the small housing of a power sensor. The USB interface on an R&S®NRP-Z sensor enables operation with an R&S®NRP power meter or with a PC running under either Microsoft® Windows®, Mac OS X or Linux.

Particularly the capabilities for use with a desktop or laptop PC make an R&S®NRP-Z sensor an ideal and cost-effective solution for lab testing or for automated systems. The sensors' rugged design makes them suitable for use in the field for performing such tasks as servicing antenna systems.

To enable integration of the sensor into custom ATE systems, a versatile and powerful VXI PnP driver is available for Microsoft® Windows®, Mac OS® X, and Linux-based systems.

This application note serves as a coding guide for situations in which the R&S®NRP-Z power sensors are to be used in custom test and measurement software. This document is structured into sections that describe different generic functions, such as opening device connections or responding to device changes. Other chapters describe measurement applications, such as measuring average power, reading trace data or measuring the CCDF of a modulated signal.

This application note does not contain complete C code. Instead, it lists the function calls and parameters needed for an individual application and explains these functions in great detail.

R&S® is a registered trademark of Rohde & Schwarz GmbH & Co. KG.

R&S® is referred to as R&S throughout this manual.

Mac and Mac OS are trademarks of Apple Inc., registered in the U.S. and other countries.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States and/or other countries.

Trade names are trademarks of their respective owners.

© 2011 Rohde & Schwarz GmbH & Co. KG

81671 Munich, Germany

2 Driver Architecture

All coding examples in this application note are based on the functions provided in the R&S®NRP-Z VXI PnP driver. This driver uses a C function interface with VISA data types. If VISA is not installed, the driver's include file defines all required data types according to the VISA standard. Using this driver is recommended for all user applications. Please see the R&S®NRP-Z software download section on the Rohde & Schwarz website for the latest version of the VXI PnP driver [2]. Custom applications must include the rsnrpz.h file when using the VXI PnP driver. It should be noted that it is not possible for multiple applications to access the R&S®NRP low-level driver simultaneously.

The minimum requirement for using the VXI PnP driver functions is the installation of the R&S®NRP Toolkit. The toolkit package contains the USB drivers as well as the NrpControl2 low-level driver DLL.

Generally, it is possible to build applications with the rsnrpz.c and rsnrpz.h files directly compiled into the application. In this case, the application only depends on NrpControl2.lib. Alternatively, the application can include rsnrpz.h and link against rsnrpz.lib.

The following diagram shows the R&S®NRP-Z driver architecture and possible application options for 32-bit and 64-bit systems.

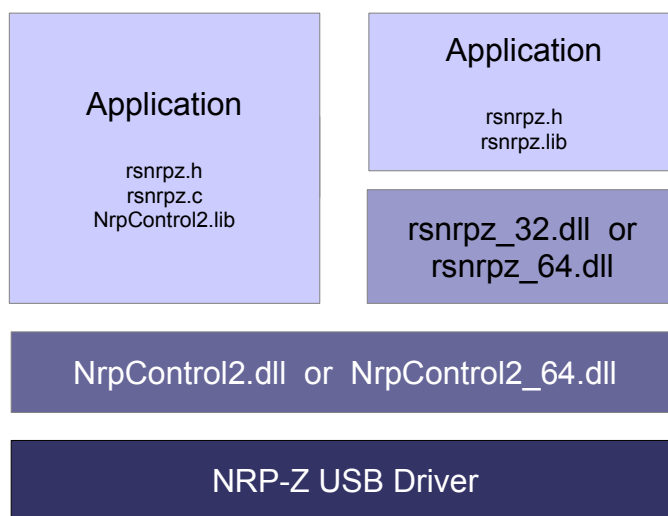


Figure: Driver architecture on Windows-based systems.

3 Connecting and Disconnecting

3.1 Sensor Resource Strings

The power sensors are identified by a unique VISA resource string. This string is passed on to the `rsnrpz_init()` function in order to open the sensor connection. The resource string has the following format:

```
USB::0x0AAD::<USB ID>::<serial>
```

The value 0x0AAD is the Rohde & Schwarz vendor ID, and it cannot be changed. The USB ID is unique for each sensor type. A list of USB device ID numbers is provided below. The serial number is the serial number for the individual sensor.

The following table provides an overview of the USB IDs for Rohde & Schwarz NRP power sensors:

Sensor	USB ID	Supported Measurement			
		Cont Av	Trace	Timeslot	Statistics
NRP-Z11	0x0C	•	•	•	
NRP-Z21	0x03	•	•	•	
NRP-Z211	0xA6	•	•	•	
NRP-Z22	0x13	•	•	•	
NRP-Z221	0xA7	•	•	•	
NRP-Z23	0x14	•	•	•	
NRP-Z24	0x15	•	•	•	
NRP-Z31	0x2C	•	•	•	
NRP-Z41	0x96	•	•	•	
NRP-Z51	0x16	•			
NRP-Z52	0x17	•			
NRP-Z55	0x18	•			
NRP-Z56	0x19	•			
NRP-Z57	0x70	•			
NRP-Z58	0xA8	•			
NRP-Z91	0x21	•			
NRP-Z81	0x23	•	•	•	•
NRP-Z85	0x83	•	•	•	•
NRP-Z86	0x95	•	•	•	•
NRP-Z27	0x2F	•			
NRP-Z37	0x2D	•			

NRP-Z28	0x51	•	•	•	
NRP-Z92	0x62	•			
NRP-Z98	0x52	•			
NRPC33	0xB6	•			
NRPC40	0x8F	•			
NRPC50	0x90	•			
NRPC33-B1	0xC2	•			
NRPC40-B1	0xC3	•			
NRPC50-B1	0xC4	•			
FSH-Z18	0x1A	•			
FSH-Z1	0x11	•			

3.2 Connecting

The first step is to open the sensor connection. The following code lines demonstrate how this is done using the driver functions. The example sets the USB timeout to 5 seconds and does not reset the sensor. The return value is the USB session that must be used in all further communication with the sensor:

```
ViStatus  lErr;
ViSession ulUSBSession;

rsnrpz_setTimeout( 5000 );

lErr = rsnrpz_init( "<USB resource string>",
                  false, false,
                  &ulUSBSession
                  );

if( lErr!=0 )
    ...error handling...
```

The `rsnrpz_init` call does not affect data collected during a previous sensor zeroing process. Thus, the zeroing remains valid when the connection is closed and then reopened.

If the sensor is no longer needed for further measurements, the connection should be closed. The following lines demonstrate how to do this. After the sensor is closed, the session number must not be used anymore:

```
lErr = rsnrpz_close( ulUSBSession );
ulUSBSession = 0;
```

3.3 Multiple Sensors

Multiple sensors may be opened simultaneously within one application. The `rsnrpz_init` function must be called once for each sensor using the sensor resource string. The USB session ID numbers returned from this function are then used to access the individual sensors.

4 Detecting Devices

4.1 Device Changes

The VXI PnP driver calls user defined callback functions if a sensor change is detected. The driver passes a value with the `NRP_SESSION` data type to this callback function. The length of this data type is 32 bits or 64 bits depending on the operating system. The argument can, therefore, be used as a function or class pointer.

```
void DeviceChangedCallback( NRP_SESSION lClassPointer );
```

The driver's callback mechanism is activated by a call to the function described below. The first argument is the pointer to the callback function itself. The second parameter is the parameter which is passed on to the callback function when it is invoked. When C++ is used, this can be the pointer to the current class (this-pointer).

```
lErr = rsnrpz_status_setDeviceChangedCallback(  
    DeviceChangedCallback,  
    (NRP_USERARG) this );
```

If sensor changes do not need to be monitored anymore, it is important to disable the driver's callback mechanism. This is done by passing zero to the callback setup function.

```
rsnrpz_status_setDeviceChangedCallback( 0, 0 );
```


When an object-oriented programming language, such as C++, is used, it might be desirable to employ a public method as the callback function. Since methods cannot be called directly, a global callback function is required. This function uses the 'this' pointer from the argument and calls the class's public method.

```
void DeviceChangedCallback( NRP_SESSION lClassPointer )
{
    if( lClassPointer==0 ) return;

    ((Class*)lClassPointer)->DeviceChangeCallback();
}

void Class::DeviceChangeCallback()
{
    // read list of sensors...
}
```

4.2 Reading the Device List

The driver provides functions that read a list of all connected R&S[®]NRP-Z sensors. Typically, the number of sensors is first determined by calling the `rsnrpz_getSensorCount` function. Then the `rsnrpz_GetSensorInfo` function is repeatedly called and provides detailed sensor information.

```
ViInt32 lCount;
lErr = rsnrpz_GetSensorCount( 0, &lCount );

char szSensorName[512];
char szSensorType[512];
char szSensorSerial[512];

i = 0 ... lCount-1

lErr = rsnrpz_GetSensorInfo( 0, i,
                           szSensorName,
                           szSensorType,
                           szSensorSerial );
```

The returned data contains the following information:

- `szSensorName` Resource String, e.g. "USB::0x0aad::0x000c::100023"
- `szSensorType` Sensor type, e.g. "NRP-Z11"
- `szSensorSerial` Serial number, e.g. "100023"

4.3 Identifying a Sensor

The SCPI standard defines the *IDN? query, which returns device information, such as the device type, manufacturer, serial number, and firmware version. The VXI PnP driver provides the `rsnrpz_chan_info` function, which reads general information from the sensor. The first two arguments to this function are the session ID and the channel number. The third parameter is the information type identifier that sets which information should be read from the sensor. The following two parameters contain the length of the data buffer and the pointer to this buffer.

A list of all supported information type identifiers are listed in the VXI PnP driver documentation [2].

```
char szTmp[256];

lErr = rsnrpz_chan_info( ulUSBSession, 1,
                        "SW Build",
                        256, szTmp );

lErr = rsnrpz_chan_info( ulUSBSession, 1,
                        "MinPower",
                        256, szTmp );
double dMinPower = atof( szTmp );

lErr = rsnrpz_chan_info( ulUSBSession, 1,
                        "MaxPower",
                        256, szTmp );
double dMaxPower = atof( szTmp );

lErr = rsnrpz_chan_info( ulUSBSession, 1,
                        "MinFreq",
                        256, szTmp );
double dMinFreq = atof( szTmp );
```

5 General Functions

5.1 Error Handling

Most `rsnrpz` driver functions return an error code. If the function call is successful, the return value is zero. If an error was returned, further information may be obtained from the `rsnrpz_error_message` function. This function translates the error code into a human-readable text message:

```
char szMessage[256];
rsnrpz_error_message( ulUSBSession,
                    lErr,
                    szMessage );
```

Additionally, errors that arise inside the sensor can be queried from the sensor error queue. The `rsnrpz_error_query` should be called as long as the `lErr2` return variable is not equal to zero and the return code is zero:

```
lErr = rsnrpz_error_query( ulUSBSession,
                        &lErr2,
                        szMessage );
```

Please note that both functions require a valid session number. If the `rsnrpz_init` function fails and no valid session number is available (session is zero), these functions cannot be used. In most cases, this indicates that the sensor has already been opened by another application, or that the sensor firmware is outdated.

5.2 Zeroing

Zeroing the sensor might be required if very low signal levels need to be measured. The time required for the zeroing procedure varies from sensor to sensor. It must also be noted that the zero offset value is not permanent. A sensor reset command does not clear the zero offset, but a power loss requires re-zeroing.

The following example starts the zeroing process with the function `rsnrpz_chan_zero`. This function returns immediately. The following call to `rsnrpz_chan_isZeroComplete` determines the completion state of the zeroing process. The function should be called repeatedly in a loop, but it must be ensured that there is enough CPU time available for the driver process (\rightarrow `Sleep()`, \rightarrow `SwitchToThread()`). Using a short sleep that gives the CPU away to the background thread is recommended. If the zeroing fails, an error code that is not equal to zero is returned:

```
lErr = rsnrpz_chan_zero( ulUSBSession, 1 );

unsigned short usMeasCompleted;

do {
    lErr = rsnrpz_chan_isZeroComplete( ulUSBSession, 1,
                                       &usMeasComplete );

    Sleep...
} while( usMeasComplete==0 && lErr==0 );
```

5.3 Numeric Results

All numeric power readings are provided in watts. The reading includes an optional level offset that is either set by the user or provided by the S-parameter device. In many applications, however, it is desirable to show power readings on a logarithmic scale. Conversion can be performed by taking the absolute value of the power reading, adding a very small offset (e.g. $1e-32$), and then using the `log10` function.

```
dValdBm = 30.0 + 10.0 * log10( fabs( dValW ) + 1e-32 );
```

A more complex approach would be to read the sensor's minimum power and limit the minimum power to this value.

```
lErr = rsnrpz_chan_info( ulUSBSession, 1,
                        "MinPower",
                        256, szTmp );
dMinPower = atof( szTmp );

if( dValW < dMinPower )
    dValW = dMinPower;
else
    dValW = dValW;

dValdBm = 30.0 + 10.0 * log10( dValW );
```

In applications that display numeric readings as logarithmic values, it is good practice to limit the minimum value to readings (such as -99.99 or -199.99) that clearly indicate that the end of the scale has been reached.

6 Continuous Average Power

This chapter describes how to implement a continuous average power measurement. In this mode, the system measures the measurement signal's average power asynchronously within definable time intervals. These measurements are performed with chopper stabilization to obtain more accurate results with reduced noise and zero offset. Therefore, a measurement is always performed over two sampling windows, with the detector output signal's polarity being reversed for the second window. Taking the difference of the output signals minimizes the video path's influence on noise and zero drift. When the averaging function is activated, the averaging factor determines how often the described measurement cycle is repeated.

First, the sensor's operation mode needs to be set. This step only needs to be performed once when multiple average power measurements are required:

```
lErr = rsnrpz_chan_mode( ulUSBSession, 1,  
                        RSNRPZ_SENSOR_MODE_CONTAV );
```

Second, the carrier frequency must be set. Setting the carrier frequency is always required for precise measurements:

```
lErr = rsnrpz_chan_setCorrectionFrequency( ulUSBSession, 1,  
                                           dCarrierHz );
```

In many cases, the power sensor is not directly connected to the DUT, and compensation must be made for additional cable loss. The two functions described below enable and set the level offset. Please note that further level-related commands (e.g. trigger level) expect levels that include the correction factor:

```
lErr = rsnrpz_corr_setOffset( ulUSBSession, 1,  
                             dOffsetdB );
```

```
lErr = rsnrpz_corr_setOffsetEnabled( ulUSBSession, 1,  
                                     true );
```

The aperture time is the time for which the sensor integrates the signal in order to generate a single sample. Normally, the sensor uses a default aperture that is best for noise and measurement speed. When measuring AM modulated signals with a known period time, it is advisable to set the aperture time to multiples of the period time. In such cases, using a low averaging filter count, such as two, can deliver stable measurement results:

```
lErr = rsnrpz_chan_setContAvAperture( ulUSBSession, 1,  
                                       dWindowS );
```

The averaging filter can be configured to either manual mode or automatic mode. The auto mode example below sets the filter to a 0.01 dB resolution. The available index values depend on the sensor and can be found in the sensor's user manual [3] under the command SENS:AVER:COUN:AUTO:RES :

```
lErr = rsnrpz_avg_configureAvgAuto( ulUSBSession, 1,  
                                    3 );
```

Alternatively, the averaging filter mode can be set to a fixed value, such as 2, 4, 8 or 16. This setting is best if a constant measurement time is required and the signal level does not change much:

```
lErr = rsnrpz_avg_configureAvgManual( ulUSBSession, 1,  
                                     ulAvCnt );
```

The averaging filter can be configured to repeating mode or to moving-filter mode. In repeating mode, each measurement cycle initially clears the filter and then accumulates measurements until the filter is entirely filled:

```
lErr = rsnrpz_avg_setTerminalControl( ulUSBSession, 1,  
                                      RSNRPZ_TERMINAL_CONTROL_REPEAT );
```

The `rsnrpz_chan_initiate` function starts one measurement cycle. The function returns immediately. Therefore, the application must subsequently poll the sensor for measurement completion:

```
lErr = rsnrpz_chan_initiate( ulUSBSession, 1 );
```

The completion state should be polled in a loop, but it must be ensured that the CPU is made available to the driver thread between subsequent polls:

```
ViBoolean bMeasCompleted

do {
    lErr = rsnrpz_chan_isMeasurementComplete(
        ulUSBSession, 1,
        &bMeasCompleted );

    Sleep( ... );
} while( !bMeasCompleted && lErr==0 );
```

When the measurement cycle has completed successfully, the result can be read, and a new measurement cycle may be started:

```
ViReal64 fMeasResult
lErr = rsnrpz_meass_fetchMeasurement( ulUSBSession, 1,
    &fMeasResult );
```


7 Trace Measurements

This chapter describes how to implement a trace measurement for a repeating signal that provides a stable trigger condition. In trace mode, the envelope power can be recorded as a function of time. This is done by sampling power over a user-specified time interval and then assigning the determined power values to a certain number of pixels. The number of pixels is largely user-selectable. The time interval that a pixel represents is obtained by dividing the trace length by $N-1$, where N is the number of pixels. In the simplest case, each pixel is assigned a single sample value which fully characterizes it. If several sample values are assigned to a pixel, the following quantities can be determined for each time interval:

- Average power
- Maximum power
- Minimum power
- A randomly selected sample value

When the averaging function is deactivated, measurements are performed without chopper stabilization, i.e. a measurement consists of a single sampling sequence activated by a trigger event. Otherwise, the detector's output-voltage polarity is reversed automatically for alternate sampling sequences. This suppresses low-frequency noise and increases the accuracy with which the average power is measured at each pixel.

First, the sensor's operation mode needs to be set. This step is only required initially:

```
lErr = rsnrpz_chan_mode( ulUSBSession, 1,
                        RSNRPZ_SENSOR_MODE_SCOPE );
```

Second, the carrier frequency must be set. Setting the carrier frequency is required for precise power measurements:

```
lErr = rsnrpz_chan_setCorrectionFrequency( ulUSBSession, 1,
                                           dCarrierHz );
```

In many cases, the power sensor is not directly connected to the DUT, and compensation must be made for additional cable loss. The following two functions enable and set the level offset:

```
lErr = rsnrpz_corr_setOffset( ulUSBSession, 1,
                              dOffsetdB );
```

```
lErr = rsnrpz_corr_setOffsetEnabled( ulUSBSession, 1,
                                      bEnOffset );
```

The function below configures the measurement bandwidth. Using a lower bandwidth decreases measurement noise and increases trigger sensitivity. The list of available bandwidth IDs depends on the sensor and can be found in the sensor's user manual under the command SENSE:BWIDth:VIDEo :

```
lErr = rsnrpz_bandwidth_setBw( ulUSBSession, 1, 0 );
```

The number of video points for the trace measurement is set using `rsnrpz_scope_setPoints`. Using 500 points usually represents a good compromise between USB transfer speed and resolution. The trace data's transfer time increases with the number of video points:

```
lErr = rsnrpz_scope_setPoints( ulUSBSession, 1,  
                               iVideoPoints );
```

The trace time sets the overall capture time for one trace measurement. Each video point represents the time period resulting from the trace time divided by the number of video points. The offset time should be set to zero before the trace time is set. This is required, because the trace time limits depend on the offset time:

```
lErr = rsnrpz_setOffsetTime( ulUSBSession, 1, 0 );
```

```
lErr = rsnrpz_scope_setTime( ulUSBSession, 1,  
                             dTraceTime );
```

The offset time is used to capture signal portions before the trigger point. The valid time range depends on the sensor and must be looked up in the sensor manual. The function call is not required if this feature is not needed. An offset time of zero starts trace capturing at the trigger position:

```
lErr = rsnrpz_scope_setOffsetTime( ulUSBSession, 1,  
                                   dOffsetTime );
```

Configuring the trigger condition is crucial for all trace measurements. The following lines configure the trigger system to internal triggering on a positive slope. The hysteresis should be set to a small value (e.g. 0.1 dB) to allow for stable triggering. The dropout time can be set optionally and requires the signal to fall below the trigger threshold for the defined period of time before the trigger system rearms. Please note that the trigger level is set in linear units.

```
lErr = rsnrpz_trigger_setSource( ulUSBSession, 1,  
                                 RSNRPZ_TRIGGER_SOURCE_INTERNAL );
```

```
lErr = rsnrpz_trigger_setSlope( ulUSBSession, 1,  
                                 RSNRPZ_SLOPE_POSITIVE );
```

```
lErr = rsnrpz_trigger_setDropoutTime( ulUSBSession, 1,  
                                       dDropoutTime );
```

```
lErr = rsnrpz_trigger_setHysteresis( ulUSBSession, 1,  
                                     0.1 );
```

```
lErr = rsnrpz_trigger_setLevel( ulUSBSession, 1,  
                                 dTrigLevelW );
```

Setting an averaging filter is, in most cases, desired when trace data is to be measured. Averaging reduces the noise dramatically and therefore increases the dynamic range:

```
lErr = rsnrpz_scope_setAverageCount( ulUSBSession, 1,  
                                      iAverageCount );
```

```
lErr = rsnrpz_scope_setAverageEnabled( ulUSBSession, 1,  
                                        true );
```

The averaging filter can be operated in either repeating or moving mode. In repeating mode, the filter content is cleared at the beginning of the measurement cycle. Once the filter is entirely filled, the measurement terminates and the result can be read:

```
lErr = rsnrpz_scope_setAverageTerminalControl(  
                                             ulUSBSession, 1,  
                                             RSNRPZ_TERMINAL_CONTROL_REPEAT );
```

The `rsnrpz_chan_initiate()` function call starts the measurement cycle and returns immediately.

```
lErr = rsnrpz_chan_initiate( ulUSBSession, 1);
```

Before any data can be read from the sensor, the measurement status must be polled repeatedly. This polling must be implemented in such a way that the CPU becomes available to the driver thread periodically:

```
ViBoolean bMeasCompleted;  
  
do {  
    lErr = rsnrpz_chan_isMeasurementComplete(  
                                                ulUSBSession, 1,  
                                                &bMeasCompleted );  
  
    Sleep( ... );  
  
} while( !bMeasCompleted && lErr==0 );
```

After the measurement has completed, the data array can be read using the `rsnrpz_meass_fetchBufferMeasurement` function. The values that are returned are in linear units and include any offset configured using `rsnrpz_corr_setOffset`. The number of trace points must match the number of video points set with `rsnrpz_scope_setPoints`:

```
ViReal64 pdMeasAv[iTracePoints];
ViInt32 iReadCount;

lErr = rsnrpz_meass_fetchBufferMeasurement(
    ulUSBSession, 1,
    iTracePoints,
    pdMeasAv,
    &iReadCount );
```

The settings from the above example return the *average* trace representation. Based on the averaging filter settings and the trace time, the sensor captures multiple samples and calculates the average trace data.

The return data is provided in linear units. However, many applications require power values on a logarithmic scale. The conversion can be done using the following equation:

$$P_{log} = 10 \cdot \log_{10}(P_{lin}) \text{ dBm} + 30 \text{ dBm}$$

Care must be taken if signal portions close to the noise floor must be converted. Depending on the zero reference point for the internal analog-to-digital converter, negative power readings may occur.

This is normal behavior, and in most cases, it is possible to simply use the linear power reading's absolute value for the `log10` function. The low power values do not typically contribute to any measurement.

In very rare cases, a power value that is exactly zero may arise. Zero cannot be converted into the logarithmic scale and must, therefore, be replaced by another value.

7.1 Single-Shot Events

Measuring single-shot events requires slightly different averaging filter settings. Please note that disabling averaging also reduces the sensor's dynamic range.

The average filter count is set to one, and the filter is disabled:

```
lErr = rsnrpz_scope_setAverageCount( ulUSBSession, 1,  
                                     1 );
```

```
lErr = rsnrpz_scope_setAverageEnabled( ulUSBSession, 1,  
                                       false );
```

Additionally, non-Z8x sensors require enabling of the real-time mode. In this mode, the chopper is turned off and only one single trace is processed. (The R&S®NRP-Z81 sensor does not require this command.):

```
lErr = rsnrpz_scope_setRealtimeEnabled( ulUSBSession, 1,  
                                        true );
```

7.1.1 Peak Trace Data

Wideband sensors, such as the R&S®NRP-Z81, provide multiple trace data representations. The AVERAGE trace representation is always sent and cannot be deselected. Alternatively, the sensor can be switched to auxiliary mode, and it can send two additional representations, such as the RANDOM and MAXIMUM trace data:

```
lErr = rsnrpz_chan_setAuxiliary( ulUSBSession, 1,  
                                RSNRPZ_AUX_RNDMAX );
```

When auxiliary data is enabled, the trace data must be read from the driver cache using the `rsnrpz_meass_fetchBufferMeasurementAux` function. In a way similar to the regular fetch function, all data is provided in linear units and contains the level offset:

```
ViReal64 pdMeasAv[iTracePoints];  
ViReal64 pdMeasRnd[iTracePoints];  
ViReal64 pdMeasPeak[iTracePoints];  
ViInt32  iReadCount;  
  
lErr = rsnrpz_meass_fetchBufferMeasurementAux(  
        ulUSBSession, 1,  
        0,  
        iTracePoints,  
        pdMeasAv, pdMeasRnd, pdMeasPeak,  
        &iReadCount  
    );
```

7.1.2 Automatic Pulse Measurement

Wideband sensors, such as the R&S®NRP-Z81, can perform automatic pulse measurements in trace mode. Enabling the automated pulse measurement increases the measurement and processing time inside the sensor.

The following two functions enable the automatic pulse measurement and set the algorithm to the histogram type. A detailed discussion of the various algorithms can be found in the Power Viewer Plus manual [1]:

```
lErr = rsnrpz_scope_meas_setMeasEnabled( ulUSBSession, 1,  
                                         true );
```

```
lErr = rsnrpz_scope_meas_setMeasAlgorithm( ulUSBSession, 1,  
                                           RSNRPZ_SCOPE_MEAS_ALG_HIST );
```

As far as timing is concerned, it is possible to limit the automatic pulse analysis to a fraction of the entire trace measurement. The offset parameter sets the starting point for the automatic pulse analysis. The total time is set with the second function. If the entire trace measurement window should be used for the automatic pulse measurement, the two parameters should match the values set by `rsnrpz_scope_setOffsetTime` and `rsnrpz_scope_setTime`:

```
lErr = rsnrpz_scope_meas_setOffsetTime( ulUSBSession, 1,  
                                         dOffset );
```

```
lErr = rsnrpz_scope_meas_setTime( ulUSBSession, 1,  
                                   dMeasTime );
```

The only three configuration parameters required by the automatic pulse analysis are the low, mid and high thresholds as a percentage of the pulse top power:

```
lErr = rsnrpz_scope_meas_setLevelThresholds( ulUSBSession, 1,  
                                              dLevMidPercent,  
                                              dLevLowPercent,  
                                              dLevHighPercent );
```

Enabling the equivalent sampling increases the automatic pulse measurement's timing resolution. In this mode, multiple traces are measured at different timing offsets, and the resulting trace is generated from a linear interpolation of the individual measurement:

```
lErr = rsnrpz_scope_meas_setEquivalentSampling(  
                                             ulUSBSession, 1,  
                                             true );
```

After completion of the trace measurement, the pulse measurement results can be read from the driver cache using the functions listed below. Values that cannot be determined are indicated using a quiet NaN.

```
#ifndef LINUX
#define __isnan(_X) (fpclassify((float)_X)==FP_NAN)
#else
#define __isnan(_X) ((_X)!=(_X))
#endif
```

The general pulse timing can be read using the `rsnrpz_scope_meas_getPulseTimes` function. The duty cycle and period time require at least two pulses to fall into the trace window. The measurement of the pulse width requires at least a rising and falling edge.

```
lErr = rsnrpz_scope_meas_getPulseTimes( ulUSBSession, 1,
                                        &dDutyCycle,
                                        &dPulseWidth,
                                        &dPeriodTime );
```

The rising and falling edge times are read using the same function twice:

```
lErr = rsnrpz_scope_meas_getPulseTransition( ulUSBSession, 1,
                                             RSNRPZ_SLOPE_POSITIVE,
                                             &dRiseTime,
                                             &dRisePosition,
                                             &dRiseOvershot );
```

```
lErr = rsnrpz_scope_meas_getPulseTransition( ulUSBSession, 1,
                                             RSNRPZ_SLOPE_NEGATIVE,
                                             &dFallTime,
                                             &dFallPosition,
                                             &dFallOvershot );
```

There is a series of functions available for the measuring the different pulse-power levels. The pulse peak power and the pulse top power are typically of greater interest:

```
lErr = rsnrpz_scope_meas_getPulsePower( ulUSBSession, 1,
                                        &dAveragePower,
                                        &dMinPeak,
                                        &dMaxPeak );
```

```
lErr = rsnrpz_scope_meas_getPulseLevels( ulUSBSession, 1,
                                        &dTopPower,
                                        &dBasePower );
```



```
lErr = rsnrpz_scope_meas_getPulseReferenceLevels(
    ulUSBSession, 1,
    &dLowRefLevel,
    &dHighRefLevel,
    &dMidRefLevel );
```

Please note that all power readings are in linear units, and they contain any level offset that was previously set.

8 Statistics

This chapter describes how to configure a statistics measurement, such as a complementary cumulative distribution function (CCDF) or a probability density function (PDF). The following measurement parameters can be set:

- Start of the analysis window
- Length of the analysis window
- Exclusion period within the analysis window
- Number of analysis window repetitions
- Video bandwidth

Statistical analysis can either be triggered by a signal or performed continuously. In the first case, analysis is synchronized to the signal characteristic, but this is not done in the second case. Instead, the second method employs a sequence of analysis windows. Analysis is terminated when the specified number of repetitions has been reached. Statistical analysis can only be performed when chopper stabilization is deactivated. The sample size, i.e. the number of samples analyzed, equals the product of the analysis-window length, the number of repetitions and the sampling rate. In turn, the sampling rate is a function of the video bandwidth that has been set. Before the analysis result can be output, the user must specify a level range and its resolution in pixels. For each pixel, either the value of the complementary cumulative distribution function or the value of the probability density function (in W^{-1}) is output. The following output parameters can be set:

- Lower limit of level range in dBm
- Width of level range in dB
- Resolution in pixels

The size of the level interval that each pixel represents is determined by dividing the width of the level range by the number of pixels minus one. The smallest possible interval size for the R&S®NRP-Z8x power sensor is specified as 0.006 dB.

First, the sensor's operation mode needs to be set. This step is only required initially:

```
lErr = rsnrpz_chan_mode( ulUSBSession, 1,
    RSNRPZ_SENSOR_MODE_CCDF );
```

Second, the carrier frequency must be set. Setting the carrier frequency is required for precise power measurements:

```
lErr = rsnrpz_chan_setCorrectionFrequency( ulUSBSession, 1,
                                           dCarrierHz );
```

In many cases, the power sensor is not directly connected to the DUT, and compensation must be made for additional cable loss. The following two functions enable and set the level offset:

```
lErr = rsnrpz_corr_setOffset( ulUSBSession, 1,
                              dOffsetdB );
```

```
lErr = rsnrpz_corr_setOffsetEnabled( ulUSBSession, 1,
                                      bEnOffset );
```

The function below configures the measurement bandwidth. Using a lower bandwidth decreases measurement noise and increases trigger sensitivity. In the statistics measurement modes, the bandwidth must be set to at least the bandwidth of the test signal. The list of available bandwidth IDs depends on the sensor and can be found in the sensor's user manual under the command SENSE:BWIDth:VIDEo :

```
lErr = rsnrpz_bandwidth_setBw( ulUSBSession, 1, 0 );
```

The number of video points and the RF level range is set with the function `rsnrpz_stat_confScale`. Using 500 video points usually represents a good compromise between USB transfer speed and resolution. The CCDF data's transfer time increases with the number of video points. The first level parameter specifies the lower end of the level scale in dBm. The second level parameter sets the total range of the level scale in dB:

```
lErr = rsnrpz_stat_confScale( ulUSBSession, 1,
                              rfRefLevelLow,
                              rfRange,
                              iVideoPoints );
```

The measurement time sets the overall capture time for one CCDF measurement. In combination with the bandwidth setting, it defines how many signal samples are to be evaluated:

```
lErr = rsnrpz_stat_setTime( ulUSBSession, 1,
                            dMeasTime );
```

Example:

The R&S®NRP-Z81 uses a sample clock of 80 MHz when set to its highest bandwidth. This corresponds to a sample time of 12.5 ns. Setting the measurement time to 125 ms will, therefore, evaluate ten million samples.

If very large numbers of samples need to be evaluated over a longer period of time, the averaging filter can be activated. In the CCDF mode, this filter does not average the CCDF data; instead, it extends the observation period:

```
lErr = rsnrpz_scope_setAverageCount( ulUSBSession, 1,
                                     iAverageCount );

lErr = rsnrpz_scope_setAverageEnabled( ulUSBSession, 1,
                                       true );

lErr = rsnrpz_scope_setAverageTerminalControl(
      ulUSBSession, 1,
      RSNRPZ_TERMINAL_CONTROL_REPEAT );

lErr = rsnrpz_avg_reset( ulUSBSession, 1 );
```

The `rsnrpz_chan_initiate` function call starts the measurement cycle and returns immediately.

```
lErr = rsnrpz_chan_initiate( ulUSBSession, 1 );
```

Before any data can be read from the sensor, the measurement status must be polled repeatedly. This polling must be implemented in such a way that the CPU becomes available to the driver thread periodically:

```
ViBoolean bMeasCompleted;

do {
    lErr = rsnrpz_chan_isMeasurementComplete(
        ulUSBSession, 1,
        &bMeasCompleted );

    Sleep( ... );
} while( !bMeasCompleted && lErr==0 );
```

After the measurement has completed, the average power, as well as the CCDF data, can be read. The average power value is returned in linear units and includes any offset configured using `rsnrpz_corr_setOffset`. The CCDF data points are returned in linear scale between 0 and 1:

```
ViReal64 pdMeas[iCCDFPoints];
ViInt32  iReadCount;
ViReal64 dMeasAvPow;

lErr = rsnrpz_meass_fetchMeasurement(
    ulUSBSession, 1,
    &dMeasAvPow );

lErr = rsnrpz_meass_fetchBufferMeasurement(
    ulUSBSession, 1,
    iCCDFPoints,
    pdMeas,
    &iReadCount );
```

9 Timeslot

This chapter describes how to implement a timeslot measurement for a repeating signal that provides a stable trigger condition. In this mode, the average power of a definable number of successive timeslots within a frame structure with equal spacing is measured. The timeslot structure is mainly defined by the timeslot width and the timeslot count. The measurement result is an array with the same number of elements as timeslots. Each array element represents the average power in a particular timeslot. When the averaging function is activated, and an averaging factor of two or more has been chosen, measurements are performed with chopper stabilization to obtain more accurate results with reduced noise and zero offset. Time intervals that are to be excluded from the measurement can be set at the beginning, in the middle and at the end of each timeslot.

First, the sensor's operation mode needs to be set. This step is only required initially:

```
lErr = rsnrpz_chan_mode( ulUSBSession, 1,
    RSNRPZ_SENSOR_MODE_TIMESLOT );
```

Second, the carrier frequency must be set. Setting the carrier frequency is required for precise power measurements:

```
lErr = rsnrpz_chan_setCorrectionFrequency( ulUSBSession, 1,
    dCarrierHz );
```

In many cases, the power sensor is not directly connected to the DUT, and compensation must be made for additional cable loss. The following two functions enable and set the level offset:

```
lErr = rsnrpz_corr_setOffset( ulUSBSession, 1,  
                             dOffsetdB );
```

```
lErr = rsnrpz_corr_setOffsetEnabled( ulUSBSession, 1,  
                                     bEnOffset );
```

The timeslot structure is mainly defined by the number of timeslots and the timeslot width in seconds. Both parameters are set using the following function:

```
lErr = rsnrpz_tslot_configureTimeSlot( ulUSBSession, 1,  
                                       iSlotCount,  
                                       dWidth );
```

The exclude time is the amount of time at the beginning or at the end of each time slot that gets ignored during the measurement. Setting an exclude time is often required with pulsed signals for which the rising and falling edge should not be included in the measurement:

```
lErr = rsnrpz_timing_configureExclude( ulUSBSession, 1,  
                                       dExclStart,  
                                       dExclStop );
```

Configuring the trigger condition is crucial for all timeslot measurements. The following lines configure the trigger system to internal triggering on a positive slope. The hysteresis should be set to a small value (e.g. 1 dB) to allow for stable triggering. The dropout time can be set optionally and requires the signal to fall below the trigger threshold for the defined period of time before the trigger system rearms again. Please note that the trigger level is set in linear units.

```
lErr = rsnrpz_trigger_setSource( ulUSBSession, 1,  
                                 RSNRPZ_TRIGGER_SOURCE_INTERNAL );
```

```
lErr = rsnrpz_trigger_setSlope( ulUSBSession, 1,  
                                RSNRPZ_SLOPE_POSITIVE );
```

```
lErr = rsnrpz_trigger_setDropoutTime( ulUSBSession, 1,  
                                       dDropoutTime );
```

```
lErr = rsnrpz_trigger_setHysteresis( ulUSBSession, 1,  
                                     1 );
```

```
lErr = rsnrpz_trigger_setLevel( ulUSBSession, 1,  
                                dTrigLevelW );
```

The trigger delay time is an important parameter in the timeslot measurement mode. It allows precise alignment of the timeslot structure with the trigger point. Use this parameter to compensate for trigger delays caused by the length of the trigger cable. The delay time can be negative or positive:

```
lErr = rsnrpz_trigger_setDelay( ulUSBSession, 1,
                               dDelayTime );
```

Setting an averaging filter is, in most cases, desired when trace data is to be measured. Averaging reduces the noise dramatically and therefore increases the dynamic range:

```
lErr = rsnrpz_scope_setAverageCount( ulUSBSession, 1,
                                      iAverageCount );
```

```
lErr = rsnrpz_scope_setAverageEnabled( ulUSBSession, 1,
                                       true );
```

The averaging filter can be operated in either repeating or moving mode. In repeating mode, the filter content is cleared at the beginning of the measurement cycle. Once the filter is entirely filled, the measurement terminates, and the result can be read:

```
lErr = rsnrpz_scope_setAverageTerminalControl(
        ulUSBSession, 1,
        RSNRPZ_TERMINAL_CONTROL_REPEAT );
```

The `rsnrpz_chan_initiate` function call starts the measurement cycle and returns immediately.

```
lErr = rsnrpz_chan_initiate( ulUSBSession, 1);
```

Before any data can be read from the sensor, the measurement status must be polled repeatedly. This polling must be implemented in such a way that the CPU becomes available to the driver thread periodically:

```
ViBoolean bMeasCompleted;

do {
    lErr = rsnrpz_chan_isMeasurementComplete(
        ulUSBSession, 1,
        &bMeasCompleted );

    Sleep( ... );

} while( !bMeasCompleted && lErr==0 );
```

After the measurement has completed, the data array can be read using the `rsnrpz_meass_fetchBufferMeasurement` function. The values that are returned are in linear units and include any offset configured using `rsnrpz_corr_setOffset`:

```
ViReal64 pdMeasAv[iSlotCount];
ViInt32  iReadCount;

lErr = rsnrpz_meass_fetchBufferMeasurement(
        ulUSBSession, 1,
        iSlotCount,
        pdMeasAv,
        &iReadCount );
```

The settings from the above example return the average power of the individual timeslots. The return data is provided in linear units. However, many applications require power values on a logarithmic scale. The conversion can be done using the following equation:

$$P_{log} = 10 \cdot \log_{10}(P_{lin}) \text{ dBm} + 30 \text{ dBm}$$

Care must be taken if signal portions close to the noise floor must be converted. Depending on the zero reference point for the internal analog-to-digital converter, negative power readings may occur.

This is normal behavior, and in most cases, it is possible to simply use the linear power reading's absolute value for the `log10` function.

In very rare cases, a power value that is exactly zero may arise. Zero cannot be converted into the logarithmic scale and must, therefore, be replaced by another value.

9.1.1 Peak Timeslot Data

Wideband sensors, such as the R&S®NRP-Z81, provide multiple timeslot data representations. The AVERAGE data representation is always sent and cannot be deselected. Alternatively, the sensor can be switched to auxiliary mode, and it can send two additional representations, such as the RANDOM and MAXIMUM timeslot data:

```
lErr = rsnrpz_chan_setAuxiliary( ulUSBSession, 1,  
                                RSNRPZ_AUX_RNDMAX );
```

When auxiliary data is enabled, the timeslot data must be read from the driver cache using the `rsnrpz_meas_fetchBufferMeasurementAux` function. In a way similar to the method used with the regular fetch function, all data is provided in linear units and contains the level offset:

```
ViReal64 pdMeasAv[iSlotCount];  
ViReal64 pdMeasRnd[iSlotCount];  
ViReal64 pdMeasPeak[iSlotCount];  
ViInt32  iReadCount;  
  
lErr = rsnrpz_meas_fetchBufferMeasurementAux(  
        ulUSBSession, 1,  
        0,  
        iSlotCount,  
        pdMeasAv, pdMeasRnd, pdMeasPeak,  
        &iReadCount  
    );
```


10 Peculiarities

10.1 Reading Parameter Limits

The driver cache holds limits for all parameters that can be set inside the sensor. The sensor updates these limits automatically and may change them anytime. The following code demonstrates how these limits can be obtained from the sensor. First, the parameter of interest must be read from the sensor. This ensures that the most recent limits are prevalent in the driver cache. Next, the limits are queried by providing the SCPI command that is associated with the parameter. A list of SCPI commands can be found in the sensor's user manual [3]. The example below shows how the range for the trigger level can be determined:

```
double dDummy;
lErr = rsnrpz_trigger_getLevel( ulUSBSession, 1, &dDummy );

double dCurrent, dMin, dMax;
lErr = rsnrpz_chan_getCacheRange( ulUSBSession, 1,
    "TRIG:LEV",
    &dCurrent, &dMin, &dMax );
```

10.2 Reading the Trigger and Measure State

The driver cache always hold the sensor's current trigger and measuring status. The following code lines explain how these states can be read from the driver:

```
unsigned short bState;
lErr = rsnrpz_status_checkCondition( ulUSBSession,
    RSNRPZ_STATCLASS_O_MEAS,
    RSNRPZ_SENSOR_01,
    &bState);
printf( "MEAS state: %d", bState );

lErr = rsnrpz_status_checkCondition( ulUSBSession,
    RSNRPZ_STATCLASS_O_TRIGGER,
    RSNRPZ_SENSOR_01,
    &bState);
printf( "TRIGGER state: %d", bState );
```

10.3 Checking for Level Over-Range

An error condition that could occur during measurements is the level over-range indication. If such a condition occurs while a measurement is running, the call to `rsnrpz_chan_isMeasurementComplete` will generate an error code. In the error handler, the device error can be further examined by calling `rsnrpz_error_query`. The OVERLOAD error indicates that the sensor's detector is about to be destroyed by the RF signal. In contrast, the OVERRANGE error indicates that the A/D-converter limit has been reached, and the result might be wrong.

```
char szErrMsg[256];
rsnrpz_error_query( ulUSBSession, &lErr, szErrMsg );

// See NRP manual - Device dependent error messages
// Translation of codes in NrpErrorOccurredCallback()
// NRP_ERROR_DEVICE_OVERLOAD = 51
// NRP_ERROR_DEVICE_OVERRANGE = 52

if( lErr == NRP_ERROR_DEVICE_OVERLOAD ||
    lErr == NRP_ERROR_DEVICE_OVERRANGE )
    ...
```

10.4 Questionable USB Data Transfer

An error condition that could arise on older USB hardware is a disrupted USB data transfer. The following code tests to see if this error has occurred. If this has occurred, the result should not be used, and the measurement should be re-initiated.

```
char szErrMsg[256];
rsnrpz_error_query( ulUSBSession, &lErr, szErrMsg );

if( lErr==NRP_ERROR_DEVICE_RESULT_QUESTIONABLE ) {
    rsnrpz_chan_abort( ulUSBSession, 1 );
    ...
}
```

11 Literature

- [1] R&S®NRP Toolkit, Power Viewer Plus User Manual
- [2] VXI PnP Driver Documentation
<http://www.rohde-schwarz.com/downloads/drivers/nrpz.html>
- [3] R&S®NRP-Z Power Sensor Operating Manual
- [4] Application Note 1GP61 – MATLAB® Toolkit for R&S®NRP-Z Sensors

12 Ordering Information

Designation	Type	Order No.
Universal Power Sensors		
200 pW to 200 mW, 10 MHz to 8 GHz (cable length 1.6 m)	R&S®NRP-Z11	1138.3004.02
200 pW to 200 mW, 10 MHz to 8 GHz (cable length 0.4 m)	R&S®NRP-Z11	1138.3004.04
200 pW to 200 mW, 10 MHz to 18 GHz	R&S®NRP-Z21	1137.6000.02
2 nW to 2 W, 10 MHz to 18 GHz	R&S®NRP-Z22	1137.7506.02
20 nW to 15 W, 10 MHz to 18 GHz	R&S®NRP-Z23	1137.8002.02
60 nW to 30 W, 10 MHz to 18 GHz	R&S®NRP-Z24	1137.8502.02
200 pW to 200 mW, 10 MHz to 33 GHz	R&S®NRP-Z31	1169.2400.02
Two-Path Diode Sensors		
1 nW to 100 mW, 10 MHz to 8 GHz	R&S®NRP-Z211	1417.0409.02
1 nW to 100 mW, 10 MHz to 18 GHz	R&S®NRP-Z221	1417.0309.02
Wideband Power Sensors		
1 nW to 100 mW, 50 MHz to 18 GHz	R&S®NRP-Z81	1137.9009.02
1 nW to 100 mW, 50 MHz to 40 GHz	R&S®NRP-Z85	1411.7501.02
1 nW to 100 mW, 50 MHz to 40 GHz	R&S®NRP-Z86	1417.0109.40
Thermal Power Sensors		
1 µW to 100 mW, DC to 18 GHz	R&S®NRP-Z51	1138.0005.02
300 nW to 100 mW, DC to 33 GHz	R&S®NRP-Z52	1138.0005.02
300 nW to 100 mW, DC to 40 GHz	R&S®NRP-Z55	1138.2008.03
300 nW to 100 mW, DC to 44 GHz	R&S®NRP-Z55	1138.2008.04
300 nW to 100 mW, DC to 50 GHz	R&S®NRP-Z56	1171.8201.02
300 nW to 100 mW, DC to 67 GHz	R&S®NRP-Z57	1171.8401.02
300 nW to 100 mW, DC to 110 GHz	R&S®NRP-Z58	1173.7031.02
Average Power Sensors		
200 pW to 200 mW, 9 kHz to 6 GHz	R&S®NRP-Z91	1168.8004.02
2 nW to 2 W, 9 kHz to 6 GHz	R&S®NRP-Z92	1171.7005.02
2 nW to 2 W, 9 kHz to 6 GHz (incl. R&S®NRP-Z4 USB adapter cable var. 04 m, 0.5 m)	R&S®NRP-Z92	1171.7005.42
Level Control Sensors		
200 pW to 100 mW, 9 kHz to 6 GHz	R&S®NRP-Z98	1170.8508.02
200 pW to 100 mW, 10 MHz to 18 GHz	R&S®NRP-Z28	1170.8008.02
Power Sensor Modules		
4 µW to 400 mW, DC to 18 GHz	R&S®NRP-Z27	1169.4102.02

4 μ W to 400 mW, DC to 26.5 GHz	R&S®NRP-Z37	1169.3206.02
Recommended Extras		
Sensor Extension Cable to 3 m	R&S®NRP-Z2	1146.6750.03
Sensor Extension Cable to 5 m	R&S®NRP-Z2	1146.6750.05
Sensor Extension Cable to 10 m	R&S®NRP-Z2	1146.6750.10
Panel-Mount Extension Cable to 5 m	R&S®NRP-Z2	1146.6750.15
USB Adapter (active), 2 m	R&S®NRP-Z3	1146.7005.02
USB Adapter (passive), 1.5 m	R&S®NRP-Z4	1146.8001.02
USB Adapter (passive), 0.5 m	R&S®NRP-Z4	1146.8001.02
USB Adapter (passive) for panel-mount, 1 m	R&S®NRP-Z4	1146.8001.11
USB Sensor Hub	R&S®NRP-Z5	1146.7740.02

About Rohde & Schwarz

Rohde & Schwarz is an independent group of companies specializing in electronics. It is a leading supplier of solutions in the fields of test and measurement, broadcasting, radiomonitoring and radiolocation, as well as secure communications. Established more than 75 years ago, Rohde & Schwarz has a global presence and a dedicated service network in over 70 countries. Company headquarters are in Munich, Germany.

Environmental commitment

- Energy-efficient products
- Continuous improvement in environmental sustainability
- ISO 14001-certified environmental management system



Regional contact

Europe, Africa, Middle East

+49 89 4129 137 74

customersupport@rohde-schwarz.com

North America

1-888-TEST-RSA (1-888-837-8772)

customer.support@rsa.rohde-schwarz.com

Latin America

+1-410-910-7988

customersupport.la@rohde-schwarz.com

Asia/Pacific

+65 65 13 04 88

customersupport.asia@rohde-schwarz.com

This application note and the supplied programs may only be used subject to the conditions of use set forth in the download area of the Rohde & Schwarz website.

R&S® is a registered trademark of Rohde & Schwarz GmbH & Co. KG; Trade names are trademarks of the owners.

Rohde & Schwarz GmbH & Co. KG

Mühlendorfstraße 15 | D - 81671 München

Phone + 49 89 4129 - 0 | Fax + 49 89 4129 - 13777

www.rohde-schwarz.com