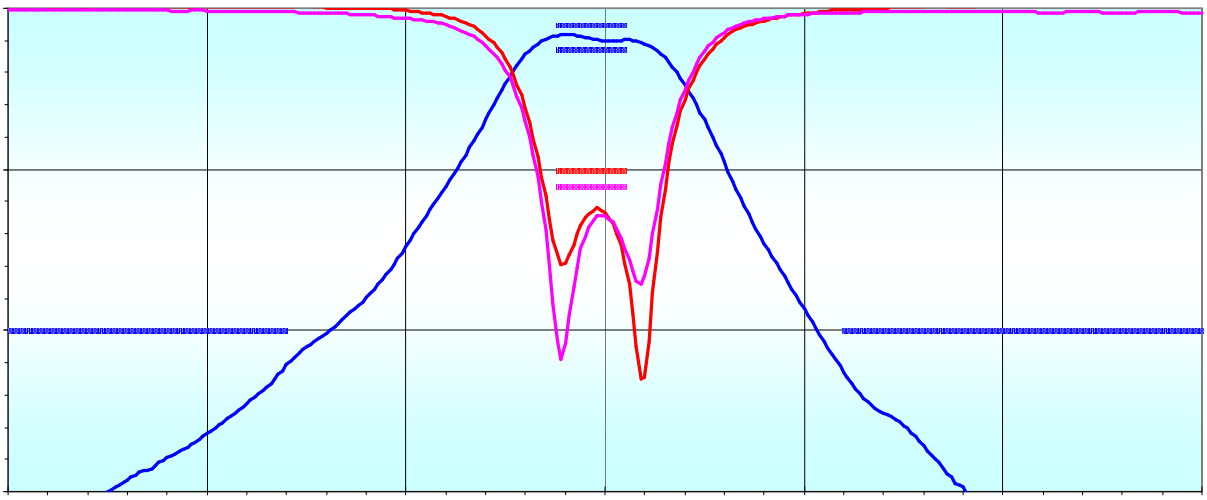
MegiQ VNA

Application Programming Interface



Contents

Revisions

Version	Date	MiQVNA version	Description
1	27-9-2015	1.4.007	First API Release
2	26-7-2016	1.5.008	New function ConnectSerial()

MegiQ VNA Application Programming Interface

1 Introduction

The MegiQ VNA API offers third party programmers access to all VNA functionality to configure and perform measurements and access the resulting data for further processing.

The API can be used for programming custom test programs that are typically used in production test environments.

2 Legal stuff

3 API interface

3.1 Programming model

The MegiQ VNA API is based on an ActiveX (COM) component that is part of the MiQVNA program.
In order to use this API, version 1.4.007 or higher of the MiQVNA program is required, unless noted otherwise.

The main component of the API is the mvnaVNAMain class. This class gives access to functions and other classes to control the VNA.

Most classes are not-creatable, i.e. one can only get an instance by calling a VNA function that creates the class. Only the data classes that contain actual measurement data are creatable because this allows further (arithmetic) manipulation of the data.

Several classes are of a transient nature and the user application should request a reference to a class only when needed to access up-to-date information, and release it afterwards. This manual describes which classes are static and can be referenced permanently.

Some classes can raise events that can be handled by the user application. These events are for signaling changes in status such as connection and measurement status, availability of new data etc.

3.2 Programming languages

The API interface is accessed through the Windows COM class specification that can be used in several programming languages. The interface has been used in Visual Basic 6 (VB6), VBA for Office (Word, Excel etc), VB.Net, LabWindows/CVI and C#.

The driver works in its own thread and some languages need specific measurements to handle multithreading.

The API is native to VB6 and VBA so the access to objects and other elements is straightforward and the code does not need to concern itself with multithreading.

C# can import the typelib and expose the functionality. However, it will convert or prefix some names to avoid C# conflicts with other names. C# also needs to use 'Delegates' and other measures to handle the multithreading.

LabWindows/CVI has a typelib import utility that creates header and code files in plain C. The access of COM objects in plain C is a bit laborious. Also the multithreading needs to be handled.

The API development kit provides full-featured examples of VB6 and C-Sharp code for accessing all API functions.

This manual provides examples mostly in VB6 because that is the simplest interface without too many language distractions. Other languages are very similar.

3.3 Naming conventions

All classes and enumerations are preceded by the '**mvna**' prefix to avoid interference with the namespace of other COM components. Events have the '**evt**' prefix to avoid naming conflicts.

This document often omits the prefix when it is not essential for the context.

3.4 Starting the driver

The VNA API can be accessed by creating an instance of the class mvnaVNAMain. This will start an instance of the MiQVNA program, with its user screen minimized. If the autoconnect flag in MiQVNA is on, the program will automatically connect to the first connected VNA it finds in the list of ports. The application can monitor the VNA Status to determine when a full connection is established.

3.5 Terminating the driver

The VNA API is terminated by releasing all VNA objects, finishing with the mvnaVNAMain object. This will disconnect the software from the VNA and close MiQVNA.

3.6 Hardware connection

The MiQVNA driver has an option to open a connection to the VNA automatically (**AutoConnect**). If this option is active the test application does not need to handle connection and disconnection, it just needs to monitor SystemStatus events. This option is on by default. The driver will not Auto Connect when it is disconnected by software or with the Connect button in the MiQVNA driver, until the VNA is unplugged and plugged.

Whether **AutoConnect** is on or off the driver can always use the Connect and Disconnect functions.

If the VNA is connected to the USB port without having a power supply it will stay in bootloader mode until power is connected. The application can check for this in software.

4 VNA Software structure

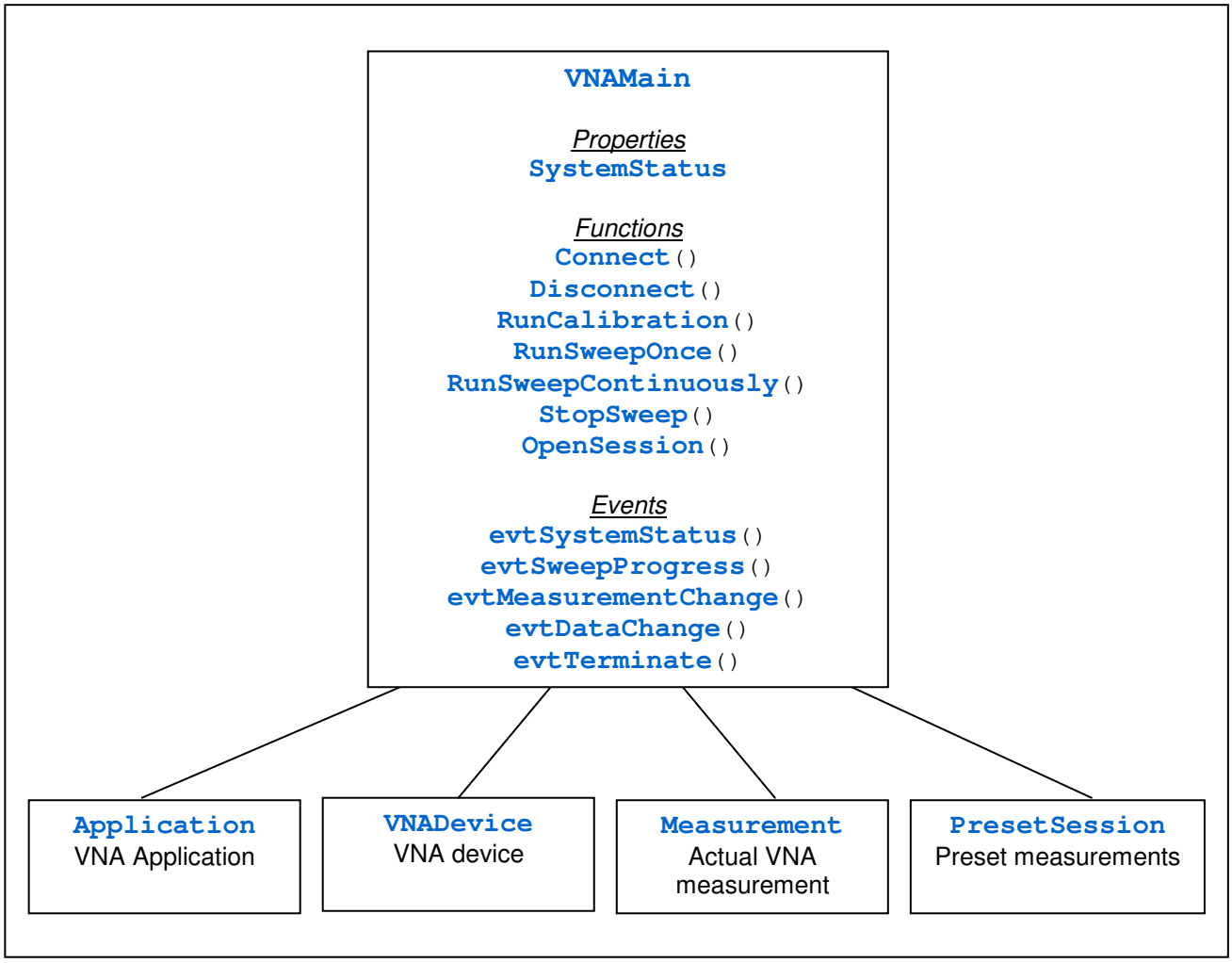
4.1 VNAMain

The main object is the mvnaVNAMain object that provides access to the VNA.

This class provides functions to control the connection between the VNA and the PC and functions to control sweeping actions.

It contains these classes:

- **Application**: provides version and path information of the MiQVNA application and control of its screen.
- **VNADevice**: allows control of some VNA hardware functions.
- **Measurement**: contains the current VNA measurement: configuration, parameters and data.
- **PresetSession**: provides access to the pre-configured preset measurements



For example, an application would start with creating a **VNAMain** class. It can use the **SystemStatus** property and event to verify or wait for proper VNA connection. It uses **Measurement** to setup a measurement if necessary and then can use **RunSweepOnce** to initiate a sweep. At the arrival of the **DataChange** event it can access the data in **Measurement** to check against a template.

4.2 Measurement

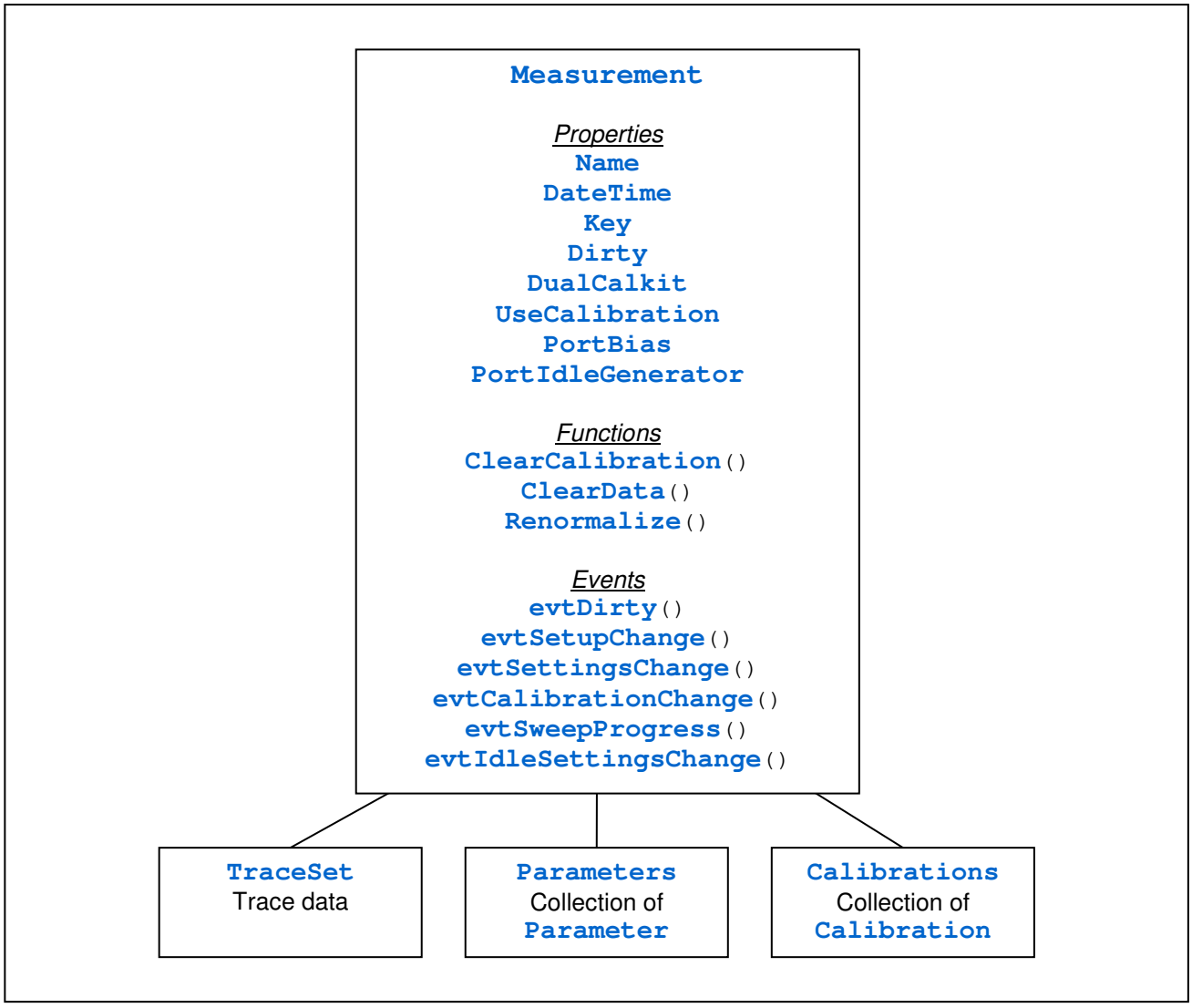
The Measurement class contains a whole measurement:

- Name and time stamp
- Port/Cycle configuration
- Static and Sweep parameters
- Calibration cycle enumeration
- Calibration data
- Measurement data

It has functions to clear data and renormalize the data.

The classes within a Measurement are:

- **TraceSet**: contains all calibration and measurement data
- **Parameters**: all parameters used during the measurement and during idle state.
- **Calibrations**: enumerates the calibration cycles for the measurement



4.3 TraceSet

The TraceSet contains the measurement data with the following hierarchy:

- **Parameters**: the static and sweep parameters
- **Traces**: a set of traces. For simple sweeps there is only 1 trace but for nested sweep there will be a trace for each of the lowest level sweeps.
- **Trace**: each lowest level sweep is a Trace and contains a set of Parameters used for this trace and a set of TraceChannels for each of the measurements S11, S22, S21 etc
- **TraceChannel**: there is a TraceChannel for each of the S-parameters. It can contain several IQData data sets for the calibration (CalOpen etc) and an IQData data set with the normalized measurement data: Return data or Through data.
- **IQData**: a set of measurement points
- **IQ**: a measurement point in IQ format

One would access a particular data point value like this:

```
double QVal = TraceSet.Traces(1).Channels("S11").DataSet("Return").QValue(25);
```

4.4 IQData

IQData contains an array with (complex) calibration, measurement or derived points. The data is generally stored as S parameter values although derived values can have another dimension.

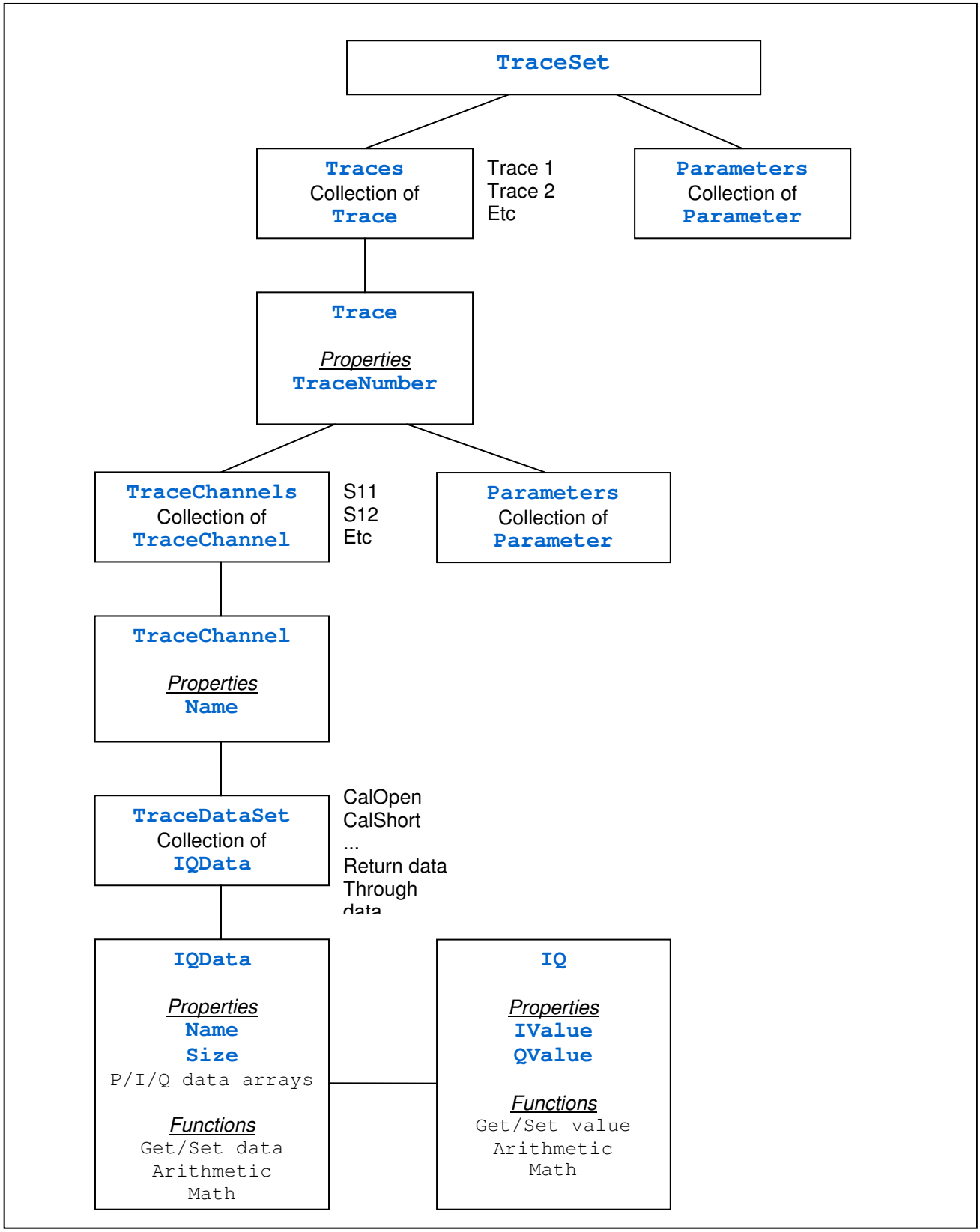
Along with the data points IQData also stores an array of 'parameter' values that correspond to the sweep parameter values of the (lowest) sweep parameter. Depending on the sweep configuration, these are usually frequency values but can be other parameter values like power or bias voltages.

Although IQData provides IQ elements for each measurement point, it actually stores its data in arrays of I-values and Q-values for computational efficiency. It is recommended to use the array functions of IQData to access the data points.

IQData also provides a number of arithmetic and math functions that operate on the arrays directly. This allows cascaded numerical operations with great efficiency.

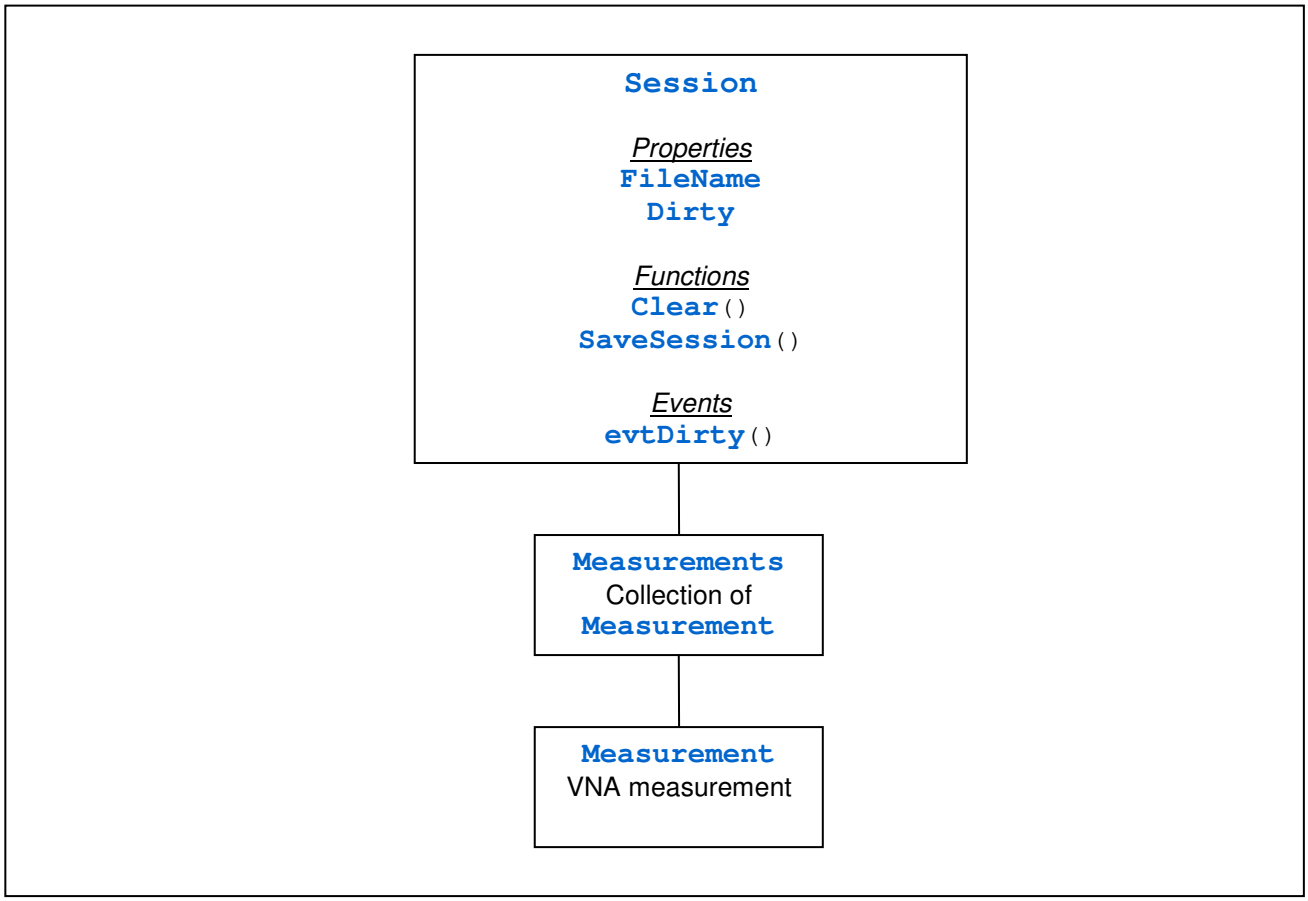
For example one can calculate the round trip gain S12 * S21 for the whole array like this:

```
mvnaIQData S21, S12, SRT;

S12 = TraceSet.Traces(1).Channels("S12").DataSet("Through");
S21 = TraceSet.Traces(1).Channels("S21").DataSet("Through");

SRT = S12.Multiply(S21);

double AmpDbVals[];
double PhaseVals[];

SRT.GetAmpPhaseValuesDbDegrees(AmpdBVals, PhaseVals);
```


TraceSet
Traces
Collection of
Trace
Trace 1
Trace 2
Etc
Parameters
Collection of
Parameter
Trace
Properties
TraceNumber
TraceChannels
Collection of
TraceChannel
S11
S12
Etc
Parameters
Collection of
Parameter
TraceChannel
Properties
Name
TraceDataSet
Collection of
IQData
CalOpen
CalShort
...
Return data
Through
data
IQData
Properties
Name
Size
P/I/Q data arrays
Functions
Get/Set data
Arithmetic
Math
IQ
Properties
IValue
QValue
Functions
Get/Set value
Arithmetic
Math

4.5 Session

A Session is a collection of measurements that can be stored in a session file. Measurements can be added or deleted through the Measurements collection.



New measurements can not be created directly. The application can take an existing measurement (either from a Session or the current VNA Measurement) and save it as a new measurement in a (new) session.

VNAMain contains a Preset Session. That is a session that is loaded at startup of MiQVNA and contains some preconfigured measurements, without calibration or measurement data.

4.6 Working with Sessions

Sessions can be used to configure and manage measurements:

- The application developer can use the MiQVNA program to create a template session with one or more preconfigured measurements. For a fixed test setup MiQVNA can be used to create the calibration data and store this in the measurement.
- The test program can open this Session file and load the measurement into the active VNA measurement of VNAMain.
- When a measurement is completed the test program can use the measured data to check against a template, or do other processing.
- If desired for logging, the completed measurement can be added to a new Session. A number of measurements can be saved into one Session file. The test program has the option to save only the measurement data or all data including the calibration data.

However, with the calibration data the Session file could to get quite large, so the measurements could be saved in batches.

The following pseudo code shows how to load a (calibrated) configuration Session, check the measured S21 gain and log failed items:

At startup:

```
// Initialize the driver and load a test measurement
VnaMain = new mvnaVNAMain;
ConfigSession = VnaMain.OpenSession("ConfigSessionFile.vns");
VnaMain.Measurement = ConfigSession.ItemByName("GainTest");

// Create an empty log session
LogSession = VnaMain.OpenSession("");
```

When measurement data arrives:

```
// Get the data from the TraceSet
S21 = VnaMain.TraceSet.Traces(1).Channels("S21").DataSet("Through");

// Get Gain dB values and check the values
S21.GetAmpValuesDb (AmpValues);
for (i = 0; i < AmpValues.Size; i++)
{
      if (AmpValues[i] < MinimumGain)
            Fail = true;
}

if (Fail)
{
      LogSession.Measurements.AddItem(VnaMain.Measurement, "Fail 001",
                Now);
}
```

When the program terminates or there are many measurements in the Session:

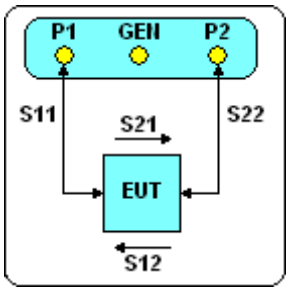
```
LogSession.SaveSession("Failed Items 001", false, mvnaVDO_DATA);

LogSession = VnaMain.OpenSession("");     // Get empty new log session
```

4.7 Measurement configuration and Channels

The measurement configuration is the specification of the (S) parameters that need to be measured on the ports. (S11, 2-Port etc). This configuration determines the measurement cycles, calibrations needed and the resulting sets of data called **Channels**.

For example, a 2-Port setup results in the Channels S11, S22, S21 and S12:

- S11 and S22 contain reflection / impedance data at the port, as well as Open/Short/Load calibrations.
- S21 and S12 contain gain / loss data between the ports, as well as Through and Isolation calibrations.



A 3-port measurement also yields S13 and S23, and using some of the external bridge configurations can yield S33, S31 and S32.

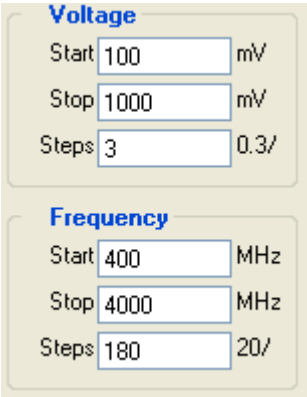
The API does not provide a way to setup the configuration of a measurement. This can only be done manually in MiQVNA and saved as a measurement.

4.8 Sweep structure and Traces

It is possible to configure all kinds of parametric sweeps that can involve any of the available parameters (frequency, power, bias etc).

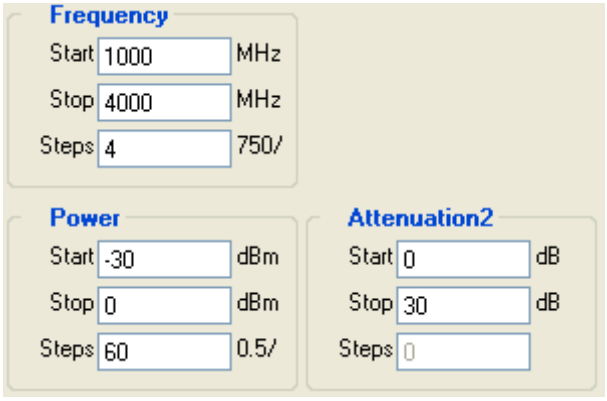
The most common sweep is the simple Frequency Sweep. The frequency is swept with all other parameters static. This results in a single `Trace` in a `TraceSet`.

MiQVNA makes it possible to configure nested and combined sweeps using multiple parameters.



For example, one can use the sweep on the left to do Frequency sweeps at different Bias Voltages. The lowest parameter Frequency is the sweep parameter and above that Voltage is a parametric parameter. Since Voltage has 3 steps we get 4 `Traces` each with 181 samples of Frequency sweep samples (and calibrations).

This sweep can be used to measure the 1dB-compression point of an amplifier. Here the Power is swept with 60 steps, along with the input Attenuator on port 2 to allow for the gain of the amplifier. This sweep is repeated over 5 Frequencies and thus yields 5 `Traces`.



The **TraceSet** holds the collection of the **Traces** of this measurement, along with all **Parameter** settings including the static parameters. The application can use the Parameter.**IsSweep** property to determine whether a parameter is static or is swept.

Each **Trace** also contains a **Parameters** set used for that Trace. In this parameter set only the lowest parameter(s) in the sweep is (are) variable (**IsSweep** = True). The parameters higher in the sweep definition are static at the current step for the Trace. This value is stored in the **CurrentValue** of the parameter.

The **Parameters** collections do not provide a means to determine the hierarchy of a sweep.

The API does not provide a way to setup the configuration of a sweep. This can only be done manually in MiQVNA and saved as a measurement.

4.9 Parameters

The settings of a **Measurement** are controlled by a number of **Parameters**. A Parameter contains information about a (physical) property of the measurement. It has a **CurrentValue** that is the actual value.

Any **Parameter** can be involved in a **Sweep**. It contains sweep settings (**StartValue**, **StopValue**, **Steps**) to control the sweep.

If a **Parameter** is static then the **CurrentValue** applies during measurements. If it is a **Sweep** Parameter then the sweep settings apply during measurements.

The **CurrentValue** also determines the value during Idle state of the VNA.

A **Sweep** reaches the **StopValue** after the number of **Steps** are performed. This means that a sweep parameter yields one more sample or trace then the number of steps.

These are the **Parameters** in the Parameters collection:

• VNA_FREQUENCY	VNA measurement frequency	Hz
• GEN_POWER	Generator Power	dBm
• DET_ATTENUATION1	Detector 1 input Attenuation	dB
• DET_ATTENUATION2	Detector 2 input Attenuation	dB
• BIAS_VOLTAGE	Bias Voltage	V
• BIAS_CURRENT	Bias Current	A

The parameter values are whole physical SI units or dBs. The **Dimension** field provides the a string that can be used as a suffix for display. It is up the application to scale this to human versions (MHz, mA).

4.10 Calibrations

If a **Measurement** involves user **Calibration** the calibration values are provided in the **DataSet** of a **Channel**. Calibrations are device independent S-parameters as they are measured at the VNA port.

There are two byproducts of a user calibration: **CalSource** and **CalSink**. These are the impedances at the two Calibration Planes that connect to the EUT. They are used in a 2-Port 12-term normalization.

4.11 Event handling

Some classes raise events to keep the application updated about the status. VNAMain has events for change in system state and measurement setup, and when new measurement data is available. A simple application will only need these events.

Measurement provides some more events for settings changes. They may be used for more elaborate screen updates.

Session provides a Dirty event that indicates if a session contains unsaved information.

4.12 Object life cycle

Most objects can not be created by the test application, they must be acquired from the API.

The main object, **VNAMain**, is created by the application and (typically) exists during the life of the application. It's constituent objects are static objects that stay alive while VNAMain exists.

When a new **Measurement** is assigned to **VNAMain**, the contents of this measurement is copied into the current measurement. This means that a reference to **VNAMain.Measurement** will stay valid. However, the **Parameters** collection and **Traceset** will be replaced during this operation.

The **Traceset** is a copy of the actual measurement data, and a new copy of the measurement data is created each time a reference to Traceset is requested.

This means that the application should not retain a reference to a **Traceset** longer than necessary. When new measurement data is available a new Traceset reference should be obtained for further data processing.

This also means that if multiple **TraceSet** operations are done in series, the application should only request the TraceSet once. Otherwise, a complete dataset copy is made each time the traceset is accessed. For example, the second of these two examples is more efficient:

Example 1:

```
Dim S11 As mvnaIQData
Dim S22 As mvnaIQData

' Will make two TraceSet copies of the measurement data:
Set S12 = Measurement.TraceSet.Traces(1).Channels("S12").DataSet("Through")
Set S21 = Measurement.TraceSet.Traces(1).Channels("S21").DataSet("Through")
```

Example 2:

```
Dim TS As mvnaTraceSet
Dim S11 As mvnaIQData
Dim S22 As mvnaIQData

' Will make only one TraceSet copy:
Set TS = Measurement.TraceSet
Set S12 = TS.Traces(1).Channels("S12").DataSet("Through")
Set S21 = TS.Traces(1).Channels("S21").DataSet("Through")
```

5 Application Programming

The VNA API provides the means to make a custom application (Test Application) for specific VNA tests or repetitive tasks and verify the measurement results automatically. This chapter gives a brief outline how a VNA application can be structured.

Although the MiQVNA program is used as an interface between the test application and the VNA hardware, the user interface of MiQVNA is not intended to be actively involved in the test application. The test application is supposed to provide its own user interface. This is typically a simple user screen, possibly with a result and template graph and often with a Pass-Fail decision. The test application can choose to make the MiQVNA screen visible but this is mainly intended for development of the application. However, the API provides the functionality to control almost all VNA measurement features.

The API makes it possible to configure the parameters (frequency, power, bias settings etc) of the current measurement, but it is not possible to change the port and sweep configuration. The idea is that the test application loads a preconfigured measurement from a session file and use this during operation. The preconfigured measurement can be loaded from the Preset session or it can be a specific measurement that was previously setup by the application developer and stored in a custom session. The application can change the measurement configuration at any time to perform different tests.

The API does not provide any graphical functions for plotting data. This is left to the test application if desired.

5.1 A simple VNA test program

The following code shows a minimal application program that will load a preconfigured measurement, perform a sweep and check the result.

The developer has already made a session file (using MiQVNA) with an S11 measurement with the proper Open-Short-Load calibration for a particular test fixture.

The test will pass if all Return Loss values are smaller than -10 dB.

The usual error checking was omitted for clarity of the program structure.

```
Private WithEvents clsVNA As mvnaVNAMain
Private clsLogSession As mvnaSession
Private lngSerialNumber As Long

Private Sub InitApplication()
      Dim SS As mvnaSession

      ' Create a VNA object and hide the screen
      Set clsVNA = new mvnaVNAMain
      clsVNA.Application.ShowState = mvnaWSS_Hidden

      ' Open a session file with a configured test measurement
      Set SS = clsVNA.OpenSession("MyPath\MySession.vns")
      Set clsVNA.Measurement = SS.Measurements.Item("MyS11Measurement")

      ' Create a log session
      Set clsLogSession = clsVNA.OpenSession(“”)
End Sub

Private Sub ExitApplication()
      ' Save the log session
      Call clsLogSession.SaveSession("MyPath\MyLogSession.vns", False,
      mvnaVDO_DATA)
      Set clsLogSession = Nothing

      ' Delete the VNA object
      Set clsVNA = Nothing
End Sub

Private Sub clsVNA_evtSystemStatus(ByVal Status As mvnaVNAStatus)
      ' Event handler: System / Connection status has changed
      lblStatus.Caption = Format(Status)

      ' Enable or disable the test button
      if (Status = mvnaVST_Idle) Then
            cmdRunTest.Enabled = True
      Else
            cmdRunTest.Enabled = False
      End If
End Sub
```

```
Private Sub clsVNA_evtDataChange(ByVal TraceNr As Long, ByVal NrTraces As Long)
      ' Event handler: New data has arrived

      If (TraceNr = NrTraces) Then

            ' Check new measurement data
            If (CheckTestData() = True) Then

                  ' Log Pass data
                  Call clsLogSession.Measurements.AddItem(clsVNA.Measurement,
                     Format(lngSerialNumber), Now)

                  lngSerialNumber = lngSerialNumber + 1

                  Call MsgBox("Pass")

            Else

                  Call MsgBox("Fail")

            End If

      End If
End Sub

Private Sub cmdRunTest_Click()
      ' Event handler: RunTest button pressed
      Call clsVNA.RunSweepOnce()
End Sub

Private Function CheckTestData() As Boolean
      ' Check test data for Return Loss > -10dB
      Dim TS As mvnaTraceSet
      Dim S11 As mvnaIQData
      Dim RLVals() As Double
      Dim i As Long

      ' Get S11 data
      Set TS = clsVNA.TraceSet
      Set S11 = TS.Traces(1).Channels("S11").DataSet("Return")

      ' Get amplitude in dB
      Call S11.GetAmpValuesDb(RLVals)

      ' Check the Return Loss
      For i = 0 To S11.Size - 1
            If (RLVals(i) > -10.0) Then
                  CheckTestData = False
                  Exit Function
            End if
      Next i

      CheckTestData = True
End Function
```

6 API Reference

6.1 Class mvnaVNAMain

This is the main driver class.

Properties

6.1.1.1 SystemStatus

```
C#:   virtual mvnaVNAStatus SystemStatus { get; }
VB:   Property Get SystemStatus() As mvnaVNASTATUS
```

Get the status of the VNA system. SystemStatus can be one of the following:

```
mvnaVST_Disconnected
mvnaVST_Initializing
mvnaVST_Idle
mvnaVST_Calibrating
mvnaVST_Sweeping
```

6.1.1.2 Application

```
C#:   virtual mvnaApplication Application { get; }
VB:   Property Get Application() As mvnaApplication
```

Returns a reference to the Application class with info about the VNA driver application MiQVNA.

6.1.1.3 VNADevice

```
C#:   virtual mvnaVNADevice VNADevice { get; }
VB:   Property Get VNADevice() As mvnaVNADevice
```

Returns a reference to the VNADevice class to control the VNA hardware. The information is valid when the system is not in **mvnaVST_Disconnected** state.

6.1.1.4 Measurement

```
C#:   virtual mvnaMeasurement get_Measurement()
      virtual void set_Measurement(ref mvnaMeasurement value)
VB:   Property Get Measurement() As mvnaMeasurement
      Property Set Measurement(Val As mvnaMeasurement)
```

Returns a reference to the current, active VNA measurement.

This reference is static during the lifetime of the driver, so the application can keep a reference to catch measurement events. However, it's constituent classes are volatile and can change when sweeping or when loading a new measurement.

6.1.1.5 PresetSession

```
C#:   virtual mvnaSession PresetSession { get; }
VB:   Property Get PresetSession() as mvnaSession
```

Returns a reference to the VNA Preset **Session** with preconfigured measurements.

The Preset Session should only be read and not manipulated.

Functions

6.1.1.6 Connect

```
C#:   virtual void Connect()
VB:   Sub Connect()
```

Connect to the VNA hardware. This can only be called when the system is in **mvnaVST_Disconnected** state. If the hardware is present the driver will generate **evtSystemStatus** events with the states **mvnaVST_Initializing** and then **mvnaVST_Idle**.

If there is no VNA hardware connected an error -2147221502 will be raised.

If a VNA is connected without its power supply it will be in bootloader mode. In that case only the system will stay in the **mvnaVST_Initializing** state until the power is connected or **Disconnect** is called. This can be checked with the VNAMain.Application.**IsBootloader** flag.

If the 'Auto Connect' option in MiQVNA is on then the driver will automatically connect and initialize.

6.1.1.7 ConnectSerial

```
C#:   virtual void ConnectSerial(ref string SerialNumber)
VB:   Sub ConnectSerial(SerialNumber As String)
```

This function is available from MiQVNA version 1.5.008. It allows connecting to a VNA with a specific serial number. This makes it possible to use multiple VNAs connected to one computer, running multiple instances of a client program. All leading zeros must be supplied. If the serial number is an empty string then this function will behave the same as Connect().

6.1.1.8 Disconnect

```
C#:   virtual void Disconnect()
VB:   Sub Disconnect()
```

Disconnect the driver from the VNA.

The 'Auto Connect' option in MiQVNA will not cause the driver to reconnect unless the VNA is unplugged and plugged again.

6.1.1.9 RunCalibration

```
C#:   void virtual void RunCalibration(int CalIndex)
VB:   Sub RunCalibration(ByVal CalIndex As Long)
```

Runs a calibration sweep.

CalIndex sets the desired calibration type (Open, Short, Load etc). The application can enumerate the necessary calibrations and indexes with the Calibrations collection of the current measurement.

The **SystemStatus** event or property can be used to monitor the calibrating state. No **evtDataChange** event is raised during calibration.

6.1.1.10 RunSweepOnce

```
C#:   virtual void RunSweepOnce()
VB:   Sub RunSweepOnce()
```

Runs a measurement sweep.

The **SystemStatus** event or property can be used to monitor the sweeping state. During sweeping the **evtDataChange** event will report when a sweep is finished and sweep data can be processed.

6.1.1.11 RunSweepContinuously

```
C#:   virtual void RunSweepContinuously()
VB:   Sub RunSweepContinuously()
```

Starts continuous sweeping.

The **SystemStatus** event or property can be used to monitor the sweeping state. During sweeping the **evtDataChange** event will report when a sweep is finished and sweep data can be processed.

6.1.1.12 StopSweep

```
C#:   virtual void StopSweep()
VB:   Sub StopSweep()
```

Stops the continuous sweeping.

The SystemStatus event or property can be used to determine when the last sweep is finished and the driver returns to **mvnaVST_Idle** state.

6.1.1.13 OpenSession

```
C#:   virtual mvnaSession OpenSession(ref string FileName)
VB:   Function OpenSession(FileName As String) As mvnaSession
```

Returns a Session object.

FileName designates a session file and must include the .vns extension. If **FileName** is an empty string then a new and empty Session is returned.

Events

6.1.1.14 evtSystemStatus

```
C#:   virtual event __mvnaVNAMain_evtSystemStatusEventHandler evtSystemStatus
VB:   Event evtSystemStatus(ByVal Status As mvnaVNAStatus)
```

Reports the status of the VNA system. Status can be one of the following:

```
mvnaVST_Disconnected
mvnaVST_Initializing
mvnaVST_Idle
mvnaVST_Calibrating
mvnaVST_Sweeping
```

6.1.1.15 evtSweepProgress

```
C#:   virtual event __mvnaVNAMain_evtSweepProgressEventHandler evtSweepProgress
VB:   Event evtSweepProgress(ByVal PointsReceived As Long, ByVal PointsTotal As
      Long)
```

Reports the progress during a sweep.

The values **PointsReceived** and **PointsTotal** can be used to control a progress bar. If **PointsTotal** is zero the progressbar can be hidden. A progressbar can be desirable during long sweeps.

6.1.1.16 evtMeasurementChange

```
C#:   virtual event __mvnaVNAMain_evtMeasurementChangeEventHandler
      evtMeasurementChange
VB:   Event evtMeasurementChange()
```

Reports when the Current Measurement has been replaced, either by the application or directly by a user in MiQVNA.

This event can be used to update settings on the screen.

6.1.1.17 evtDataChange

```
virtual event __mvnaVNAMain_evtDataChangeEventHandler evtDataChange
VB:   Event evtDataChange(ByVal TraceNr As Long, ByVal NrTraces As Long)
```

Reports when a trace is complete and data can be processed.

For a multi-trace (parametric) sweep **NrTraces** reports the total number of traces in the **TraceSet** and **TraceNr** reports the number of the trace that has new data.

Though for simple traces both parameters will always be 1 it is recommended to use code like:

```
if (TraceNr == NrTraces)
{
      process data here;
}
```

6.1.1.18 evtTerminate

```
C#:   virtual event __mvnaVNAMain_evtTerminateEventHandler evtTerminate
VB:   Event evtTerminate()
```

Reports when the driver is terminated.

Although this event will also fire when the application terminates the driver, the main purpose is to detect that the driver is terminated by other means: the exit button in MiQVNA or by Windows task manager. The application can use it to clean up and exit if desired. The event may not fire when MiQVNA is terminated improperly.

6.2 Class mvnaApplication

This class contains information about the MiQVNA driver application.

Properties

6.2.1.1 Name

```
C#:   virtual string Name { get; }
VB:   Property Get Name() As String
```

Returns the driver name (MiQVNA).

6.2.1.2 ExeName

```
C#:   virtual string ExeName { get; }
VB:   Property Get ExeName() As String
```

Returns the name of the driver executable (MiQVNA.);

6.2.1.3 Path

```
C#:   virtual string Path { get; }
VB:   Property Get Path() As String
```

Returns the full path, without file name, to the driver executable.

6.2.1.4 Version

```
C#:   virtual string Version { get; }
VB:   Property Get Version() As String
```

Returns the driver version.

6.2.1.5 ShowState

```
C#:   virtual mvnaWindowShowState ShowState { set; get; }
VB:   Property Get ShowState() As mvnaWindowShowState
      Property Let ShowState(ByVal Val As mvnaWindowShowState)
```

Controls the screen state of MiQVNA.

WindowState can be one of the following:

```
mvnaWSS_Hidden
mvnaWSS_Minimized
mvnaWSS_Normal
mvnaWSS_Maximized
```

In Hidden mode MiQVNA is not visible and there is no button on the taskbar. In Minimized mode there will be a taskbar button but the screen is minimized.

6.3 Class mvnaVNADevice

Provides info about the VNA hardware. Also allows control over the port LEDs.

Properties

6.3.1.1 VNAInfoString

```
C#:   virtual string VNAInfoString { get; }
VB:   Property Get VNAInfoString() As String
```

Returns a summary string with VNA device info, formatted with tabs and returns.

6.3.1.2 IsBootloader

```
C#:   virtual bool IsBootloader { get; }
VB:   Property Get IsBootloader() As Boolean
```

Returns True when the device is in bootloader mode.

The bootloader is active when the VNA is connected to the USB port without power supply. There is (almost) no API functionality in this mode. The devices switches back to full VNA functionality when the power is reapplied.

6.3.1.3 Vendor

```
C#:   virtual string Vendor { get; }
VB:   Property Get Vendor() As String
```

Returns the Vendor name.

6.3.1.4 Product

```
C#:   virtual string Product { get; }
VB:   Property Get Product() As String
```

Returns the Product name (VNA21 series).

6.3.1.5 DeviceType

```
C#:   virtual string DeviceType { get; }
VB:   Property Get DeviceType() As String
```

Returns the device type (VNA).

6.3.1.6 SerialNumber

```
C#:   virtual string SerialNumber { get; }
VB:   Property Get SerialNumber() As String
```

Returns the serial number of 8 decimal digits.

6.3.1.7 HardwareVersion

```
C#:   virtual string HardwareVersion { get; }
VB:   Property Get HardwareVersion() As String
```

Returns the hardware version (201)

6.3.1.8 FirmwareVersion

```
C#:   virtual string FirmwareVersion { get; }
VB:   Property Get FirmwareVersion() As String
```

Returns the version of the firmware of the VNA.

In bootloader mode this will be the version of the bootloader.

6.3.1.9 CPLDVersion

```
C#:   virtual string CpldVersion { get; }
VB:   Property Get CPLDVersion() As String
```

Returns the version of the CPLD code.

This is not available in bootloader mode.

6.3.1.10 Ports

```
C#:   virtual mvnaVNAPort Ports { get; }
VB:   Property Get Ports() As mvnaVNAPort

Returns the available ports.

Ports is a bitmap of these values:

      mvnaVNP_None  = 0
      mvnaVNP_Port1 = 1
      mvnaVNP_Port2 = 2
      mvnaVNP_Port3 = 4
```

6.3.1.11 BiasOption

```
C#:   virtual bool BiasOption { get; }
VB:   Property Get BiasOption() As Boolean

Returns True if the bias option is installed.
```

6.3.1.12 DeviceID

```
C#:   virtual string DeviceID { get; }
VB:   Property Get DeviceID() As String

Returns the device ID.

The device ID is a unique code that identifies the VNA.
```

6.3.1.13 OverrideLedState

```
C#:   virtual void OverrideLedState(mvnaVNAPort Port, mvnaColor Color,
      mvnaLedState State)
VB:   Sub OverrideLedState(ByVal Port As mvnaVNAPort, ByVal Color As mvnaColor,
      ByVal State As mvnaLedState)
```

This function is used to control the port LEDs at the VNA front panel.

Port specifies the target LED(s). Multiple LEDs can be addressed at once by OR-ing bits.

Color is a bitmap that specifies the color of the LED(s). Bits can be OR-ed:

```
mvnaCOL_None  = 0
mvnaCOL_Red   = 1
mvnaCOL_Green = 2
mvnaCOL_Blue  = 4
mvnaCOL_All   = 7
```

State specifies the override state:

```
mvnaLED_Off
mvnaLED_On
mvnaLED_Blink
mvnaLED_BlinkFast
```

The override is released when all colors of a LED are in the off-state. It is therefore not possible to force a LED off.

6.4 Class mvnaSession

Session manages a collection of Measurements.

Properties

6.4.1.1 FileName

```
C#:   virtual string FileName { get; }
VB:   Property Get FileName() As String
```

Returns the file name from which the session is read, or to which the session is saved.

6.4.1.2 Dirty

```
C#:   virtual bool Dirty { get; }
VB:   Property Get Dirty() As Boolean
```

Returns the Dirty flag that indicates when the session has been changed since it was last read or saved.

6.4.1.3 Measurements

```
C#:   virtual mvnaMeasurements Measurements { get; }
VB:   Property Get Measurements() As mvnaMeasurements
```

Returns the collection of **Measurements**

Functions

6.4.1.4 Clear

```
C#:   virtual void Clear()
VB:   Public Sub Clear()
```

Clears the Measurements collection and file name.

6.4.1.5 SaveSession

```
C#:   virtual bool SaveSession(ref string FileName, bool SaveAs,
      mvnaVNADataOptions SaveOptions)
VB:   Function SaveSession(FileName As String, ByVal SaveAs As Boolean, ByVal
      SaveOptions As mvnaVNADataOptions) As Boolean
```

Save the session to a file. It will overwrite an existing file.

FileName specifies a full path with filename and extension. The normal extension is '.vns' but another extension is possible.

SaveAs is not used and must be False.

SaveOptions is a bitmask that specifies which elements are to be saved:

```
mvnaVDO_NONE        = 0
mvnaVDO_CALIBRATION = 2
mvnaVDO_DATA        = 4
```

When **mvnaVDO_NONE** is used the configuration is saved without calibration or measurement data.

Events

6.4.1.6 evtDirty

```
C#:   virtual event __mvnaSession_evtDirtyEventHandler evtDirty
VB:   Event evtDirty(ByVal Flag As Boolean)
```

Reports that the session has changed and may need to be saved.

6.5 Class mvnaMeasurements

Collection of mvnaMeasurement.

Each item will be assigned a key by the class. This key can be used to identify an item. Keys can be obtained by enumerating the collection with a for loop.

Properties

6.5.1.1 Item

```
C#:   virtual mvnaMeasurement get_Item(ref object IndexKey)
VB:   Property Get Item(IndexKey As Variant) As mvnaMeasurement
```

Returns a **Measurement**.

IndexKey can contain an index (base 1) or a key string.

6.5.1.2 ItemByName

```
C#:   virtual mvnaMeasurement get_ItemByName(ref string Name)
VB:   Property Get ItemByName(Name As String) As mvnaMeasurement
```

Returns a **Measurement** by Name.

Name is not case sensitive.

Note: Measurements in a Session can have **duplicate** names. When this property is used only the first of duplicate names is returned.

6.5.1.3 Count

```
C#:   virtual int Count { get; }
VB:   Property Get Count() As Long
```

Returns the number of **Measurements** in the collection.

Functions

6.5.1.4 AddItem

```
C#:   virtual mvnaMeasurement AddItem(ref mvnaMeasurement Measurement, ref
      string Name, System.DateTime DateTime)
VB:   Function AddItem(Measurement As mvnaMeasurement, Name As String, ByVal
      DateTime As Date) As mvnaMeasurement
```

Add a **Measurement** to the collection.

Measurement is the item to add. **Name** is the name of the measurement. It is allowed to add multiple items with the same name. **DateTime** is a timestamp.

6.5.1.5 Remove

```
C#:   virtual void Remove(ref object IndexKey)
VB:   Sub Remove(IndexKey As Variant)
```

Removes a **Measurement**.

IndexKey can contain an index (base 1) or a key string.

6.6 Class mvnaMeasurement

The **Measurement** class contains functions and properties to control the VNA, measurement flow and access to the measurement data.

Properties

6.6.1.1 Name

```
C#:   virtual string get_Name()
      virtual void set_Name(ref string value)
VB:   Property Get Name() As String
      Property Let Name(Val As String)
```

Set or Get the **Name** of the measurement. It is up to the application to assign a name.

6.6.1.2 DateTime

```
C#:   virtual System.DateTime DateTime { set; get; }
VB:   Property Get DateTime() As Date
      Property Let DateTime(ByVal Val As Date)
```

Set or Get the timestamp of the measurement. It is up to the application to assign a timestamp.

6.6.1.3 Key

```
C#:   virtual string Key { get; }
VB:   Property Get Key() As String
```

Returns the **Key** in the Measurements collection for this Measurement.

Since the collection allows duplicate names, the key can be used to identify a measurement. The key is assigned by the driver and is not necessarily the same after saving and retrieving a measurement.

6.6.1.4 Dirty

```
C#:   virtual bool Dirty { get; }
VB:   Property Get Dirty() As Boolean
```

Get the **Dirty** flag.

The Dirty flag indicates whether the measurement has changed since it was retrieved from file or since it was last saved.

6.6.1.5 DualCalkit

```
C#:   virtual bool DualCalkit { set; get; }
VB:   Property Get DualCalkit() As Boolean
      Property Let DualCalkit(ByVal Val As Boolean)
```

Set or Get a flag that indicates whether a dual calibration kit is used.

Using a dual cal kit allows several calibration cycles to be combined for a (much) shorter calibration procedure. Changing this flag will cause the Calibrations collection to change (and cause a CalibrationChange event). However, the calibration data, if present, will be preserved.

6.6.1.6 UseCalibration

```
C#:   virtual bool UseCalibration { set; get; }
```

```
VB:   Property Get UseCalibration() As Boolean
      Property Let UseCalibration(ByVal Val As Boolean)
```

Get or Set a flag to apply the user calibration to the measurement or not.

If this flag is True, the user calibration will be applied to the measurement. If the flag is False, only the port calibration is used. Changing this flag will not cause a renormalization of the measurement. The Renormalize() function is used to cause a renormalization.

6.6.1.7 PortBias

```
C#:   virtual mvnaVNABiasControl get_PortBias(mvnaVNAPort Port)
      virtual void set_PortBias(mvnaVNAPort Port, mvnaVNABiasControl value)
VB:   Property Get PortBias(ByVal Port As mvnaVNAPort) As mvnaVNABiasControl
      Property Let PortBias(ByVal Port As mvnaVNAPort, ByVal Bias As
      mvnaVNABiasControl)
```

Get or Set a Bias switch control flag for each VNA port.

This function controls the routing of the Bias generator output to each of the ports.

For the Set function **Port** is a bitmask allowing multiple ports, but it must designate a single port for the Get function.

Bias is one of the following:

```
mvnaVBC_Off         The port has a high impedance.
mvnaVBC_Ground      The port is routed to ground for DC.
mvnaVBC_On          The port is connected to the bias generator.
```

When the bias switch is Off, the port has a DC resistance to ground of 100 kOhm.

When the switch is set to Ground, the port has a DC resistance of about 20 Ohm to ground. This allows for a return path of the Bias current through a DUT.

When the switch is set to On, there is a DC impedance of about 20 Ohm to the Bias generator.

The Bias generator itself (voltage and current) are controlled through the Bias Voltage and Bias Current parameters in the Parameters collection. They can be part of a sweep.

6.6.1.8 PortIdleGenerator

```
C#:   virtual mvnaVNAPort PortIdleGenerator { set; get; }
VB:   Property Get PortIdleGenerator() As mvnaVNAPort
      Property Let PortIdleGenerator (ByVal Val As mvnaVNAPort)
```

Get or Set the routing of the VNA signal generator during idle state.

Port must designate a single VNA port.

Although this parameter is not part of the measurement configuration, this parameter is stored with the measurement anyway.

6.6.1.9 TraceSet

```
C#:   virtual mvnaTraceSet TraceSet { get; }
VB:   Property Get TraceSet() As mvnaTraceSet
```

Returns a **TraceSet** object that contains the measurement data.

The **TraceSet** is a copy of the actual measurement data. Please refer to the section about the TraceSet object life cycle for more information.

6.6.1.10 Parameters

```
C#:   virtual mvnaParameters Parameters { get; }
VB:   Property Get Parameters() As mvnaParameters
```

Returns a **Parameters** collection that contains all measurement parameters.

6.6.1.11 Calibrations

```
C#:   virtual mvnaCalibrations Calibrations { get; }
VB:   Property Get Calibrations() As mvnaCalibrations
```

Returns a **Calibrations** collection that contains an enumeration of the calibration cycles for the measurement.

Functions

6.6.1.12 ClearCalibration

```
C#:   virtual void ClearCalibration()
VB:   Sub ClearCalibration()
```

Clears the calibration data of the measurement.

The (normalized) measurement data will remain intact but can not be renormalized, unless a new calibration is performed again.

6.6.1.13 ClearData

```
C#:   virtual void ClearData()
VB:   Sub ClearData()
```

Clears the measurement data of the measurement.

The calibration data will remain intact.

6.6.1.14 Renormalize

```
C#:   virtual void Renormalize()
VB:   Sub Renormalize()
```

Renormalizes the measurement data with the calibration set.

It is possible to run a measurement without calibration, then perform a calibration and use Renormalize() to apply the new calibration set on the measurement data.

Events

6.6.1.15 evtDirty

```
C#:   virtual event __mvnaMeasurement_evtDirtyEventHandler evtDirty
VB:   Event evtDirty(ByVal Flag As Boolean)
```

Reports that the measurement has changed and may need to be saved.

6.6.1.16 evtSetupChange

```
C#:   virtual event __mvnaMeasurement_evtSetupChangeEventHandler evtSetupChange
VB:   Event evtSetupChange()
```

Reports that the setup (port or sweep configuration) has changed.

This event can be used to update the user interface of the test application.

6.6.1.17 evtSettingsChange

```
C#:   virtual event __mvnaMeasurement_evtSettingsChangeEventHandler
      evtSettingsChange
VB:   Event evtSettingsChange()
```

Reports that one of the parameter settings has been changed.

This event can be used to update the user interface of the test application.

6.6.1.18 evtCalibrationChange

```
C#:   virtual event __mvnaMeasurement_evtCalibrationChangeEventHandler
      evtCalibrationChange
VB:   Event evtCalibrationChange(ByVal CalibrationNr As Long, ByVal
      NrCalibrations As Long)
```

Reports that list of calibrations has been changed.

This event can be used to update a list of calibrations in the user interface of the test application.

6.6.1.19 evtSweepProgress

```
C#:   virtual event __mvnaMeasurement_evtSweepProgressEventHandler
      evtSweepProgress
VB:   Event evtSweepProgress(ByVal PointsReceived As Long, ByVal PointsTotal As
      Long)
```

Reports the progress during a sweep.

The values **PointsReceived** and **PointsTotal** can be used to control a progress bar. If **PointsTotal** is zero the progressbar can be hidden. A progressbar can be desirable during long sweeps.

6.6.1.20 evtIdleSettingsChange

```
C#:   virtual event __mvnaMeasurement_evtIdleSettingsChangeEventHandler
      evtIdleSettingsChange
VB:   Event evtIdleSettingsChange()
```

Reports that an idle setting has changed.

Currently this event only applies to the PortIdleGenerator setting.

6.7 Class mvnaTraceSet

The TraceSet class contains all data pertaining to a full sweep.

Example to retrieve a single Q-value:

```
C#:   double QVal1 =
      TraceSet.Traces[1].Channels["S11"].DataSet["Return"].get_QValue(1);

VB:   Dim QVal1 As Double
      QVal1 = TraceSet.Traces(1).Channels("S11").DataSet("Return").QValue(1)
```

Properties

6.7.1.1 Traces

```
C#:   virtual mvnaTraces Traces { get; }
VB:   Property Get Traces() As mvnaTraces
```

Get the collection of **Traces** in this traceset

6.7.1.2 Parameters

```
C#:   virtual mvnaParameters Parameters { get; }
VB:   Property Get Parameters() As mvnaParameters
```

Get the collection of **Parameters** that are used for measuring this traceset.

6.8 Class mvnaTraces

Collection of a full TraceSet.

Properties

6.8.1.1 Item

```
C#:   virtual mvnaTrace this[ref object IndexKey] { get; }
VB:   Property Get Item(IndexKey As Variant) As mvnaTrace
```

Returns a Trace.

IndexKey can contain an index (base 1). Keys are not used.

6.8.1.2 Count

```
C#:   virtual int Count { get; }
VB:   Property Get Count() As Long
```

Returns the number of Traces in the collection.

6.9 Class mvnaTrace

Contains the data of one Trace.

Properties

6.9.1.1 TraceNumber

```
C#:   virtual int TraceNumber { get; }
VB:   Property Get TraceNumber() As Long
```

Returns the number of the **Trace** in the **TraceSet**.

6.9.1.2 Channels

```
C#:   virtual mvnaTraceChannels Channels { get; }
VB:   Property Get Channels() As mvnaTraceChannels
```

Returns the **Channels** collection with channel data.

6.9.1.3 Parameters

```
C#:   virtual mvnaParameters Parameters { get; }
VB:   Property Get Parameters() As mvnaParameters
```

Returns the collection of **Parameters** that are used for this trace.

6.10 Class mvnaTraceChannels

Collection of all **Channels** in the **Trace**.

These are the possible **TraceChannels** in this collection:

- S11
- S22
- S33
- S21
- S12
- S31
- S32

It is possible that future versions contain more names.

Properties

6.10.1.1 Item

```
C#:   virtual mvnaTraceChannel this[ref object IndexKey] { get; }
VB:   Property Get Item(IndexKey As Variant) As mvnaTraceChannel
```

Returns a **TraceChannel**.

IndexKey can contain an index (base 1), or a name.

6.10.1.2 Count

```
C#:   virtual int Count { get; }
VB:   Property Get Count() As Long
```

Returns the number of **TraceChannels** in the collection.

6.11 Class mvnaTraceChannel

Contains the DataSet for a TraceChannel.

Properties

6.11.1.1 Name

```
C#:   virtual string Name { get; }
VB:   Property Get Name() As String
```

Returns the number of the **Name** of the **Trace**.

6.11.1.2 DataSet

```
C#:   virtual mvnaTraceDataSet DataSet { get; }
VB:   Property Get DataSet() As mvnaTraceDataSet
```

Returns the **DataSet** that contains the calibration and measurement data.

6.12 Class mvnaTraceDataSet

Collection of calibration and measurement **IQData**.

If the data is **Return** data the IQData names can be:

- CalOpen
- CalShort
- CalLoad
- CalSource
- Return

CalSource is not a physical calibration but results from the Open / Short / Load calibration. It represents the source impedance at the Calibration Plane of the Source port, as it drives the EUT.

For **Through** data the IQData names can be:

- CalThrough
- CalIsolation
- CalSink
- Through

CalSink is measured during the Through calibration. It represents the impedance at the Calibration Plane of the Sink port, as it loads the EUT.

CalSource and CalSink are factored in during a 12-term normalization.

It is possible that future versions contain more names.

Properties

6.12.1.1 Item

```
C#:   virtual mvnaIQData this[ref object IndexKey] { get; }
VB:   Property Get Item(IndexKey As Variant) As mvnaIQData
```

Returns an **IQData** object.

IndexKey can contain an index (base 1), or a name of an **IQData** object.

6.12.1.2 Count

```
C#:   virtual int Count { get; }
VB:   Property Get Count() As Long
```

Returns the number of **IQData** objects in the collection.

6.13 Class mvnaParameters

Collection of **Parameters**.

The Parameters in this collection can be:

- VNA_FREQUENCY
- GEN_POWER
- DET_ATTENUATION1
- DET_ATTENUATION2
- BIAS_VOLTAGE
- BIAS_CURRENT

It is possible that future versions contain more names.

Parameter values are doubles that specify physical SI units or dBs. The Parameter will provide the dimension.

Properties

6.13.1.1 Item

```
C#:   virtual mvnaParameter this[ref object IndexKey] { get; }
VB:   Property Get Item(IndexKey As Variant) As mvnaParameter
```

Returns a **Parameter**.

IndexKey can contain an index (base 1), or a parameter name.

6.13.1.2 Count

```
C#:   virtual int Count { get; }
VB:   Property Get Count() As Long
```

Returns the number of **Parameters** in the collection.

6.14 Class mvnaParameter

Properties

6.14.1.1 Name

```
C#:   virtual string Name { get; }
VB:   Property Get Name() As String
```

Returns the name of the parameter.

6.14.1.2 Dimension

```
C#:   virtual string Dimension { get; }
VB:   Property Get Dimension() As String

Returns the dimension of the parameter.
```

Dimension can be ‘dB’, ‘V’ or another physical unit.

6.14.1.3 MinValue

```
C#:   virtual double MinValue { get; }
VB:   Property Get MinValue() As Double
```

Returns the minimum allowable value of the parameter.

6.14.1.4 MaxValue

```
C#:   virtual double MaxValue { get; }
VB:   Property Get MaxValue() As Double
```

Returns the maximum allowable value of the parameter.

6.14.1.5 CurrentValue

```
C#:   virtual double CurrentValue { set; get; }
VB:   Property Get CurrentValue() As Double
      Property Let CurrentValue(ByVal Val As Double)
```

Get or Set the current parameter value.

CurrentValue is used during the idle state of the VNA, or as a fixed value if the parameter is not used as a sweep parameter.

6.14.1.6 StartValue

```
C#:   virtual double StartValue { set; get; }
VB:   Property Get StartValue() As Double
      Property Let StartValue(ByVal Val As Double)
```

Get or Set the start value of the parameter of a sweep.

StartValue, **StopValue** and **Steps** are used when the parameter is a sweep parameter.

6.14.1.7 StopValue

```
C#:   virtual double StopValue { set; get; }
VB:   Property Get StopValue() As Double
      Property Let StopValue(ByVal Val As Double)
```

Get or Set the stop value of the parameter of a sweep.

StartValue, **StopValue** and **Steps** are used when the parameter is a sweep parameter.

6.14.1.8 Steps

```
C#:   virtual int Steps { set; get; }
VB:   Property Get Steps() As Long
      Property Let Steps(ByVal Val As Long)
```

Get or Set the number of steps of a sweep.

StartValue, **StopValue** and **Steps** are used when the parameter is a sweep parameter.

6.14.1.9 IsSweep

```
C#:   virtual bool IsSweep { get; }
VB:   Property Get IsSweep() As Boolean
```

Returns True if the parameter is used as a sweep parameter.

6.14.1.10 SweepType

```
C#:   virtual mvnaSweepType get_SweepType()
      virtual void set_SweepType(ref mvnaSweepType value)
VB:   Property Get SweepType() As mvnaSweepType
      Property Let SweepType(Val As mvnaSweepType)
```

Get or Set the type of sweep. mvnaSweepType can be one of the following:

mvnaSWT_Lin	Linear sweep of **Steps** + 1 points from **StartValue** to **StopValue**
mvnaSWT_List	Sweep **Steps** + 1 points from a list of sweep values contained in **SweepValueList**.

6.14.1.11 SweepValueList

```
C#:   System.Array get_SweepValueList()
      void set_SweepValueList(ref System.Array value)
VB:   Property Get SweepValueList() As Double()
      Property Let SweepValueList(Val() As Double)
```

Get or Set the list of sweep values for sweep type **mvnaSWT_List**. The variable **Steps** must be manually set to the number of points in the list – 1.

Functions

6.14.1.12 Update

```
C#:   virtual void Update()
VB:   Public Sub Update()
```

This function must be called after one or more of the parameter values have been changed, to put the changes into effect.

6.15 Class mvnaCalibrations

This is a collection of **Calibrations**.

Properties

6.15.1.1 Item

```
C#:   virtual mvnaCalibration get_Item(ref object IndexKey)
VB:   Property Get Item(IndexKey As Variant) As mvnaCalibration
```

Returns a **Calibration**.

IndexKey can contain an index (base 1), or a calibration name.

6.15.1.2 Count

```
C#:   virtual int Count { get; }
VB:   Property Get Count() As Long
```

Returns the number of **Calibrations** in the collection.

6.16 Class mvnaCalibration

This is a class for enumeration of the calibration cycles. It does not contain actual calibration data. The calibration data is stored in a **DataSet**.

Properties

6.16.1.1 Index

```
C#:   virtual int Index { get; }
VB:   Property Get Index() As Long
```

Returns the index number (base 1) in the **Calibrations** collection.

6.16.1.2 Caption

```
C#:   virtual string Caption { get; }
VB:   Property Get Caption() As String
```

Returns a caption for use in a listbox or such.

6.16.1.3 Complete

```
C#:   virtual bool Complete { get; }
VB:   Property Get Complete() As Boolean
```

Returns True if the calibration is complete, i.e. that all calibration cycles have been performed. This can be used for completion indication in the user interface of the application.

6.17 Class mvnaIQData

After digging deeply into the TraceSet, we get to the actual calibration or measurement data, stored in the **IQData** object.

The **IQData** object holds measured or calculated I/Q samples of a sweep, including 'parametric' values (usually frequencies). The samples are stored as (float) doubles in I/Q format. The data is stored in three arrays: **IValues**, **QValues** and **PValues**.

From an 'objective' view it would be logical to store these values in objects (the **mvnaIQ** class) but this makes data manipulation extremely slow. An interface to use values as IQ objects is included to allow some handy manipulation of individual samples, but it is strongly recommended to perform data processing and handling on the **IQData** object as a whole or by iterating through the **IValues** and **QValues** arrays that can be obtained from this class.

There are several categories of functions:

- Get or set individual I, Q and P values
- Get or Set the whole I, Q and P arrays
- Get or Set IQ objects and array
- Fill the I and Q arrays with constant values
- Convert between I/Q and Amp/Phase values, linear or dB
- Return a new IQData object with constant (zero, one, two) values
- Unitary arithmetic (negate, root, smoothing)
- Binary arithmetic (add, subtract, multiply, divide)
- Convert S parameters to and from physical values (Z, SWR, RL, FL)

The arithmetic functions operate on the whole data arrays at once and they return a new IQData object with the result. This makes it possible to cascade the functions and perform complex operations very efficiently. The arithmetic functions don't copy the PData to the target, that has to be done manually if desired. All arrays involved must have the same size.

For example, the following code calculates the impedances from the S11 samples:

```
' Calculate Z = -(S11+1) * Zo / (S11-1)

Dim Zo As mvnaIQData
Dim Z As mvnaIQData

' Create array Zo with 50 Ohm values
Set Zo = New mvnaIQData
Call Zo.CreateIQData(0, 0, 50.0, 0, S11.Size)

' Calculate Z
Set Z = S11.Add(S11.One).Neg.Multiply(Zo).Divide(S11.Subtract(S11.One))

' PValues are not copied in arithmetic functions, copy them here
Call Z.SetPValues(S11.PValues)
```

This code is implemented in IQData.GetZ().

6.17.1 Object functions

```
C#:   virtual string get_Name()
      virtual void set_Name(ref string value)
VB:   Property Get Name() As String
      Property Let Name(Val As String)

C#:   virtual int Size { set; get; }
VB:   Property Get Size() As Long
      Property Let Size(ByVal Val As Long)

C#:   virtual int SizePreserve { set; }
VB:   Property Let SizePreserve(ByVal Val As Long)
```

SizePreserve() allows enlarging the size without losing data.

```
C#:   virtual void CopyFrom(ref mvnaIQData IQData)
VB:   Sub CopyFrom(IQData As mvnaIQData)

C#:   virtual mvnaIQData GetCopy()
VB:   Function GetCopy() As mvnaIQData
```

6.17.2 Sample value manipulation

```
C#:   virtual double PLower { get; }
VB:   Property Get PLower() As Double

C#:   virtual double PUpper { get; }
VB:   Property Get PUpper() As Double
```

Return the lowest and highest values in PValues.

```
C#:   virtual double get_IValue(int Index)
C#:   virtual void set_IValue(int Index, double value)
VB:   Property Get IValue(ByVal Index As Long) As Double
VB:   Property Let IValue(ByVal Index As Long, ByVal Value As Double)

C#:   virtual double get_QValue(int Index)
C#:   virtual void set_QValue(int Index, double value)
VB:   Property Get QValue(ByVal Index As Long) As Double
VB:   Property Let QValue(ByVal Index As Long, ByVal Value As Double)

C#:   virtual double get_PValue(int Index)
C#:   virtual void set_PValue(int Index, double value)
VB:   Property Get PValue(ByVal Index As Long) As Double
VB:   Property Let PValue(ByVal Index As Long, ByVal Value As Double)

C#:   virtual mvnaIQ get_Value(int Index)
C#:   virtual void set_Value(int Index, ref mvnaIQ value)
VB:   Property Get Value(ByVal Index As Long) As mvnaIQ
VB:   Property Set Value(ByVal Index As Long, Value As mvnaIQ)

C#:   virtual System.Array IValues { get; }
C#:   virtual void GetIValues(ref System.Array Vals)
C#:   virtual void SetIValues(ref System.Array Vals)
VB:   Property Get IValues() As Double()  ' Creates and returns a new array
VB:   Sub GetIValues(Vals() As Double)    ' Redimensions and fills 'Vals()'
VB:   Sub SetIValues(Vals() As Double)    ' Copies 'Vals()' to 'IValues'

C#:   virtual System.Array QValues { get; }
C#:   virtual void GetQValues(ref System.Array Vals)
C#:   virtual void SetQValues(ref System.Array Vals)
```

```
VB:   Property Get QValues() As Double()
VB:   Sub GetQValues(Vals() As Double)
VB:   Sub SetQValues(Vals() As Double)

C#:   virtual System.Array PValues { get; }
C#:   virtual void GetPValues(ref System.Array Vals)
C#:   virtual void SetPValues(ref System.Array Vals)
VB:   Property Get PValues() As Double()
VB:   Sub GetPValues(Vals() As Double)
VB:   Sub SetPValues(Vals() As Double)

C#:   virtual System.Array Values { get; }
C#:   virtual void SetValues(ref System.Array Vals)
VB:   Property Get Values() As mvnaIQ()
VB:   Sub SetValues(Vals() As mvnaIQ)
```

When assigning an array to IQData, its Size will be adjusted automatically.

```
C#:   virtual void set_ValuesAll(ref mvnaIQ value)
VB:   Property Set ValuesAll(Value As mvnaIQ)
```

ValuesAll() sets all values to that of 'Value'

```
C#:   virtual void CreateIQData(ref double PMin, ref double PMax, ref double
      IVal, ref double QVal, ref int NrPoints)
VB:   Sub CreateIQData(PMin As Double, PMax As Double, IVal As Double, QVal As
      Double, NrPoints As Long)
```

CreateIQData() fills IQData with PValues from PMin to PMax and IValues and QValues with the constant value IVal and QVal. The object will be resized.

6.17.3 Array arithmetic

```
C#:   virtual mvnaIQData Zero { get; }
C#:   virtual mvnaIQData One { get; }
C#:   virtual mvnaIQData Two { get; }
VB:   Property Get Zero() As mvnaIQData
VB:   Property Get One() As mvnaIQData
VB:   Property Get Two() As mvnaIQData
```

Return an IQData object with IValues all 0, 1 or 2.

```
C#:   virtual mvnaIQData OneQ { get; }
VB:   Property Get OneQ() As mvnaIQData
```

Returns an IQData object with QValues all 1.

```
C#:   virtual mvnaIQData Neg { get; }
C#:   virtual mvnaIQData SqRoot { get; }
VB:   Property Get Neg() As mvnaIQData
VB:   Property Get SqRoot() As mvnaIQData

C#:   virtual mvnaIQData Add(ref mvnaIQData Val)
C#:   virtual mvnaIQData Subtract(ref mvnaIQData Val)
C#:   virtual mvnaIQData Multiply(ref mvnaIQData Val)
C#:   virtual mvnaIQData Divide(ref mvnaIQData Val)
VB:   Function Add(Val As mvnaIQData) As mvnaIQData
VB:   Function Subtract(Val As mvnaIQData) As mvnaIQData
VB:   Function Multiply(Val As mvnaIQData) As mvnaIQData
VB:   Function Divide(Val As mvnaIQData) As mvnaIQData
```

```
C#:   virtual mvnaIQData Amp { get; }
VB:   Property Get Amp() As mvnaIQData
```

Returns an IQData object with IValues set to the amplitudes.

```
C#:   virtual mvnaIQData get_Smooth(ref double Factor)
VB:   Property Get Smooth(Factor As Double) As mvnaIQData
```

Returns an IQData object with the values sequentially averaged. Factor is between 0 and 1.

6.17.4 Sample conversion

```
C#:   virtual double get_AmpValue(int Index)
C#:   virtual double get_AmpValueDB(int Index)
VB:   Property Get AmpValue(ByVal Index As Long) As Double
VB:   Property Get AmpValueDB(ByVal Index As Long) As Double
```

The conversion from I/Q to dB is always 20 * log(|Amp|)

```
C#:   virtual void GetAmpValuesDb(ref System.Array Vals)
C#:   virtual void GetAmpPhaseValuesDbDegrees(ref System.Array Amp, ref
      System.Array Phase)
VB:   Sub GetAmpValuesDb(Vals() As Double)
VB:   Sub GetAmpPhaseValuesDbDegrees(Amp() As Double, Phase() As Double)
```

Fill the arguments with the converted value arrays.

6.17.5 Computations

```
C#:   virtual void GetZ(ref System.Array Z, ref mvnaIQ Zo)
C#:   virtual void SetZ(ref System.Array Z, ref mvnaIQ Zo)
VB:   Sub GetZ(Z() As mvnaIQ, Zo As mvnaIQ)     ' Z   = -(S11+1) * Zo / (S11-1)
VB:   Sub SetZ(Z() As mvnaIQ, Zo As mvnaIQ)     ' S11 = (Z - Zo) / (Z + Zo)
```

SetZ() will convert the impedance values to I/Q values.

```
C#:   virtual void GetSWR(ref System.Array SWR)
C#:   virtual void GetFL(ref System.Array FL)
VB:   Sub GetSWR(SWR() As Double)               ' SWR = (1 + S11) / (1 - S11)
VB:   Sub GetFL(FL() As Double)                 ' FL  = 10 Log(1- S11^2)
```

FL is the Forward Loss: the power lost in forward direction due to impedance mismatch.

```
C#:   virtual void AverageData(ref mvnaIQData Data, double Average)
VB:   Sub AverageData(Data As mvnaIQData, ByVal Average As Double)
```

Calculates and stores the (running) average between the object and a new IQData object, with an Average factor between 0 and 1. A factor of 0.1 gives a (long term) average over 10 samples.

6.18 Class mvnaIQ

The IQ class holds a single I/Q value. It provides some arithmetic.

```
C#:   virtual double IVal { set; get; }
C#:   virtual double QVal { set; get; }
VB:   IVal As Double
VB:   QVal As Double

C#:   virtual mvnaIQ get_Value()
C#:   virtual void set_Value(ref mvnaIQ value)
VB:   Property Get Value() As mvnaIQ      ' Returns a new, copied class
VB:   Property Set Value(Val As mvnaIQ)

C#:   virtual double Amp { get; }
C#:   virtual double Phi { get; }
VB:   Property Get Amp() As Double
VB:   Property Get Phi() As Double

C#:   virtual void AssignPolar(double Amp, double Phi)
C#:   virtual void AssingPolarDB180(double Amp, double Phi)
VB:   Sub AssignPolar(ByVal Amp As Double, ByVal Phi As Double)
VB:   Sub AssingPolarDB180(ByVal Amp As Double, ByVal Phi As Double)

C#:   virtual double AmpDB { get; }
C#:   virtual double Phi180 { get; }
VB:   Property Get AmpDB() As Double
VB:   Property Get Phi180() As Double

C#:   virtual mvnaIQ Zero { get; }
C#:   virtual mvnaIQ One { get; }
C#:   virtual mvnaIQ OneQ { get; }
C#:   virtual mvnaIQ Two { get; }
VB:   Property Get Zero() As mvnaIQ
VB:   Property Get One() As mvnaIQ
VB:   Property Get OneQ() As mvnaIQ
VB:   Property Get Two() As mvnaIQ

C#:   virtual mvnaIQ Neg { get; }
VB:   Property Get Neg() As mvnaIQ

C#:   virtual mvnaIQ Add(ref mvnaIQ V)
C#:   virtual mvnaIQ Subtract(ref mvnaIQ V)
C#:   virtual mvnaIQ Multiply(ref mvnaIQ V)
C#:   virtual mvnaIQ Divide(ref mvnaIQ V)
VB:   Function Add(V As mvnaIQ) As mvnaIQ
VB:   Function Subtract(V As mvnaIQ) As mvnaIQ
VB:   Function Multiply(V As mvnaIQ) As mvnaIQ
VB:   Function Divide(V As mvnaIQ) As mvnaIQ

C#:   virtual bool get_IsEqual(ref mvnaIQ Val)
VB:   Property Get IsEqual(Val As mvnaIQ) As Boolean
```

6.19 Enum mvnaVNAStatus

```
mvnaVST_Disconnected     = 0
mvnaVST_Initializing     = 1   ' Connecting and initializing
mvnaVST_Idle             = 2
mvnaVST_Calibrating      = 3
mvnaVST_Sweeping         = 4
```

6.20 Enum mvnaSweepType

```
mvnaSWT_Lin              = 0
mvnaSWT_List             = 2
```

6.21 Enum mvnaVNAPort

Bitmask:

```
mvnaVNP_None             = 0
mvnaVNP_Port1            = 1
mvnaVNP_Port2            = 2
mvnaVNP_Port3            = 4
```

6.22 Enum mvnaVNADataOptions

Bit mask:

```
mvnaVDO_NONE             = 0
mvnaVDO_CALIBRATION      = 2
mvnaVDO_DATA             = 4
```

6.23 Enum mvnaVNABiasControl

```
mvnaVBC_Off              = 0
mvnaVBC_Ground           = 1
mvnaVBC_On               = 2
```

6.24 Enum mvnaWindowShowState

```
mvnaWSS_Hidden           = 0
mvnaWSS_Minimized        = 1
mvnaWSS_Normal           = 2
mvnaWSS_Maximized        = 3
```

6.25 Enum mvnaColor

Bitmask:

```
mvnaCOL_None             = 0
mvnaCOL_Red              = 1
mvnaCOL_Green            = 2
mvnaCOL_Blue             = 4
mvnaCOL_All              = 7
```

6.26 Enum mvnaLedState

```
mvnaLED_Off              = 0
mvnaLED_On               = 1
mvnaLED_Blink            = 2
mvnaLED_BlinkFast        = 3
```