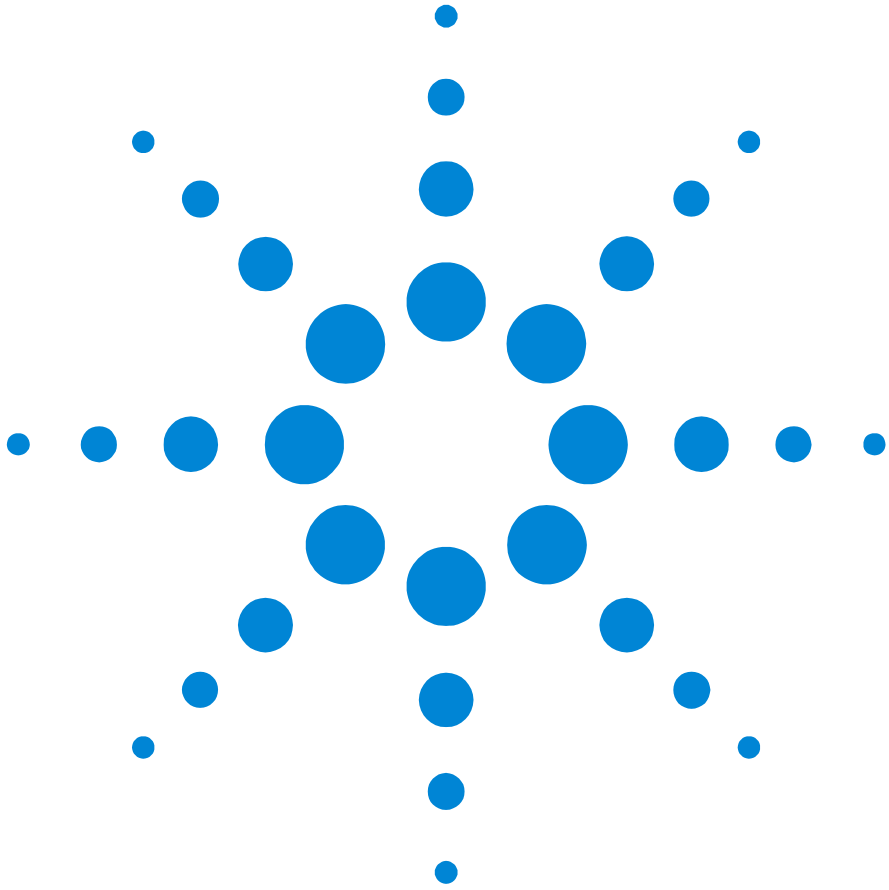


Agilent ParBERT 81250 Measurement Software

Bit Error Rate Measurement Programming Reference

Agilent Technologies

Important Notice

Revision

Revision 4.3B, July 2002

Printed in Germany

Agilent Technologies
Herrenberger Straße 130
D-71034 Böblingen
Germany

Authors: t3 medien GmbH

Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

Safety Notices

CAUTION

A CAUTION notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a CAUTION notice until the indicated conditions are fully understood and met.

WARNING

A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

Trademarks

Windows NT ® and MS Windows ® are U.S. registered trademarks of Microsoft Corporation.

Contents

Contents

About this Reference

This document describes the functions, properties and methods for controlling the Bit Error Rate measurement from a remote application.

NOTE Basic knowledge on handling the *Agilent ParBERT 81250 Measurement Software* is assumed. For further information, refer to the *Framework User Guide* and the *Bit Error Rate Measurement User Guide*.

For general information on remote programming, refer to the *Measurement Software Programming Guide*.

Programming Reference

The following sections describe the methods and properties from a Visual Basic, VEE or LabView user perspective. Some differences in the syntax exist for the Visual C (VC) user. The VC syntax is denoted in each of the properties and methods.

NOTE

- Some of the methods, events and properties are not available to the wrapper dll user.
- The methods `InitMeasurement` and `CloseMeasurement` are only available to the wrapper dll user.

The functions are sorted according to the following categories:

- *"Measurement Basics" on page 10* shows the functions used to handle measurements, to establish the connection to the firmware server and to work with the settings.
- *"Measurement Setup" on page 31* shows the functions used to work with ports and terminals, and to set the parameters for the measurement.
- *"Running the Measurement" on page 59* shows the functions used to run and stop the measurement.
- *"Handling Events and Callbacks" on page 63* shows the functions used to handle events and callbacks for the measurement.
- *"Error Handling" on page 67* shows the functions used to analyze errors.
- *"Handling Measurement Results" on page 69* shows the functions used to get, calculate, and modify measurement results.
- *"Pass/Fail Functions" on page 82* shows the functions used to set and evaluate pass/fail decisions.
- *"Copy/Paste Functions" on page 90* shows the functions used to edit the data display.

- *"Persistence" on page 98* shows the functions used to load/save measurements and to export data from the measurements.
- *"Functions for General Purposes" on page 110* shows the functions used to access the online help, for example.

Measurement Basics

The following section shows the functions used to handle measurements, to establish the firmware connection and to work with systems and system settings.

Functions to Get/Delete Measurement Handles

The following table gives an overview on the methods, events and properties available to handle measurements:

Purpose	Refer to...
To close a specific measurement.	*"CloseMeasurement" on page 14*
To initialize a measurement.	*"InitMeasurement" on page 24*

Functions to Establish the Firmware Connection

The following table gives an overview on the methods, events and properties available to control measurements:

Purpose	Refer to...
To set the connection to the firmware server.	*"Server" on page 26*
To set the port the firmware server is connected to.	*"ServerPort" on page 27*
To specify the analyzer system.	*"AnalyzerSystem" on page 12*
To specify the generator system.	*"GeneratorSystem" on page 18*
To specify the name of a system to be delayed.	*"DelayStartSystem" on page 17*
To specify a start delay between two systems.	*"StartDelay" on page 28*
To create a new measurement.	*"CreateMeasurementEx" on page 15*
To prepare the measurement execution.	*"CreateMeasEx" on page 14*
To check whether the connection to the firmware server is valid.	*"IsFWSConnected" on page 24*

Working with Settings

The following table gives an overview on the methods, events and properties available to handle systems and system settings:

Purpose	Refer to...
To get the number of system settings defined for the analyzer.	*"GetAnalyzerSettingsCount" on page 20*
To get the names of the system settings defined for the analyzer.	*"GetAnalyzerSettingsName" on page 21*
To get the number of system settings defined for the generator.	*"GetGeneratorSettingsCount" on page 22*
To get the names of the system settings defined for the generator.	*"GetGeneratorSettingsName" on page 23*
To specify the system setting for the analyzer.	*"AnalyzerSystemSetting" on page 13*
To specify the system setting for the generator.	*"GeneratorSystemSetting" on page 19*
To send the analyzer's system setting to the firmware server.	*"UseAnalyzerSettings" on page 29*
To send the generator's system setting to the firmware server.	*"UseGeneratorSettings" on page 30*

AnalyzerSystem

ActiveX syntax

```
Object.AnalyzerSystem = [sSystem]
```

For Visual C:

```
Object.SetAnalyzerSystem(sSystem)
sSystem = Object.GetAnalyzerSystem()
```

Wrapper dll syntax

```
BERSetAnalyzerSystem(hMeasurement,
                     sSystem)
BERGetAnalyzerSystem(hMeasurement,
                     bufferSize,
                     *sSystem)
```

Description

Sets/returns the name of the analyzer system. The analyzer system specifies the analyzer for the measurement together with the related system setting.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

sSystem Specifies the name of the analyzer system (data type: `string`). For the wrapper dll GET function, this parameter is a pointer.

Example

To set the analyzer system "DSRA":

```
m_BERCTRL.AnalyzerSystem = "DSRA"
```

Related functions and methods

"AnalyzerSystemSetting" on page 13
"GeneratorSystem" on page 18
"CreateMeasurementEx" on page 15

AnalyzerSystemSetting

ActiveX syntax

```
Object.AnalyzerSystemSetting = [sSetting]
```

For Visual C:

```
Object.SetAnalyzerSystemSetting(sSetting)
sSetting = Object.GetAnalyzerSystemSetting()
```

Wrapper dll syntax

```
BERSetAnalyzerSystemSetting(hMeasurement,
                            sSetting)
BERGetAnalyzerSystemSetting(hMeasurement,
                            bufferSize,
                            *sSetting)
```

Description Sets/returns the name of the analyzer system setting specified in the *Agilent 81250 User Software*. The analyzer system specifies together with the related system setting the analyzer for the measurement.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

sSetting Specifies the name of the analyzer system setting (data type: `string`). For the wrapper dll GET function, this parameter is a pointer.

Example To set the analyzer system setting "TEST_APP":

```
m_BERCTRL.AnalyzerSystemSetting = "TEST_APP"
```

Related functions and methods *"AnalyzerSystem" on page 12*

CloseMeasurement

NOTE This method is only available when using the wrapper dll.

Wrapper dll syntax

```
BERCloseMeasurement(hMeasurement)
```

Description Closes the measurement. The handle to the measurement is deleted. Any subsequent property or method using the handle will return an error.

Input parameter **hMeasurement.** Handle to the measurement (data type: `ViSession`) returned by `InitMeasurement`.

Related functions and methods

"CreateMeasurementEx" on page 15
"InitMeasurement" on page 24

CreateMeasEx

ActiveX syntax

```
Object.CreateMeasEx()
```

Wrapper dll syntax

```
BERCreateMeasEx(hMeasurement)
```

Description Creates the connection to the server, sets the generator and analyzer systems, delay system and delay. It assumes that the server name, port number, analyzer and generator system, delay system and delay value have been set by using the appropriate objects properties.

Input parameter **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

Remarks You have to set the following properties before using this method: `Server`, `ServerPort`, `AnalyzerSystem`, `GeneratorSystem`, `DelayStartSystem`, `StartDelay`.

After the `CreateMeasEx` call, use `IsFWSConnected` to check for the firmware connection.

Related functions and methods

"Server" on page 26
"ServerPort" on page 27
"AnalyzerSystem" on page 12
"GeneratorSystem" on page 18
"DelayStartSystem" on page 17
"StartDelay" on page 28
"IsFWSConnected" on page 24

CreateMeasurementEx

ActiveX syntax

```
Object.CreateMeasurementEx(sServer,
                           sPort,
                           sAnalyzerSystem,
                           sGeneratorSystem,
                           sDelaySystem,
                           dDelay,
                           measType)
```

Wrapper dll syntax

```
BERCreateMeasurementEx(hMeasurement,
                       sServer,
                       sPort,
                       sAnalyzerSystem,
                       sGeneratorSystem,
                       sDelaySystem,
                       dDelay,
                       measType)
```

Description

Creates the connection to the server, sets the generator and analyzer systems, delay system, delay, and measurement type.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement (data type: `ViSession`).

sServer IP address of the server or LOCALHOST, for example, "10.3.6.14" (data type: `String`).

sPort Port number, set 2203 for the default port (data type: `String`).

sAnalyzerSystem Name of the analyzer system, for example, "DSRA" (data type: `String`).

sGeneratorSystem Name of the generator system, for example, "DSRB" (data type: `String`).

sDelaySystem Name of the delayed system. It should either be the same name as the analyzer system or the generator system. If the analyzer and generator systems are the same, set this to an empty string (data type: `String`).

dDelay Delay in seconds that the `sDelaySystem` is delayed from the other system (data type: `Double`).

measType Measurement type; can be either `MeasTypeElectrical`, `MeasTypeOptical`, or `MeasTypeAll` (data type: `MeasTypeEnums`).

Example

To connect to the firmware server under IP address **10.3.6.14**, port number **2203**, using the system **DSRA** as analyzer and generator, with no start delay, for a system with only electrical connections:

```
m_BERCTRL.CreateMeasurementEx("10.3.6.14","2203","DSRA",
                              "DSRA","",0,MeasTypeElectrical)
```

Related functions and methods

"AnalyzerSystem" on page 12
"AnalyzerSystemSetting" on page 13
"DelayStartSystem" on page 17
"GeneratorSystem" on page 18
"GeneratorSystemSetting" on page 19
"MeasurementType" on page 25
"StartDelay" on page 28
"Server" on page 26
"ServerPort" on page 27

DelayStartSystem

ActiveX syntax

```
Object.DelayStartSystem = [sDelaySystem]
```

For Visual C:

```
Object.SetDelayStartSystem(sDelaySystem)
sDelaySystem = Object.GetDelayStartSystem()
```

Wrapper dll syntax

```
BERSetDelayStartSystem(hMeasurement,
                       sDelaySystem)
BERGetDelayStartSystem(hMeasurement,
                       bufferSize,
                       *sDelaySystem)
```

Description

Sets the name of the system that is delayed.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

sDelaySystem Name (data type: `String`) of the delayed system. It should be either the name of the analyzer or generator system set by `AnalyzerSystem` and `GeneratorSystem`. The delay system can be a null string, resulting in no delay. For the wrapper dll GET function, this parameter is a pointer.

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

Example

To define a delay for the system **`DSRA`**:

```
m_BERCTRL.DelayStartSystem = "DSRA"
```

Related functions and methods

"AnalyzerSystem" on page 12
"GeneratorSystem" on page 18
"StartDelay" on page 28

GeneratorSystem

ActiveX syntax

```
Object.GeneratorSystem = [sSystem]
```

For Visual C:

```
Object.SetGeneratorSystem(sSystem)
sSystem = Object.GetGeneratorSystem()
```

Wrapper dll syntax

```
BERSetGeneratorSystem(hMeasurement,
                      sSystem)
BERGetGeneratorSystem(hMeasurement,
                      bufferSize,
                      *sSystem)
```

Description

Sets/gets the name of the generator system. The generator system specifies together with the related system setting the generator for the measurement.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

sSystem Specifies the name of the generator system (data type: `string`). For the wrapper dll GET function, this parameter is a pointer.

Example

To select the generator system "DSRA":

```
m_BERCTRL.GeneratorSystem = "DSRA"
```

Related functions and methods

"AnalyzerSystem" on page 12
"GeneratorSystemSetting" on page 19

GeneratorSystemSetting

ActiveX syntax

```
Object.GeneratorSystemSetting = [sSetting]
```

For Visual C:

```
Object.SetGeneratorSystemSetting(sSetting)
sSetting = Object.GetGeneratorSystemSetting()
```

Wrapper dll syntax

```
BERSetGeneratorSystemSetting(hMeasurement,
                             sSetting)

BERGetGeneratorSystemSetting(hMeasurement,
                             bufferSize,
                             *sSetting)
```

Description Sets/returns the name of the generator system setting specified in the *Agilent 81250 User Software*. The generator system specifies the generator for the measurement together with the related system setting.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

sSetting Specifies the name of the generator system setting (data type: `String`). For the wrapper dll GET function, this parameter is a pointer.

Example To set the generator system setting "TEST_APP":

```
m_BERCTRL.GeneratorSystemSetting = "TEST_APP"
```

Related functions and methods

"GeneratorSystem" on page 18
"AnalyzerSystemSetting" on page 13

GetAnalyzerSettingsCount

ActiveX syntax

```
nItems = Object.GetAnalyzerSettingsCount()
```

Wrapper dll syntax

```
BERGetAnalyzerSettingsCount(hMeasurement,
                            *nItems)
```

Description

Returns the number of settings stored in firmware for the analyzer system.

Output parameter

nItems Number of settings for the analyzer system (data type: `Integer`). For the wrapper dll GET function, this parameter is a pointer.

Input parameter

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

Example

To get the number of settings defined for the analyzer system:

```
Dim nItems as Integer
nItems = m_BERCTRL.GetAnalyzerSettingsCount()
```

Related functions and methods

"GetAnalyzerSettingsName" on page 21
"GetGeneratorSettingsCount" on page 22

GetAnalyzerSettingsName

ActiveX syntax

```
sSettingName = Object.GetAnalyzerSettingsName(nIndex)
```

Wrapper dll syntax

```
BERGetAnalyzerSettingsName(hMeasurement,
                           nIndex,
                           bufferSize,
                           *sSettingName)
```

Description Returns the setting name of the analyzer for a designated index.

Output parameter

sSettingName Name of the analyzer setting (data type: `String`). For the wrapper dll GET function, this parameter is a pointer.

Input parameter

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nIndex Unique identifier for the setting. This is an index beginning at 0 (data type: `Integer`).

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

Example To get the first setting name for the analyzer:

```
Dim sSettingName as String
sSettingName = m_BERCTRL.GetAnalyzerSettingsName(0)
```

Related functions and methods

"GetAnalyzerSettingsCount" on page 20
"GetGeneratorSettingsName" on page 23

GetGeneratorSettingsCount

ActiveX syntax

```
nItems = Object.GetGeneratorSettingsCount()
```

Wrapper dll syntax

```
BERGetGeneratorSettingsCount(hMeasurement,
                             *nItems)
```

Description Returns the number of settings stored in firmware for the generator system.

Output parameter **nItems** Number of settings (data type: `Integer`) for the generator system. For the wrapper dll GET function, this parameter is a pointer.

Input parameter **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

Example To get the number of system settings defined for the generator system:

```
Dim nItems as Integer
nItems = m_BERCTRL.GetGeneratorSettingsCount()
```

Related functions and methods

"GetGeneratorSettingsName" on page 23
"GetAnalyzerSettingsCount" on page 20

GetGeneratorSettingsName

ActiveX syntax

```
sSettingName = Object.GetGeneratorSettingsName(nIndex)
```

Wrapper dll syntax

```
BERGetGeneratorSettingsName(hMeasurement,
                            nIndex,
                            bufferSize,
                            *sSettingName)
```

Description Returns the setting name of the generator for a designated index.

Output parameter

sSettingName Name of the generator setting (data type: `String`). For the wrapper dll GET function, this parameter is a pointer.

Input parameter

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nIndex Unique identifier (data type: `Integer`) for the setting, an index starting at 0.

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

Example To get the first setting name for the generator:

```
Dim sSettingName as String
sSettingName = m_BERCTRL.GetGeneratorSettingsName(0)
```

Related functions and methods

"GetAnalyzerSettingsName" on page 21
"GetGeneratorSettingsCount" on page 22

InitMeasurement

NOTE This method is only available when using the wrapper dll.

Wrapper dll syntax

```
ViSession hMeasurement = VI_NULL;
BERInitMeasurement(&hMeasurement)
```

Description Initializes the measurement and the handle to the measurement is returned. Any subsequent property or method calls should use the handle value that is returned.

Output parameter **hMeasurement** Handle to the measurement (data type: `ViSession`).

Related functions and methods *"CloseMeasurement" on page 14*

IsFWSConnected

ActiveX syntax

```
bConnect = Object.IsFWSConnected()
```

Wrapper dll syntax

```
BERIsFWSConnected(hMeasurement,
                  *bConnect)
```

Description Returns whether there is a connection to the firmware server. To run the measurement a connection to the firmware server must be established.

Output parameter **bConnect** The following constants (data type: `Boolean`) are returned:

Constant	Description
`True`	There is a connection to the firmware server.
`False`	There is no connection to the firmware server.

Input parameter **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

Example To check the connection to the firmware server:

```
Dim bConnect as Boolean
bConnect = m_BERCTRL.IsFWSConnected()
```

Related functions and methods *"Server" on page 26*
"ServerPort" on page 27

MeasurementType

ActiveX syntax

```
Object.MeasurementType = [measType]
```

For Visual C:

```
Object.SetMeasurementType(measType)
measType = Object.GetMeasurementType()
```

Wrapper dll syntax

```
BERSetMeasurementType(hMeasurement,
                      measType)
BERGetMeasurementType(hMeasurement,
                      *measType)
```

Description Sets/returns the measurement's *type* (whether the measurement is electrical only, optical only, or a combination of both).

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

measType Type of the measurement (data type: `MeasTypeEnums`). The value can be either `MeasTypeElectrical`, `MeasTypeOptical`, or `MeasTypeAll` (both types in one system). For the wrapper dll GET function, this parameter is a pointer.

Example To set the measurement type for electrical only:

```
m_BERCTRL.MeasurementType = MeasTypeElectrical
```

Related functions and methods *"CreateMeasurementEx" on page 15*

Server

ActiveX syntax

```
Object.Server = [sServer]
```

For Visual C:

```
Object.SetServer(sServer)
sServer = Object.GetServer()
```

Wrapper dll syntax

```
BERSetServer(hMeasurement,
             sServer)
BERGetServer(hMeasurement,
             bufferSize,
             *sServer)
```

Description Sets/returns the server name for the *81200 Firmware Server*.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

sServer Address of the server (data type: `String`). If the 81200 firmware server is located on the local machine, any empty string "" is used. For the wrapper dll GET function, this parameter is a pointer.

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

Example To establish a connection to the server address "10.3.6.14":

```
m_BERCTRL.Server = "10.3.6.14"
```

Related functions and methods *"ServerPort" on page 27*

ServerPort

ActiveX syntax

```
Object.ServerPort = [sPort]
```

For Visual C:

```
Object.SetServerPort(sPort)
SPort = Object.GetServerPort()
```

Wrapper dll syntax

```
BERSetServerPort(hMeasurement,
                 bufferSize,
                 sPort)
BERGetServerPort(hMeasurement,
                 *sPort)
```

Description Sets/returns the port id for the firmware server connection.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

sPort Port number for the firmware server connection (data type: `String`). For the wrapper dll GET function, this parameter is a pointer.

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

Example To establish a connection to the server located at port "2203":

```
m_BERCTRL.ServerPort = "2203"
```

Related functions and methods *"Server" on page 26*

StartDelay

ActiveX syntax

```
Object.StartDelay = [dDelay]
```

For Visual C:

```
Object.SetStartDelay(dDelay)
dDelay = Object.GetStartDelay()
```

Wrapper dll syntax

```
BERGetStartDelay(hMeasurement,
                 *dDelay)
BERSetStartDelay(hMeasurement,
                 dDelay)
```

Description

Sets the start delay between the two systems. The `DelayStartSystem` property defines which system is delayed, the generator or analyzer.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

dDelay Start delay in seconds between the two systems (data type: `Double`). For the wrapper dll GET function, this parameter is a pointer.

Example

To set the delay time to 0.5 seconds:

```
m_BERCTRL.StartDelay = 0.5
```

Related functions and methods

"DelayStartSystem" on page 17

UseAnalyzerSettings

ActiveX syntax

```
Object.UseAnalyzerSettings = [boolean]
```

For Visual C:

```
Object.SetUseAnalyzerSettings(boolean)
boolean = Object.GetUseAnalyzerSettings()
```

NOTE This property is not available for the wrapper dll.

Description Sets/returns whether the analyzer settings will be sent to the firmware server.

Parameters **boolean** The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Send the analyzer settings to the firmware server.
`False`	Do not send the analyzer settings to the firmware server (default setting).

Remarks If this parameter is set to `TRUE`, a setting must be selected for the analyzer.

Example To send the selected system setting for the analyzer to the firmware server:

```
m_BERCTRL.UseAnalyzerSettings = True
```

Related functions and methods *"UseGeneratorSettings" on page 30*

UseGeneratorSettings

ActiveX syntax

```
Object.UseGeneratorSettings = [boolean]
```

For Visual C:

```
Object.SetUseGeneratorSettings(boolean)
Boolean = Object.GetUseGeneratorSettings()
```

NOTE This property is not available for the wrapper dll.

Description Sets/returns whether the generator settings will be sent to the firmware server.

Parameter **boolean** The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Send the selected generator setting to the firmware server.
`False`	Do not send the generator setting to the firmware server (default setting).

Remarks If this parameter is set to true, a setting must be selected for the generator.

Example To send the system setting specified for the generator to the firmware server:

```
m_BERCTRL.UseGeneratorSettings = True
```

Related functions and methods *"UseAnalyzerSettings" on page 29*

Measurement Setup

The following section shows the functions used to handle the parameters for the measurement.

Working with Ports and Terminals

The following table gives an overview on the methods, events and properties available to handle ports and terminals involved into the measurement:

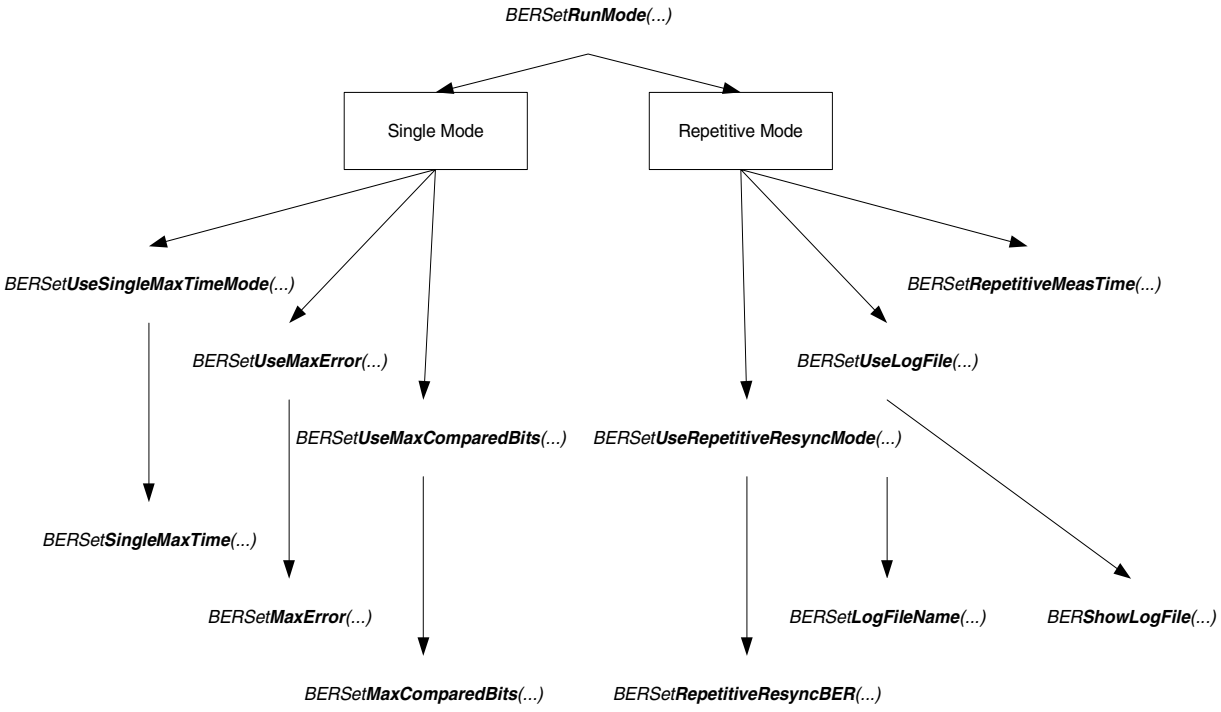
Purpose	Refer to...
To get the number of analyzer ports configured in the system.	*"GetAnalyzerPortCount" on page 34*
To get the names of analyzer ports configured in the system.	*"GetAnalyzerPortName" on page 35*
To get the number of analyzer terminals configured in the system.	*"GetAnalyzerTermCount" on page 36*
To get the name of one analyzer terminal.	*"GetAnalyzerTermName" on page 37*
To get a port id for a given index.	*"GetPortId" on page 38*
To get a terminal id for a given index and port.	*"GetTerminalId" on page 39*
To set/return if a given port is part of the measurement.	*"PortInvolved" on page 44*
To set/return if a given terminal is part of the measurement.	*"TermInvolved" on page 52*

Setting Bit Error Rate Parameters

The following table gives an overview on the methods, events and properties available to handle the values of the *Parameters* page:

Purpose	Refer to...
To set/return the measurement running mode.	*"RunMode" on page 47*
To set/return whether the measurement stops after a specified time.	*"UseSingleMaxTimeMode" on page 58*
To set/return the time before the measurement stops.	*"SingleMaxTime" on page 51*
To set/return whether the measurement stops after a specified number of compared bits.	*"UseMaxComparedBits" on page 55*
To set the number of bits to be compared before the measurement stops.	*"MaxComparedBits" on page 42*
To activate the property MaxError.	*"UseMaxError" on page 56*
To set the number of errors to be compared before the measurement stops.	*"MaxError" on page 43*
To set/return the measurement period when the measurement is running in repetitive mode.	*"RepetitiveMeasTime" on page 46*
To set/return whether resynchronization is performed after a specific bit error rate.	*"UseRepetitiveResyncMode" on page 57*

Purpose	Refer to...
To set the bit error rate to be measured before resynchronization is performed.	*"RepetitiveResyncBER" on page 48*
To set/return whether a log file is generated.	*"UseLogFile" on page 54*
To show the log file.	*"ShowLogFile" on page 50*
To set/return the name of the log file.	*"LogFileName" on page 41*



Setting Bit Error Rate Pass/Fail Parameters

The following table gives an overview on the methods, events and properties available to handle the values of the *Pass/Fail* page:

Purpose	Refer to...
To specify the BER threshold.	*"BERThreshold" on page 33*
To activate the pass/fail check for the BERThreshold.	*"UseBERThreshold" on page 53*

Setting Bit Error Rate View Parameters

The following table gives an overview on the methods, events and properties available to handle the values of the *View* page:

Purpose	Refer to...
To set the number of decimal places to be displayed in the numerical view.	*"GridPrecision" on page 40*

Specify the Properties Dialog

The following table gives an overview on the methods, events and properties available to display the *Properties* dialog:

Purpose	Refer to...
To set the title of the *Properties* dialog.	*"PropertiesTitle" on page 45*
To display the *Properties* dialog.	*"ShowProperties" on page 50*

BERThreshold

ActiveX syntax

```
Object.BERThreshold = [dBERThreshold]
```

For Visual C:

```
Object.SetBERThreshold(dBERThreshold)
dBERThreshold = Object.GetBERThreshold()
```

Wrapper dll syntax

```
BERGetBERThreshold(hMeasurement,
                   *dBERThreshold)
BERSetBERThreshold(hMeasurement,
                   dBERThreshold)
```

Description

Sets/returns the BER threshold value that is used as Pass/Fail criterion for the measurement.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

dBERThreshold Specifies the BER threshold (data type: `Double`). For the wrapper dll GET function, this parameter is a pointer.

Example

To set the BER threshold to $4 \cdot 10^{-3}$:

```
m_BERCTRL.BERThreshold = 4E-3
```

Related functions and methods

"UseBERThreshold" on page 53

GetAnalyzerPortCount

ActiveX syntax

```
nPorts = Object.GetAnalyzerPortCount()
```

Wrapper dll syntax

```
BERGetAnalyzerPortCount(hMeasurement,
                        *nPorts)
```

Description Returns the number of analyzer ports configured on the 81200 hardware.

Output parameter **nPorts** Number of data ports (data type: `Integer`) configured on the 81200 hardware. For the wrapper dll GET function, this parameter is a pointer.

Input parameter **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

Example To get the number of analyzer ports:

```
Dim nPorts as Integer
nPorts = m_BERCTRL.GetAnalyzerPortCount()
```

Related functions and methods

"GetAnalyzerPortName" on page 35
"GetAnalyzerTermCount" on page 36

GetAnalyzerPortName

ActiveX syntax

```
sName = Object.GetAnalyzerPortName(nPortID)
```

Wrapper dll syntax

```
BERGetAnalyzerPortName(hMeasurement,
                       nPortID,
                       bufferSize,
                       *sName)
```

Description Returns the analyzer port name for a designated port id.

Output parameters

nPortID A port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

sName Name of the port as configured on the GUI (data type: `String`). For the wrapper dll GET function, this parameter is a pointer.

Input parameter

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

Example To get the name of port 1:

```
Dim sName as String
sName = m_BERCTRL.GetAnalyzerPortName(1)
```

Related functions and methods

"GetAnalyzerPortCount" on page 34
"GetAnalyzerTermName" on page 37

GetAnalyzerTermCount

ActiveX syntax

```
nTermCount = Object.GetAnalyzerTermCount(nPortID)
```

Wrapper dll syntax

```
BERGetAnalyzerTermCount(hMeasurement,
                        nPortID,
                        *nTermCount)
```

Description

Returns the number of terminals for the specified port as configured on the 81200 hardware.

Output parameter

nTermCount Number of terminals for the specified port (data type: `Integer`). For the wrapper dll GET function, this parameter is a pointer.

Input parameter

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nPortID A port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

Example

To get the number of terminals configured for port 1:

```
Dim nTermCount as Integer
nTermCount = m_BERCTRL.GetAnalyzerTermCount(1)
```

Related functions and methods

"GetAnalyzerTermName" on page 37
"GetAnalyzerPortCount" on page 34

GetAnalyzerTermName

ActiveX syntax

```
sName = Object.GetAnalyzerTermName(nPortID, nTerminalID)
```

Wrapper dll syntax

```
BERGetAnalyzerTermName(hMeasurement,
                       nPortID,
                       nTerminalID,
                       bufferSize,
                       *sName)
```

Description Returns the analyzer terminal name for a designated terminal id.

Output parameter

sName Name of the port as configured on the GUI (data type: `String`). For the wrapper dll GET function, this parameter is a pointer.

Input parameter

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nPortID A port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

nTerminalID A terminal is addressed by the terminal number (data type: `Integer`). This is an index starting at 1 for each port.

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

Example To get the name of terminal 1:

```
Dim sName as String
sName = m_BERCTRL.GetAnalyzerTermName(1)
```

Related functions and methods

"GetAnalyzerTermCount" on page 36
"GetAnalyzerPortName" on page 35

GetPortId

ActiveX syntax

```
nPortID = Object.GetPortId(nIndex)
```

Wrapper dll syntax

```
BERGetPortId(hMeasurement,
             nIndex,
             *nPortID)
```

Description

Returns the port id associated with an index. In some configurations, the ports are not sequential and may not begin with port ID = 1. This method allows you to uniquely identify ports.

Output parameter

nPortID A port is addressed by the port number (data type: `Integer`). This is an index starting at 1. For the wrapper dll GET function, this parameter is a pointer.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nIndex Unique identifier (data type: `Integer`) for the port, an index starting at 0.

Example

To get the port ID for the first index:

```
Dim nPortID as Integer
nPortID = m_BERCTRL.GetPortId(0)
```

Related functions and methods

"GetTerminalId" on page 39

GetTerminalId

ActiveX syntax

```
nTermID = Object.GetTerminalId(nIndex,
                               nPortID)
```

Wrapper dll syntax

```
BERGetTerminalId(hMeasurement,
                 nIndex,
                 nPortID,
                 *nTermID)
```

Description Returns the terminal id associated with an index and a port. In some configurations, the terminals are not sequential and may not begin with terminal ID = 1. This method allows you to uniquely identify terminals.

Output parameter

nTermID Returned value: A terminal is addressed by the terminal number (data type: `Integer`). For the wrapper dll GET function, this parameter is a pointer.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nIndex Index starting at 0 (data type: `Integer`).

nPortID A port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

Example To get the terminal ID for the first index of port 1:

```
Dim nTermID as Integer
nTermID = m_BERCTRL.GetTerminalId(0, 1)
```

Related functions and methods *"GetTerminalId" on page 39*

GridPrecision

ActiveX syntax

```
Object.GridPrecision = [ePrecision]
```

For Visual C:

```
Object.SetGridPrecision(ePrecision)
ePrecision = Object.GetGridPrecision()
```

NOTE This property is not available for the wrapper dll.

Description Sets/returns the number of decimal places shown in the numerical view.

Parameter **ePrecision** The following values (data type: `PRECISION`) are defined:

Constant	Description
`Zero`	No significant digits after the decimal place are shown.
`One`	One significant digit after the decimal place is shown.
`Two`	Two significant digits after the decimal place are shown.
`Three`	Three significant digits after the decimal place are shown.

Example To set the number of decimal places to be displayed to 2:

```
m_BERCTRL.GridPrecision = Two
```

LogFileName

ActiveX syntax

```
Object.LogFileName = [sName]
```

For Visual C:

```
sName = Object.GetLogFileName()
Object.SetLogFileName(sName)
```

Wrapper dll syntax

```
BERGetLogFileName(hMeasurement,
                  *sName)

BERSetLogFileName(hMeasurement,
                  sName)
```

Description Sets/returns the name of the log file.

NOTE This value can only be set, if the repetitive run mode and the creation of a log file are activated. To return/set these modes, use *"RunMode" on page 47* and *"UseLogFile" on page 54* successively.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

sName Specifies the name of the log file (data type: `String`). For the wrapper dll GET function, this parameter is a pointer.

Full path and name of the log file (data type: `String`). The Measurement User Interface application stores these files with an extension of `txt`. For the wrapper dll GET function, this parameter is a pointer.

Example To set the log file name to "BER.txt":

```
m_BERCTRL.SetLogFileName("C:\\Temp\\BER.txt")
```

Related functions and methods *"ShowLogFile" on page 50*

MaxComparedBits

ActiveX syntax

```
Object.MaxComparedBits = [dMComparedBits]
```

For Visual C:

```
Object.SetMaxCompareBits(dMComparedBits)
dMComparedBits = Object.GetMaxCompareBits()
```

Wrapper dll syntax

```
BERGetMaxComparedBits(hMeasurement,
                      *dMComparedBits)

BERSetMaxComaredBits(hMeasurement,
                     dMComparedBits)
```

Description

Sets/returns the maximum number of compared bits. This value is used as a stop criterion for the measurement.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

dMComparedBits Number of compared bits that must be reached before the measurement stops (data type: `Double`). For the wrapper dll GET function, this parameter is a pointer.

Example

To stop the measurement after 100000 compared bits:

```
m_BERCTRL.MaxComparedBits = 100000
```

Related functions and methods

"MaxError" on page 43

MaxError

ActiveX syntax

```
Object.MaxError = [dErrors]
```

For Visual C:

```
Object.SetMaxError(dErrors)
dErrors = Object.GetMaxError()
```

Wrapper dll syntax

```
BERGetMaxError(hMeasurement,
               *dErrors)
BERSetMaxError(hMeasurement,
               dErrors)
```

Description

Sets/returns the maximum number of errors. This value is used as a stop criterion for the measurement.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

dErrors Maximum number of errors that must be reached before the measurement stops (data type: `Double`). For the wrapper dll GET function, this parameter is a pointer.

Example

To allow a maximum of 1000 errors:

```
m_BERCTRL.MaxError = 1000
```

Related functions and methods

"MaxComparedBits" on page 42
"UseMaxError" on page 56

PortInvolved

ActiveX syntax

```
Object.SetPortInvolved(nPortID,
                       boolean)

boolean = Object.GetPortInvolved(nPortID)
```

Wrapper dll syntax

```
BERSetPortInvolved(hMeasurement,
                   nPortID,
                   boolean)

BERGetPortInvolved(hMeasurement,
                   nPortID,
                   *boolean)
```

Description

Sets/returns whether a port is involved in the measurement.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nPortID A port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

boolean Indicates if the port is involved in the measurement. The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Port is involved in the measurement.
`False`	Port is not involved in the measurement.

For the wrapper dll GET function, this parameter is a pointer.

Example

To set port 1 involved in the measurement:

```
m_BERCTRL.SetPortInvolved(1, True)
```

To get whether port 1 is involved in the measurement:

```
Dim bIsInvolved as Boolean
bIsInvolved = m_BERCTRL.GetPortInvolved(1)
```

Related functions and methods

"TermInvolved" on page 52

PropertiesTitle

ActiveX syntax

```
Object.PropertiesTitle = [sTitle]
```

For Visual C:

```
Object.SetPropertiesTitle(sTitle)
sTitle = Object.GetPropertiesTitle()
```

NOTE This property is not available for the wrapper dll.

Description Sets/returns the title of the *Properties* dialog.

Parameter **sTitle** Specifies the title of the *Properties* dialog (data type: `String`).

Example To set the title of the *Properties* dialog to "`BER`":

```
m_BERCTRL.PropertiesTitle = "BER"
```

Related functions and methods *"ShowProperties" on page 50*

RepetitiveMeasTime

ActiveX syntax

```
Object.RepetitiveMeasTime = [MeasTime]
```

For Visual C:

```
MeasTime = Object.GetRepetitiveMeasTime()
Object.SetRepetitiveMeasTime(MeasTime)
```

Wrapper dll syntax

```
BERGetRepetitiveMeasTime(hMeasurement,
                         *MeasTime)

BERSetRepetitiveMeasTime(hMeasurement,
                         MeasTime)
```

Description

Sets/returns the measurement period when the measurement is running in repetitive mode. To set the repetitive mode, use *"RunMode" on page 47*.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

MeasTime Specifies the period in which the bit error rate measurement results are accumulated (data type: `Double`).

For the wrapper dll GET function, this parameter is a pointer.

Example

To set the repetitive time to 5 seconds:

```
m_BERCTRL.RepetitiveMeasTime = 5
```

Related functions and methods

"RepetitiveResyncBER" on page 48
"UseRepetitiveResyncMode" on page 57

RunMode

ActiveX syntax

```
Object.RunMode = [bRunMode]
```

For Visual C:

```
bRunMode = Object.GetRunMode()
Object.SetRunMode(bRunMode)
```

Wrapper dll syntax

```
BERGetRunMode(hMeasurement,
              *bRunMode)

BERSetRunMode(hMeasurement,
              bRunMode)
```

Description

Sets/returns the mode in which the measurement is running: single or repetitive mode.

- **Single** mode allows you to stop the measurement automatically:
  - After a specific time interval.

    See *"UseSingleMaxTimeMode" on page 58*.
  - After a specific number of errors occurred.

    See *"UseMaxError" on page 56*.
  - After a specific number of compared bits.

    See *"UseMaxComparedBits" on page 55*.
- **Repetitive** mode allows you:
  - To specify the measurement period.

    See *"RepetitiveMeasTime" on page 46*.
  - To enable resyncronisation after a specific bit error rate occurred.

    See *"UseRepetitiveResyncMode" on page 57*.
  - To create of a log file.

    See *"UseLogFile" on page 54*.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

bRunMode Indicates if the measurement runs in single or repetitive mode. The following constants (data type: `RUN_MODE`) are defined:

Constant	Description
`ModeSingle`	Single mode
`ModeRepetitive`	Repetitive mode

For the wrapper dll GET function, this parameter is a pointer.

Example

To set the measurement to run in single mode:

```
m_BERCTRL.RunMode = ModeSingle
```

RepetitiveResyncBER

ActiveX syntax

```
Object.RepetitiveResyncBER = [ResResyncBERyncBER]
```

For Visual C:

```
ResyncBER = Object.GetRepetitiveResyncBER()
Object.SetRepetitiveResyncBER(ResyncBER)
```

Wrapper dll syntax

```
BERGetRepetitiveResyncBER(hMeasurement,
                          *ResyncBER)

BERSetRepetitiveResyncBER(hMeasurement,
                          ResyncBER)
```

Description

Sets/returns the value for the bit error rate at which resynchronization is performed.

NOTE

This value can only be set, if both the repetitive run mode and the resynchronization mode are activated. To return/set these modes, use *"RunMode" on page 47* and *"UseRepetitiveResyncMode" on page 57* successively.

Parameters **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

ResyncBER Specifies the bit error rate at which resynchronization is performed (data type: `Double`). For the wrapper dll GET function, this parameter is a pointer.

Example To set the bit error rate to 10^{-6}:

```
m_BERCTRL.RepetitiveResyncBER = 1E-006
```

ShowLogFile

ActiveX syntax `Object.ShowLogFile()`

Wrapper dll syntax `BERShowLogFile(hMeasurement)`

Description Displays the log file.

Input parameter **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

Related functions and methods *"UseLogFile" on page 54*
"LogFileName" on page 41

ShowProperties

ActiveX syntax `Object.ShowProperties()`

NOTE This method is not available for the wrapper dll.

Description Displays the *Properties* dialog for the control.

Parameter None

Related functions and methods *"PropertiesTitle" on page 45*

SingleMaxTime

ActiveX syntax

```
Object.SingleMaxTime = [dMaxTime]
```

For Visual C:

```
dMaxTime = Object.GetSingleMaxTime()
Object.SetSingleMaxTime(dMaxTime)
```

Wrapper dll syntax

```
BERGetSingleMaxTime(hMeasurement,
                    *dMaxTime)

BERSetSingleMaxTime(hMeasurement,
                    dMaxTime)
```

Description Sets/returns the time period in which the measurement is running. When time passed, the measurement stops automatically.

NOTE This value can only be set, if both the single run mode and the single time mode are activated. To return/set these modes, use *"RunMode" on page 47* and *"UseSingleMaxTimeMode" on page 58* successively.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

dMaxTime Specifies the time after which the measurement stops automatically (data type: `Double`). For the wrapper dll GET function, this parameter is a pointer.

Example To set the time to 5 seconds:

```
m_BERCTRL.SingleMaxTime = 5
```

TermInvolved

ActiveX syntax

```
boolean = Object.GetTermInvolved(nPortID,
                                 nTermID)

Object.SetTermInvolved(nPortID,
                       nTermID,
                       boolean)
```

Wrapper dll syntax

```
BERGetTermInvolved(hMeasurement,
                   nPortID,
                   nTermID,
                   *boolean)

BERSetTermInvolved(hMeasurement,
                   nPortID,
                   nTermID,
                   boolean)
```

Description

Sets/returns whether a port is involved in the measurement.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nPortID A port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

nTermID A terminal is addressed by the terminal number (data type: `Integer`). This is an index starting at 1 for each port.

boolean Indicates if the port is involved in the measurement. The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Terminal is involved in the measurement.
`False`	Terminal is not involved in the measurement.

For the wrapper dll GET function, this parameter is a pointer.

Example

To set port 1, terminal 1 not involved in the measurement:

```
m_BERCTRL.SetTermInvolved(1, 1, False)
```

To get whether terminal 1 of port 1 is involved in the measurement:

```
Dim bIsInvolved as Boolean
bIsInvolved = m_BERCTRL.GetTermInvolved(1, 1)
```

Related functions and methods

"PortInvolved" on page 44

UseBERThreshold

ActiveX syntax

```
Object.UseBERThreshold = [boolean]
```

For Visual C:

```
Object.SetUseBERThresholde(boolean)
boolean = Object.GetUseBERThreshold()
```

Wrapper dll syntax

```
BERSetUseBERThreshold(hMeasurement,
                      boolean)

BERGetUseBERThreshold(hMeasurement,
                      *boolean)
```

Description

Sets/returns whether the threshold as pass/fail criterion for the BER measurement is activated. To specify the value for the threshold, use *"BERThreshold" on page 33*.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

boolean Returns whether the pass/fail criterion is activated (data type: `Boolean`):

Constant	Description
`True`	The pass/fail criterion is activated.
`False`	The pass/fail criterion is not activated.

For the wrapper dll GET function, this parameter is a pointer.

UseLogFile

ActiveX syntax

```
Object.UseLogFile = [boolean]
```

For Visual C:

```
Object.SetUseLogFile(boolean)
boolean = Object.GetUseLogFile()
```

Wrapper dll syntax

```
BERSetUseLogFile(hMeasurement,
                 boolean)

BERGetUseLogFile(hMeasurement,
                 *boolean)
```

Description Sets/returns whether a log file is created.

NOTE This is available only if the run mode is set to repetitive mode. To set the repetitive mode, use *"RunMode" on page 47*.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

boolean Returns whether the log file is created (data type: `Boolean`):

Constant	Description
`True`	A log file is created.
`False`	A log file is not created.

For the wrapper dll GET function, this parameter is a pointer.

Related functions and methods

"ShowLogFile" on page 50
"LogFileName" on page 41

UseMaxComparedBits

ActiveX syntax

```
Object.UseMaxComparedBits = [boolean]
```

For Visual C:

```
Object.SetUseMaxComparedBits(boolean)
boolean = Object.GetUseMaxComparedBits()
```

Wrapper dll syntax

```
BERGetUseMaxComparedBits(hMeasurement,
                         *boolean)
BERSetUseMaxComparedBits(hMeasurement,
                          boolean)
```

Description Sets/returns whether the measurements stops automatically after a specific number of compared bits. The number of compared bits can be set with *"MaxComparedBits" on page 42*.

NOTE This stop criterion is available only if the run mode is set to single mode. To set the single mode, use *"RunMode" on page 47*.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

boolean The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	The measurement stops after a specific number of compared bits.
`False`	The measurement does not stop after a specific number of compared bits.

For the wrapper dll GET function, this parameter is a pointer.

Example To disable the `MaxComparedBits` option:

```
m_BERCTRL.UseMaxComparedBits = False
```

UseMaxError

ActiveX syntax

```
Object.UseMaxError = [boolean]
```

For Visual C:

```
Object.SetUseMaxError(boolean)
boolean = Object.GetUseMaxError()
```

Wrapper dll syntax

```
BERGetUseMaxError(hMeasurement,
                  *boolean)
BERSetUseMaxError(hMeasurement,
                  boolean)
```

Description Sets/returns whether the measurements stops automatically after a specific number of errors occurred. The number of errors can be set with *"MaxError" on page 43*.

NOTE The stop criterion is available only if the run mode is set to single mode. To set the single mode, use *"RunMode" on page 47*.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

boolean The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Turn on the checking of max errors to stop the measurement.
`False`	Turn off the checking of max errors.

For the wrapper dll GET function, this parameter is a pointer.

Example To disable the `MaxError` option:

```
m_BERCTRL.UseMaxError = False
```

UseRepetitiveResyncMode

ActiveX syntax

```
Object.UseRepetitiveResyncMode = [boolean]
```

For Visual C:

```
boolean = Object.GetUseRepetitiveResyncMode()
Object.SetUseRepetitiveResyncMode(boolean)
```

Wrapper dll syntax

```
BERGetUseRepetitiveResyncMode(hMeasurement,
                              *boolean)

BERSetUseRepetitiveResyncMode(hMeasurement,
                              boolean)
```

Description Sets/returns whether resynchronization is performed after a specific bit error rate has been measured. To specify this bit error rate, use *"RepetitiveResyncBER" on page 48*.

NOTE This stop criterion is available only if the run mode is set to repetitive mode. To set the repetitive mode, use *"RunMode" on page 47*.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

boolean Indicates if the repetitive resynchronization is activated:

Constant	Description
`True`	Resynchronization is activated.
`False`	Resynchronization is not activated.

For the wrapper dll GET function, this parameter is a pointer.

Example To activate the resynchronization mode:

```
m_BERCTRL.UseRepetitiveResyncMode = True
```

Related functions and methods *"RunMode" on page 47*

UseSingleMaxTimeMode

ActiveX syntax

```
Object.UseSingleMaxTimeMode = [boolean]
```

For Visual C:

```
boolean = Object.GetUseSingleMaxTimeMode()
Object.SetUseSingleMaxTimeMode(boolean)
```

Wrapper dll syntax

```
BERGetUseSingleMaxTimeMode(hMeasurement,
                           *boolean)

BERSetUseSingleMaxTimeMode(hMeasurement,
                           boolean)
```

Description

Sets/returns whether the measurement stops after a specified time. To specify this time, use *"SingleMaxTime" on page 51*.

NOTE

This stop criterion is available only if the run mode is set to single mode. To set the single mode, use *"RunMode" on page 47*.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

boolean Indicates if the repetitive resynchronization is activated:

Constant	Description
`True`	The measurement stops after the specified time.
`False`	The measurement does not stop after the specified time.

For the wrapper dll GET function, this parameter is a pointer.

Example

To activate the stop criterion "time":

```
m_BERCTRL.UseSingleMaxTimeMode = True
```

Related functions and methods

"RunMode" on page 47

Running the Measurement

The following table gives an overview on the methods, events and properties available to run the measurement:

Purpose	Refer to...
To load the measurement settings to the firmware server.	*"Download" on page 59*
To start a measurement run.	*"Run" on page 61*
To get the status of a measurement.	*"MeasState" on page 60*
To ensure a synchronous run on several systems.	*"SynchronousRun" on page 62*
To stop a measurement run.	*"Stop" on page 61*

Download

ActiveX syntax `Object.Download()`

Wrapper dll syntax `BERMeasureDownload(hMeasurement)`

Description Ensures that all sequences are downloaded to the firmware server, so that a subsequent run has repeatable runtime behavior.

Input parameter **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

Related functions and methods *"Run" on page 61*

MeasState

ActiveX syntax

```
sState = Object.MeasState
```

For Visual C:

```
sState = Object.GetMeasState()
```

Wrapper dll syntax

```
BERMeasureState(hMeasurement,
                bufferSize,
                *sState)
```

Description

Returns the status of the measurement object. This property is read-only.

Output parameter

sState The following measurement states (data type: `String`) are defined:

Constant	Description
`RUNN`	Measurement is running.
`SYNC`	Measurement is synchronizing.
`PROG`	Measurement is ready.
`HALT`	Measurement is prepared to run but the clocks are not ready and therefore, the measurement is waiting for an external trigger to start.

Input parameter

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

Example

To check the state of the measurement:

```
Dim sState as String
sState = m_BERCTRL.MeasState
```

Related functions and methods

"Run" on page 61

Run

ActiveX syntax

```
Object.Run()
```

Wrapper dll syntax

```
BERMeasureRun(hMeasurement)
```

Description Starts the measurement and ensures that the systems are started in the specified order. Necessary sequence downloads are performed before so that the time between starting system 1 and system 2 is always the specified delay.

Input parameter **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

Related functions and methods

"Stop" on page 61
"SynchronousRun" on page 62

Stop

ActiveX syntax

```
Object.Stop()
```

Wrapper dll syntax

```
BERMeasureStop(hMeasurement)
```

Description Stops the measurement.

Input parameter **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

Related functions and methods

"Run" on page 61

SynchronousRun

ActiveX syntax `Object.SynchronousRun()`

NOTE This method is not available for the wrapper dll.

The wrapper dll function is called Run.

Description Starts the measurement but does not return until the measurement is completed. It ensures that the systems are started in the specified order; necessary sequence downloads are performed before, so that the specified delay between starting systems 1 and 2 is always used.

Input parameter **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

Related functions and methods *"Run" on page 61*

Handling Events and Callbacks

The following table gives an overview on the methods, events and properties available to handle events and callbacks

Purpose	Refer to...
To indicate if result data is available.	*"OnMeasDataAvailable" on page 63*
To indicate if the measurement run is complete.	*"OnMeasurementComplete" on page 64*
To indicate that a transition in the measurement status occured.	*"OnMeasurementState" on page 65*
To register a callback function for certain events.	*"SetMeasEventsCallback" on page 66*

OnMeasDataAvailable

ActiveX syntax

```
Private Sub Object_OnMeasDataAvailable(lItemCount as Long)
```

NOTE This event is not available for the wrapper dll.

Description Returns the number of data items that are currently available in the measurement object. The returned number can be used to allocate the required buffer size for requesting measured data or to check whether data is available or not.

Input parameter **lItemCount** A long value that indicates the number of data points available.

Example To get the number of data points available and display the value in a message box:

```
Private Sub MeasureBER1_OnMeasDataAvailable(lItemCount as Long)
   Dim msg as string
   msg = "Number of items:"+ lItemCount
   Msgbox msg
End Sub
```

Related functions and methods

"OnMeasurementComplete" on page 64
"OnMeasurementState" on page 65

OnMeasurementComplete

ActiveX syntax

```
Private Sub Object_OnMeasurementComplete(ByVal lStatus As Long)
```

NOTE This event is not available for the wrapper dll.

Description Returns the measurement status upon completion of the measurement.

Input parameter **lStatus** A long value that indicates that the measurement run has been completed.

Example To check whether the measurement run is finished and display the fact in a message box:

```
Private Sub MeasureBER1_OnMeasurementComplete(
                                        ByVal lStatus As Long)
   MsgBox "Measurement is complete"
End Sub
```

Related functions and methods

"OnMeasDataAvailable" on page 63
"OnMeasurementState" on page 65

OnMeasurementState

ActiveX syntax `Private Sub Object_OnMeasurementState(eMeasState As Long)`

NOTE This event is not available for the wrapper dll.

Description Returns the measurement status when a transition in the measurement status occurs.

Input parameter **eMeasState** Indicates the measurement status. Possible values (data type: `Long`) are:

Constant	Value	Description
`StateProg`	1	Measurement data is transferred.
`StateRunning`	2	The measurement is running.
`StateComplete`	3	The measurement run has been completed.
`StateAbort`	4	The measurement run has been aborted.
`StateError`	5	An error occurred.
`StateSync`	6	The system is synchronizing.
`StateHalt`	7	The system is in halted state, waiting for external start.
`StateRunPulse`	8	Heartbeat state.

For further information on the states, refer to *Handle Events and Callbacks* in the *Measurement Software Programming Guide*.

Example To check the measurement status and display a message box when the measurement is complete:

```
Private Sub MeasureBER1_OnMeasurementState
                              (ByVal lStatus as Long)
' This event is called when the measurement state changes
Select Case lStatus
   Case 1
   ' Data transferred
   Case 2
   ' Measurement running
   Case 3
   ' Measurement complete
      MsgBox "The measurement is complete."
   Case 4
   ' Measurement aborted
   Case 5
   ' Error occurred
   Case 6
   ' System synchronizing
   Case 7
```

```
   ' System waiting for start
   Case 8
   ' Heartbeat state
End Select
End Sub
```

Related functions and methods

"OnMeasDataAvailable" on page 63
"OnMeasurementComplete" on page 64

SetMeasEventsCallback

NOTE This function is only available when using the wrapper dll.

Wrapper dll syntax

```
BERSetMeasEventsCallback(hMeasurement,
                         lpMeasEventsFunc,
                         lParamUserCBData)
```

Description

To register a callback function which is called for certain events. For further information, refer to *Handle Events and Callbacks* in the *Measurement Software Programming Guide*.

Parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

lpMeasEventsFunc Callback function, for example:

```
ViStatus CALLBACK MeasEventsProc(ViSession hMeasurement,
                                 LPARAM lUserData,
                                 LONG lEventId,
                                 LPARAM lParam)
```

lParamUserCBData Measurement information structure.

Error Handling

The following table gives an overview on the methods, events and properties available to handle errors:

Purpose	Refer to...
To get the most recent error number from the firmware server.	*"GetLastMeasError" on page 67*
To specify if errors are to be reported when running a measurement.	*"SilentMode" on page 68*

GetLastMeasError

ActiveX syntax

```
sError = Object.GetLastMeasError(lError)
```

Wrapper dll syntax

```
BERGetLastMeasError(hMeasurement,
                    *lError,
                    sError,
                    bufferSize)
```

Description Returns the last measurement error description from the firmware server.

Output parameters

sError Error description (data type: `String`).

Input parameter

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

lError Error number (data type: `Long`). For the wrapper dll GET function, this parameter is a pointer.

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`)

Related functions and methods *"SilentMode" on page 68*

SilentMode

ActiveX syntax

```
Object.SilentMode = [boolean]
```

For Visual C:

```
Object.SetSilentMode(boolean)
boolean = Object.GetSilentMode()
```

NOTE This property is not available for the wrapper dll.

Description Turns the display of error messages on and off.

Input parameter **boolean** The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Turn on the display of error messages.
`False`	Turn off the display of error messages (default setting).

Example To turn error messages on:

```
m_BERCTRL.SilentMode = True
```

Related functions and methods *"GetLastMeasError" on page 67*

Handling Measurement Results

The following table gives an overview on the methods, events and properties available to get and calculate measurement results:

Purpose	Refer to...
To get the number of data points available for a terminal.	*"DataAvailable" on page 70*
To get one single data point (result).	*"GetBERDataPoint" on page 71*
To get the raw measurement results.	*"GetMeasData" on page 74*
To get the calculated results for the measurement.	*"GetMeasCalculatedValue" on page 73*
To get the calculated results for one port.	*"GetPortCalculatedValue" on page 76*
To get the calculated results for one terminal.	*"GetTermCalculatedValue" on page 78*
To get the measurement period.	*"MeasPeriod" on page 80*
To reset all calculations for the measurement.	*"ResetMeasCalculations" on page 80*
To reset all calculations for one port.	*"ResetPortCalculations" on page 81*
To reset all calculations for one terminal.	*"ResetTermCalculations" on page 81*

DataAvailable

ActiveX syntax

```
Object.DataAvailable(nPortId,
                     nTerminalID,
                     lNumPoints)
```

Wrapper dll syntax

```
BERDataAvailable(hMeasurement,
                 nPortId,
                 nTerminalID,
                 lNumPoints)
```

Description

Returns the number of data points that are available for a specified port and terminal.

Output parameter

lNumPoints The returned value (data type: `long`) specifies the number of points that are available in the measurement.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nPortId A port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

nTerminalID A terminal is addressed by the terminal number (data type: `Integer`). This is an index starting at 1 for each port.

Example

To get the number of data points available for terminal 1 of port 1:

```
Dim nItems as Long
m_BERCTRL.DataAvailable(1, 1, nItems)
```

GetBERDataPoint

ActiveX syntax

```
Object.GetBERDataPoint(nPortID,
                       nTermID,
                       lDataIndex,
                       bResyncDone,
                       lIntervalNr,
                       dElapsedTime,
                       dComparedBits,
                       dErroneousBits,
                       dErroneousZeros,
                       dErroneousOnes)
```

Wrapper dll syntax

```
BERGetBERDataPoint(hMeasurement,
                   nPortID,
                   nTermID,
                   lDataIndex,
                   *bResyncDone,
                   *lIntervalNr,
                   *dElapsedTime,
                   *dComparedBits,
                   *dErroneousBits,
                   *dErroneousZeros,
                   *dErroneousOnes)
```

Description

Returns a single data point for a particular port and terminal ID. Use the `DataAvailable` method to get the number of data points.

Output parameters

dComparedBits Number of compared bits (data type: `Double`). For the wrapper dll GET function, this parameter is a pointer.

dErroneousBits Number of errors (data type: `Double`). For the wrapper dll GET function, this parameter is a pointer.

dErroneousZeros Number of errors from zeros (data type: `Double`). For the wrapper dll GET function, this parameter is a pointer.

dErroneousOnes Number of errors from ones (data type: `Double`). For the wrapper dll GET function, this parameter is a pointer.

bResyncDone Indicates if resynchronization has been done at the specified data point.

lIntervalNr Number of the data point. This is an index starting at 1.

dElapsedTime Indicates the elapsed time since the measurement started.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nPortID The port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

nTermID The terminal is addressed by the terminal number (data type: `Integer`). This is an index starting at 1 for each port.

lDataIndex Data index (data type: `Long`) starting at 0. Use the `DataAvailable` method to return the number of data points.

Example

To get the first data point for terminal 1 of port 1:

```
Dim dElapsedTime, dComparedBits, dErroneousBits, dErroneousZeros,
dErroneousOnes as Double
Dim lIntervalNr as Long
Dim bResyncDone as Boolean
m_BERCTRL.GetBERDataPoint(1, 1, 0, bResyncDone,
                          dElapsedTime, lIntervalNr,
                          dComparedBits, dErroneousBits,
                          dErroneousZeros, dErroneousOnes)
```

Related functions and methods

"GetPortCalculatedValue" on page 76
"GetTermCalculatedValue" on page 78
"GetMeasCalculatedValue" on page 73
"DataAvailable" on page 70

GetMeasCalculatedValue

ActiveX syntax

```
dValue = Object.GetMeasCalculatedValue(eAnalysisTerm,
                                       bValid)
```

Wrapper dll syntax

```
BERGetMeasCalculatedValue(hMeasurement,
                          eAnalysisTerm,
                          *bValid,
                          *dValue)
```

Description

Returns the measurement parameter for the designated analysis term.

Output parameter

dValue Returned value (data type: `Double`): Calculated value at the BER threshold for the designated analysis term. For the wrapper dll GET function, this parameter is a pointer.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

eAnalysisTerm The following constants (data type: `MEAS_ANALYSIS_TERM`) are defined:

Constant	Equivalent to parameter...
`MeasElaspedTime`	Total Elapsed Measurement Time
`MeasResyncs`	Total Number of Resynchronizations
`MeasNbrIntervals`	Total Number of Measurement Intervals

bValid Returns whether the value returned is valid for the measurement. There are several cases where the value may not be valid, for example, `MeasResync` is not valid when the measurement runs in single mode. The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Value is valid for measurement.
`False`	Value is not valid for measurement.

For the wrapper dll GET function, this parameter is a pointer.

Remarks

You have to check if `bValid` = `True` before reporting or using the returned value.

Example To get the calculated parameter `MeasElaspedTime`:

```
Dim dValue as Double
Dim bValid as boolean
dValue = m_BERCTRL.GetMeasCalculatedValue(MeasElaspedTime,
                                          bValid)
```

Related functions and methods

"GetPortCalculatedValue" on page 76
"GetTermCalculatedValue" on page 78

GetMeasData

ActiveX syntax

```
vData = Object.GetMeasData(nPortID,
                           nTermID,
                           lStartItem,
                           lItemCount,
                           lItemsRet)
```

Wrapper dll syntax

```
BERGetMeasData(hMeasurement,
               nPortID,
               nTermID,
               lStartItem,
               lItemCount,
               *lItemsRet,
               *vData)
```

Description Returns a variant that contains the *raw* data from the measurement for a designated port and terminal. The method allows you to enter a start index and the number of data points to be returned. It also returns the number of data points found.

Output parameters

vData Array (data type: `Variant`) of data. The index starts at 0.

Data Array	Description
`bResyncDone = vData(index, 0)`	Resynchronization indicator (data type:Boolean)
`lIntevalNr = vData (index, 1)`	Interval number (data type: Long)
dElapsedTime = `vData (index, 2)`	Elapsed time (data type:Double)
`nComparedBits = vData(index, 3)`	Number of compared bits (data type: `Double`)
`nErrors = vData(index, 4)`	Errors (data type: `Double`)
`nErrors0s = vData(index, 5)`	Errors from 0s (data type: `Double`)
`nErrors1s = vData(index, 6)`	Errors from 1s (data type: `Double`)

lItemsRet Number of data points (data type: `Long`) returned to the variant. You can request more data points than may be available. For the wrapper dll GET function, this parameter is a pointer.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nPortID The port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

nTermID The terminal is addressed by the terminal number (data type: `Integer`). This is an index starting at 1 for each port.

lStartItem Index of data points to be returned (data type: `Long`). The index starts at 0.

lItemCount Number of data points to be returned (data type: `Long`).

Example

To return a variant that contains 5 sets of data if `lItemRet` returns 5:

```
Dim vData as Variant
Dim lItemRet as Long
vData = m_BERCTRL.GetMeasData(1, 1, 0, 5, lItemRet)
```

Related functions and methods

"GetBERDataPoint" on page 71

GetPortCalculatedValue

ActiveX syntax

```
dValue = Object.GetPortCalculatedValue(nPortID,
                                       eAnalysisTerm,
                                       bValid)
```

Wrapper dll syntax

```
BERGetPortCalculatedValue(hMeasurement,
                          nPortID,
                          eAnalysisTerm,
                          *bValid,
                          *dValue)
```

Description

Returns the measurement parameter for the designated port and analysis term.

Output parameter

dValue Returned value (data type: `Double`): Calculated value at the BER threshold for the designated analysis term. For the wrapper dll GET function, this parameter is a pointer.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nPortID A port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

eAnalysisTerm The following constants (data type: `ANALYSIS_TERM`) are defined:

Constant	Equivalent to parameter...
`NbrResyncs`	Number of Resynchronizations
`ActualBERAll`	Actual Bit Error Rate BER (All Errors, the default) is defined as the total number of errors, divided by the total number of compared bits.
`ActualBER0`	Actual Bit Error Rate for expected 0s In case of 0s Error, the BER is the number of errors if a 0 is expected, divided by the total number of compared bits.
`ActualBER1`	Actual Bit Error Rate for expected 1s In case of 1s Error, the BER is the number of errors if a 1 is expected, divided by the total number of compared bits.
`ActualComparedBitsAll`	Actual Compared Bits
`ActualNbrErrorsAll`	Actual Numbers of Errors
`ActualNbrErrors0`	Actual Number of Errors from expected 0s

Constant	Equivalent to parameter...
ActualNbrErrors1	Actual Number of Errors from expected 1s
AccumBERAll	Accumulative BER
AccumBER0	Accumulated Bit Error Rate for expected 0s.
AccumBER1	Accumulated Bit Error Rate for expected 1s.
AccumComparedBitsAll	Accumulated Compared Bits
AccumNbrErrorsAll	Accumulated Numbers of Errors
AccumNbrErrors0	Accumulated Numbers of Errors from expected 0s
AccumNbrErrors1	Accumulated Numbers of Errors from expected 1s
AllAnalysis	All Analysis Fields

For more information on the parameters, refer to *Explanation of the Measured Parameters* in the *Bit Error Rate Measurement User Guide*.

bValid Returns whether the value returned is valid for the measurement (data type: `Boolean`). The following constants (data type: `Boolean`) are defined:

Constant	Description
True	Value is valid for measurement.
False	Value is not valid for measurement.

For the wrapper dll GET function, this parameter is a pointer.

Remarks You have to check if `bValid = True` before reporting or using the returned value.

Example To get the calculated parameter `ActualBER0` for port 1:

```
Dim dValue as Double
Dim bValid as boolean
dValue = m_BERCTRL.GetPortCalculatedValue(1, ActualBER0,
                                          bValid)
```

Related functions and methods *"GetMeasCalculatedValue" on page 73*

GetTermCalculatedValue

ActiveX syntax

```
dValue = Object.GetTermCalculatedValue(nPortID,
                                       nTermID,
                                       eAnalysisTerm,
                                       bValid)
```

Wrapper dll syntax

```
BERGetTermCalculatedValue(hMeasurement,
                          nPortID,
                          nTermID,
                          eAnalysisTerm,
                          *bValid,
                          *dValue)
```

Description

Returns the measurement parameter for the designated port and terminal and analysis term. The parameter is calculated at the BER threshold value.

Output parameter

dValue Returned value (data type: `Double`): Calculated value at the BER threshold for the designated analysis term. For the wrapper dll GET function, this parameter is a pointer.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nPortID A port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

nTermID A terminal is addressed by the terminal number (data type: `Integer`). This is an index starting at 1 for each port.

eAnalysisTerm The following constants (data type: `ANALYSIS_TERM`) are defined:

Constant	Equivalent to parameter...
`NbrResyncs`	Number of Resynchronizations
`ActualBERAll`	Actual Bit Error Rate BER (All Errors, the default) is defined as the total number of errors, divided by the total number of compared bits.
`ActualBER0`	Actual Bit Error Rate for expected 0s In case of 0s Error, the BER is the number of errors if a 0 is expected, divided by the total number of compared bits.
`ActualBER1`	Actual Bit Error Rate for expected 1s In case of 1s Error, the BER is the number of errors if a 1 is expected, divided by the total number of compared bits.
`ActualComparedBitsAll`	Actual Compared Bits

Constant	Equivalent to parameter...
ActualNbrErrorsAll	Actual Numbers of Errors
ActualNbrErrors0	Actual Number of Errors from expected 0s
ActualNbrErrors1	Actual Number of Errors from expected 1s
AccumBERAll	Accumulative BER
AccumBER0	Accumulated Bit Error Rate for expected 0s
AccumBER1	Accumulated Bit Error Rate for expected 1s
AccumComparedBitsAll	Accumulated Compared Bits
AccumNbrErrorsAll	Accumulated Numbers of Errors
AccumNbrErrors0	Accumulated Numbers of Errors from expected 0s
AccumNbrErrors1	Accumulated Numbers of Errors from expected 1s
AllAnalysis	All Analysis Fields

For more information on the parameters, refer to *Explanation of the Measured Parameters* in the *Bit Error Rate Measurement User Guide*.

bValid Returns whether the value returned is valid for the measurement. The following constants (data type: `Boolean`) are defined:

Constant	Description
True	Value is valid for measurement.
False	Value is not valid for measurement.

For the wrapper dll GET function, this parameter is a pointer.

Remarks You have to check if `bValid` = `True` before reporting or using the returned value.

Example To get the calculated parameter `ActualNbrErrors0` for terminal 3 of port 1:

```
Dim dValue as Double
Dim bValid as boolean
dValue = m_BERCTRL.GetTermCalculatedValue(1, 3, ActualNbrErrors0,
                                          bValid)
```

Related functions and methods

"GetMeasCalculatedValue" on page 73
"GetPortCalculatedValue" on page 76

MeasPeriod

ActiveX syntax

```
dPeriod = Object.MeasPeriod
```

For Visual C:

```
dPeriod = Object.GetMeasPeriod()
```

Wrapper dll syntax

```
BERGetMeasPeriod(hMeasurement,
                 *dPeriod)
```

Description Returns the measurement period for the unit interval. This is needed to convert from unit interval to seconds and vice versa. This property is read-only.

Output parameter **dPeriod** Measurement period in seconds (data type: `Double`). For the wrapper dll GET function, this parameter is a pointer.

Input parameter **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

Example To get the measurement period:

```
Dim dPeriod as Double
dPeriod = m_BERCTRL.MeasPeriod
```

ResetMeasCalculations

ActiveX syntax

```
Object.ResetMeasCalculations()
```

Wrapper dll syntax

```
BERResetMeasCalculations(hMeasurement)
```

Description Resets all calculations for all ports and all terminals.

Input parameter **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

Related functions and methods *"GetMeasCalculatedValue" on page 73*

ResetPortCalculations

ActiveX syntax

```
Object.ResetPortCalculations(nPortID)
```

Wrapper dll syntax

```
BERResetPortCalculations(hMeasurement,
                         nPortID)
```

Description Resets all calculations for the designated port.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nPortID The port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

Related functions and methods *"GetPortCalculatedValue" on page 76*

ResetTermCalculations

ActiveX syntax

```
Object.ResetTermCalculations(nPortID,
                             nTermID)
```

Wrapper dll syntax

```
BERResetTermCalculations(hMeasurement,
                         nPortID,
                         nTermID)
```

Description Resets all calculations for the designated port and terminal.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nPortID The port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

nTermID The terminal is addressed by the terminal number (data type: `Integer`). This is an index starting at 1 for each port.

Related functions and methods *"GetTermCalculatedValue" on page 78*

Pass/Fail Functions

The following sections show the functions used to set and evaluate pass/fail decisions.

The bit error rate measurement provides only one Pass/Fail criterion: the BER threshold. This Pass/Fail criterion can be activated or deactivated with UseBERThreshold.

The following table gives an overview on the methods, events and properties available to check if a measurement, port or terminal has passed or failed:

Purpose	Refer to...
To check if a parameter has passed for the complete measurement.	*"GetMeasPassValue" on page 82*
To check if a parameter has passed for a given port.	*"GetPortPassValue" on page 85*
To check if a parameter has passed for a given terminal.	*"GetTermPassValue" on page 87*

GetMeasPassValue

NOTE This function is only available when using the wrapper dll.

Wrapper dll syntax

```
BERGetMeasPassValue(hMeasurement,
                    nAnalysisTerm,
                    *pIsPass)
```

Description Returns whether a measurement parameter has passed or failed. The criteria are set by separate pass/fail methods.

Output parameter

pIsPass Returns whether the designated measurement parameter passed or failed. The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Measurement passed.
`False`	Measurement failed.

For the wrapper dll GET function, this parameter is a pointer.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nAnalysisTerm The following constants (data type: `ANALYSIS_TERM`) are defined:

Constant	Equivalent to parameter...
`NbrResyncs`	Number of Resynchronizations
`ActualBERAll`	Actual Bit Error Rate BER (All Errors, the default) is defined as the total number of errors, divided by the total number of compared bits.
`ActualBER0`	Actual Bit Error Rate for expected 0s In case of 0s Error, the BER is the number of errors if a 0 is expected, divided by the total number of compared bits.
`ActualBER1`	Actual Bit Error Rate for expected 1s In case of 1s Error, the BER is the number of errors if a 1 is expected, divided by the total number of compared bits.
`ActualComparedBitsAll`	Actual Compared Bits
`ActualNbrErrorsAll`	Actual Numbers of Errors
`ActualNbrErrors0`	Actual Number of Errors from expected 0s
`ActualNbrErrors1`	Actual Number of Errors from expected 1s
`AccumBERAll`	Accumulative BER
`AccumBER0`	Accumulated Bit Error Rate for expected 0s
`AccumBER1`	Accumulated Bit Error Rate for expected 1s
`AccumComparedBitsAll`	Accumulated Compared Bits
`AccumNbrErrorsAll`	Accumulated Numbers of Errors
`AccumNbrErrors0`	Accumulated Numbers of Errors from expected 0s
`AccumNbrErrors1`	Accumulated Numbers of Errors from expected 1s
`AllAnalysis`	All Analysis Fields

For more information on the parameters, refer to *Explanation of the Measured Parameters* in the *Bit Error Rate Measurement User Guide*.

Remarks The measurement parameter will always pass if the pass/fail criterion has been turned off using the property `UseBERThreshold`. For example, if `UseBERThreshold = False`, `GetPortPassValue` will always return `True` regardless of the criterion.

Related functions and methods

"GetPortPassValue" on page 85
"GetTermPassValue" on page 87

GetPortPassValue

ActiveX syntax

```
bPass = Object.GetPortPassValue(nPortID,
                                eAnalysisTerm,
                                bValid)
```

Wrapper dll syntax

```
BERGetPortPassValue(hMeasurement,
                    nPortID,
                    eAnalysisTerm,
                    *bValid,
                    *bPass)
```

Description

Returns whether a measurement parameter has passed or failed the pass/fail criterion. The criteria are set by separate pass/fail methods.

Output parameter

bPass Returns whether the designated measurement parameter passed or failed. The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Port measurement value passed.
`False`	Port measurement value failed.

For the wrapper dll GET function, this parameter is a pointer.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nPortID A port is addressed by the port number (data type: `Integer`). This is an index starting at 1.

eAnalysisTerm The following constants (data type: `ANALYSIS_TERM`) are defined:

Constant	Equivalent to parameter...
`NbrResyncs`	Number of Resynchronizations
`ActualBERAll`	Actual Bit Error Rate BER (All Errors, the default) is defined as the total number of errors, divided by the total number of compared bits.
`ActualBER0`	Actual Bit Error Rate for expected 0s In case of 0s Error, the BER is the number of errors if a 0 is expected, divided by the total number of compared bits.
`ActualBER1`	Actual Bit Error Rate for expected 1s In case of 1s Error, the BER is the number of errors if a 1 is expected, divided by the total number of compared bits.
`ActualComparedBitsAll`	Actual Compared Bits

Constant	Equivalent to parameter...
ActualNbrErrorsAll	Actual Numbers of Errors
ActualNbrErrors0	Actual Number of Errors from expected 0s
ActualNbrErrors1	Actual Number of Errors from expected 1s
AccumBERAll	Accumulative BER
AccumBER0	Accumulated Bit Error Rate for expected 0s
AccumBER1	Accumulated Bit Error Rate for expected 1s
AccumComparedBitsAll	Accumulated Compared Bits
AccumNbrErrorsAll	Accumulated Numbers of Errors
AccumNbrErrors0	Accumulated Numbers of Errors from expected 0s
AccumNbrErrors1	Accumulated Numbers of Errors from expected 1s
AllAnalysis	All Analysis Fields

For more information on the parameters, refer to *Explanation of the Measured Parameters* in the *Bit Error Rate Measurement User Guide*.

bValid Returns whether the value returned is valid for the measurement. The following constants (data type: `Boolean`) are defined:

Constant	Description
True	Data is valid for the measurement.
False	Data is not valid for the measurement.

For the wrapper dll GET function, this parameter is a pointer.

Remarks

The parameter will always pass if the pass/fail criterion has been turned off using the property `UseBERThreshold`. If `UseBERThreshold = False`, `GetPortPassValue` will always return `True` regardless of the criterion.

You have to check if `bValid = True` before reporting or using the returned value.

Example

To check the pass/fail criterion `ActualBER0` for port 1:

```
Dim bPass, bValid as Boolean
bPass = m_BERCTRL.GetPortPassValue(1, ActualBER0, bValid)
```

Related functions and methods

"GetMeasPassValue" on page 82
"GetTermPassValue" on page 87

GetTermPassValue

ActiveX syntax

```
bPass =  Object.GetTermPassValue(nPortID,
                                 nTermID,
                                 eAnalysisTerm,
                                 bValid)
```

Wrapper dll syntax

```
BERGetTermPassValue(hMeasurement,
                    nPortID,
                    nTermID,
                    eAnalysisTerm,
                    *bValid,
                    *bPass)
```

Description

Returns whether the measured value of the object passed or failed. The criteria are set by separate pass/fail methods.

For more information on how to set pass/fail criteria, refer to *How to Set Pass/Fail Criteria* in the *Bit Error Measurement User Guide*.

NOTE

Before `GetTermPassValue` can return a valid result, the following method have to be called:

- `UseBERThreshold(True);`

  Turns on the pass/fail check.

- `BERThreshold(dValue);`

  Sets the pass/fail criterion for the measurement parameter *BER Threshold*.

If these values are not correctly set, `GetTermPassValue` always returns `TRUE`.

Output parameters

bPass Returns whether the designated measurement parameter passed or failed. The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Terminal measurement value passed.
`False`	Terminal measurement value failed.

For the wrapper dll GET function, this parameter is a pointer.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

nPortID A port is addressed by the port number (data type: `Integer`). This is an index starting at 1 for each clock group.

nTermID A terminal is addressed by the terminal number (data type: `Integer`). This is an index starting at 1 for each port.

eAnalysisTerm Defines the parameter to be checked. The following constants (data type: `ANALYSIS_TERM`) are defined:

Constant	Equivalent to parameter...
`NbrResyncs`	Number of Resynchronizations
`ActualBERAll`	Actual Bit Error Rate BER (All Errors, the default) is defined as the total number of errors, divided by the total number of compared bits.
`ActualBER0`	Actual Bit Error Rate for expected 0s In case of 0s Error, the BER is the number of errors if a 0 is expected, divided by the total number of compared bits.
`ActualBER1`	Actual Bit Error Rate for expected 1s In case of 1s Error, the BER is the number of errors if a 1 is expected, divided by the total number of compared bits.
`ActualComparedBitsAll`	Actual Compared Bits
`ActualNbrErrorsAll`	Actual Numbers of Errors
`ActualNbrErrors0`	Actual Number of Errors from expected 0s
`ActualNbrErrors1`	Actual Number of Errors from expected 1s
`AccumBERAll`	Accumulative BER
`AccumBER0`	Accumulated Bit Error Rate for expected 0s
`AccumBER1`	Accumulated Bit Error Rate for expected 1s
`AccumComparedBitsAll`	Accumulated Compared Bits
`AccumNbrErrorsAll`	Accumulated Numbers of Errors
`AccumNbrErrors0`	Accumulated Numbers of Errors from expected 0s
`AccumNbrErrors1`	Accumulated Numbers of Errors from expected 1s
`AllAnalysis`	All Analysis Fields

For more information on the parameters, refer to *Explanation of the Measured Parameters* in the *Bit Error Rate Measurement User Guide*.

bValid Returns (data type: `Boolean`) whether the value returned is valid for the measurement.

For the wrapper dll GET function, this parameter is a pointer.

Example To check the pass/fail criterion for the actual bit error rate for expected 1s for terminal 3 of port 1:

```
m_MeasureBER.SetUseBERThreshold(TRUE);
m_MeasureBER.SetBERThreshold(4E-3);
bPass1=m_MeasureBER.GetTermPassValue(1, 3, ActualBER1, &bBool);
```

Related functions and methods

"GetMeasPassValue" on page 82
"GetPortPassValue" on page 85

Copy/Paste Functions

The following table gives an overview on the methods, events and properties available to cut, copy and delete measurement results:

Purpose	Refer to...
To copy data to the clipboard.	*"CopyToClipboard" on page 91*
To cut data from the measurement and copy the data to the clipboard.	*"CutToClipboard" on page 92*
To delete data from the numerical view.	*"EditDelete" on page 93*
To determine whether a copy function call is possible.	*"IsCopyAvailable" on page 94*
To determine whether a cut function call is possible.	*"IsCutAvailable" on page 95*
To determine whether a delete function call is possible.	*"IsEditDeleteAvailable" on page 95*
To determine whether a paste function call is possible.	*"IsPasteAvailable" on page 96*
To insert data previously copied to the clipboard.	*"PasteFromClipboard" on page 97*

CopyToClipboard

ActiveX syntax `Object.CopyToClipboard(boolean)`

NOTE This method is not available for the wrapper dll.

Description Copies the measurement window and the data of the numerical view to the clipboard.

To get the information from the clipboard, use *Paste Special* and then select *Enhanced Metafile* to get the graphics, *Unformatted Text* to get the data stored in the grid as comma delimited ASCII text, *HTML Format* to get the grid in HTML format. *Paste* gets the data displayed in the grid in HTML format.

In addition, data is stored in the clipboard in a MUI proprietary format that can be returned using the `PasteFromClipboard` method.

Use the `IsCopyAvailable` method to determine if a copy operation can be performed before calling `CopyToClipboard`.

Input parameter **boolean** The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	The clipboard is cleared and the measurement window is copied to the clipboard. In addition, the measurement data in the numerical view is copied to the clipboard.
`False`	The clipboard is not cleared. However, the measurement window and measurement data are copied to the clipboard. Prior to this call, if there is other data formats in the clipboard, they will remain in the clipboard.

Example To clear the clipboard and copy measurement data to it:

```
m_BERCTRL.CopyToClipBoard(True)
```

Related functions and methods

"CutToClipboard" on page 92
"PasteFromClipboard" on page 97
"IsCopyAvailable" on page 94

CutToClipboard

ActiveX syntax `Object.CutToClipboard(boolean)`

NOTE This method is not available for the wrapper dll.

Description Cuts data from the measurement and copies the data into the clipboard. To get the information from the clipboard, use *Paste Special* and then select *Enhanced Metafile* to get the graphics, *Unformatted Text* to get the data stored in the grid as comma delimited ASCII text, *HTML Format* to get the grid in HTML format. Paste returns the data displayed in the grid as HTML.

In addition, data is stored in the clipboard in a proprietary format that can be returned using the `PasteFromClipboard` method.

Use the `IsCutAvailable` method to determine if a cut operation can be performed before calling `CutToClipboard`.

Input parameter **boolean** The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	The clipboard is cleared out and the measurement window is copied to the clipboard. In addition the measurement data in the grid is copied into the clipboard.
`False`	The clipboard is not cleared out, however the measurement window and measurement data are copied into the clipboard. Prior to this call, if there is other data formats in the clipboard, they will remain in the clipboard.

Example To clear the clipboard, remove data from the measurement and copy the data to the clipboard:

```
m_BERCTRL.CutToClipBoard(True)
```

Related functions and methods

"CopyToClipboard" on page 91
"IsCutAvailable" on page 95

EditDelete

ActiveX syntax `Object.EditDelete(boolean)`

NOTE This method is not available for the wrapper dll.

Description Deletes copied data in the grid. What is deleted depends on where the focus is. If there is no focus then nothing is deleted. If there is focus on the copied data line then this data will be deleted.

Use the `IsEditDeleteAvailable` method to determine if deleting is possible before calling `EditDelete`.

Input parameter **boolean** The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Deletes copied data.
`False`	Reserved for future use.

Example To delete copied data from the numerical view of the measurement:

```
m_BERCTRL.EditDelete(True)
```

Related functions and methods *"IsEditDeleteAvailable" on page 95*

IsCopyAvailable

ActiveX syntax `bCopy = Object.IsCopyAvailable()`

NOTE This method is not available for the wrapper dll.

Description Returns whether a copy function can be called.

Output parameter **bCopy** The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Copy is a valid operation to call
`False`	Copy is not a valid operation to call.

Example To check whether a copy operation is possible:

```
Dim bCopy as Boolean
bCopy = m_BERCTRL.IsCopyAvailable()
```

Related functions and methods *"IsCutAvailable" on page 95*

IsCutAvailable

ActiveX syntax `bCut = Object.IsCutAvailable()`

NOTE This method is not available for the wrapper dll.

Description Returns whether a cut function can be called.

Output parameter **bCut** The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Cut operation can be called.
`False`	Cut operation can not be called.

Example To check whether a cut operation is possible:

```
Dim bCut as Boolean
bCut = m_BERCTRL.IsCutAvailable()
```

Related functions and methods *"IsCopyAvailable" on page 94*

IsEditDeleteAvailable

ActiveX syntax `bDelete = Object.IsEditDeleteAvailable()`

NOTE This method is not available for the wrapper dll.

Description Returns whether a delete function can be called.

Output parameter **bDelete** The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Delete is a valid operation to call.
`False`	Delete is not a valid operation to call.

Example To check whether a delete operation is possible:

```
Dim bDelete as Boolean
bDelete = m_BERCTRL.IsEditDeleteAvailable()
```

IsPasteAvailable

ActiveX syntax `bPaste = Object.IsPasteAvailable()`

NOTE This method is not available for the wrapper dll.

Description Returns whether a paste function can be called. If there is anything on the clipboard, this function returns `True`.

Output parameter **bPaste** The following constants (data type: `Boolean`) will be returned:

Constant	Description
`True`	Paste operation can be called.
`False`	Paste operation can not be called.

Example To check whether a paste operation is possible:

```
Dim bPaste as Boolean
bPaste = m_BERCTRL.IsPasteAvailable()
```

Related functions and methods

"IsCutAvailable" on page 95
"IsCopyAvailable" on page 94

PasteFromClipboard

ActiveX syntax `Object.PasteFromClipboard(boolean)`

NOTE This method is not available for the wrapper dll.

Description Pastes information into the grid. What is pasted depends on what was selected during the copy operation:

- If there is no focus during the copy operation, the paste will copy all of the rows into the grid.
- If the focus was on the measurement line, the paste will copy all of the rows into the grid.
- If the focus was on an individual port, only that port will be pasted.
- If the focus was on an individual terminal, only the terminal will be pasted.

Before calling `PasteFromClipboard`, use `IsPasteAvailable` to determine if a paste operation can be performed.

Input parameter **boolean** The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	Copies the information from the clipboard and then clears the clipboard.
`False`	Copies the information from the clipboard. The information remains in the clipboard.

Related functions and methods

"CopyToClipboard" on page 91
"CutToClipboard" on page 92

Persistence

The following section shows the functions used to load/save measurements and to export data from the measurements.

Functions to Load/Save Measurements

The following table gives an overview on the methods, events and properties available to get and calculate measurement results:

Purpose	Refer to...
To load a stored measurement.	*"LoadMeasurement" on page 99*
To save a measurement.	*"SaveMeasurement" on page 99*

Export Data from Measurements

The following table gives an overview on the methods, events and properties available to export measurement results:

Purpose	Refer to...
To export data to the clipboard or to a file.	*"ExecuteExport" on page 100*
To set if the calculated results or the raw data will be exported.	*"ExportDataType" on page 101*
To set the delimiter for the data export.	*"ExportDelimiter" on page 102*
To set the file name the data will be exported to.	*"ExportFileName" on page 103*
To set the date format for export.	*"ExportLocale" on page 104*
To specify if the results will be exported to file or clipboard.	*"ExportToClipboard" on page 105*
To export only results for expected 0s.	*"ExportUse0s" on page 106*
To export only results for expected 1s.	*"ExportUse1s" on page 107*
To export results for expected 0s and expected 1s.	*"ExportUseAll1s0s" on page 108*
To indicate whether the data point was extrapolated.	*"ExportUseExtrapolatedFlag" on page 109*

SaveMeasurement

ActiveX syntax

```
Object.SaveMeasurement(FileName)
```

Wrapper dll syntax

```
BERSaveMeasurement(hMeasurement,
                   FileName)
```

Description Saves the measurement into a designated file.

Input parameters **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

FileName Full path and name of the stored measurement file (data type: `String`). The Measurement User Interface application stores these files with an extension of `.mcp`.

Example To save the measurement `BERTiming.mcp`:

```
m_BERCTRL.SaveMeasurement("C:\\Temp\\BERTiming.mcp")
```

LoadMeasurement

ActiveX syntax

```
Object.LoadMeasurement(FileName)
```

Wrapper dll syntax

```
BERLoadMeasurement(hMeasurement,
                   FileName)
```

Description Loads the measurement stored in the designated file into the control.

Input parameters **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

FileName Full path and name of the stored measurement file (data type: `String`). The Measurement User Interface application stores these files with the extension `.mcp`.

Example To load the measurement `BERTiming.mcp`:

```
m_BERCTRL.LoadMeasurement("C:\\Temp\\BERTiming.mcp")
```

ExecuteExport

ActiveX syntax `Object.ExecuteExport()`

Wrapper dll syntax `BERExecuteExport(hMeasurement)`

Description Exports the measurement data to the clipboard or to a file. The format of the export is determined by the Export properties: `ExportDataType`, `ExportLocale`, `ExportDelimiter`, `ExportFileName`, `ExportToClipboard`, `ExportUse0s`, `ExportUse1s`, `ExportUseAll1s0s`, and `ExportUseExtrapolatedFlag`.

Input parameter **hMeasurement** Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

NOTE The exported data is exported as raw data, meaning that the time value is reported in absolute seconds.

Related functions and methods

"ExportDataType" on page 101
"ExportLocale" on page 104
"ExportDelimiter" on page 102
"ExportFileName" on page 103
"ExportToClipboard" on page 105
"ExportUse0s" on page 106
"ExportUse1s" on page 107
"ExportUseAll1s0s" on page 108
"ExportUseExtrapolatedFlag" on page 109

ExportDataType

ActiveX syntax

```
Object.ExportDataType = [eExportData]
```

For Visual C:

```
Object.SetExportDataType(eExportData)
eExportData = Object.GetExportDataType()
```

Wrapper dll syntax

```
BERGetExportDataType(hMeasurement,
                     *eExportData)
BERSetExportDataType(hMeasurement,
                     eExportData)
```

Description

Determines what data will be exported to the clipboard or file.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

eExportData The following constants (data type: `EXPORT_DATA`) are defined:

Constant	Description
`BER`	Exports only the BER data.
`ALL`	Exports the BER, compared bits and errors data.

For the wrapper dll GET function, this parameter is a pointer.

Example

To export only the data of the bit error rate:

```
m_BERCTRL.ExportDataType = BER
```

Related functions and methods

"ExecuteExport" on page 100
"ExportFileName" on page 103
"ExportLocale" on page 104
"ExportToClipboard" on page 105
"ExportUse0s" on page 106
"ExportUse1s" on page 107
"ExportUseAll1s0s" on page 108
"ExportUseExtrapolatedFlag" on page 109

ExportDelimiter

ActiveX syntax

```
Object.ExportDelimiter = [sDelimiter]
```

For Visual C:

```
Object.SetExportDelimiter(sDelimiter)
sDelimiter = Object.GetExportDelimiter()
```

Wrapper dll syntax

```
BERGetExportDelimiter(hMeasurement,
                      bufferSize,
                      *sDelimiter)
BERSetExportDelimiter(hMeasurement,
                      sDelimiter)
```

Description

Sets the delimiter for export.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

sDelimiter The delimiter (data type: `String`) that will be used between data elements in the export. The delimiter can be a space, tab or any writable character, for example, "," or "~". For the wrapper dll GET function, this parameter is a pointer.

Example

To set the delimiter between the data to "^":

```
m_BERCTRL.ExportDelimiter = "^"
```

Related functions and methods

"ExportDataType" on page 101
"ExecuteExport" on page 100
"ExportFileName" on page 103
"ExportLocale" on page 104
"ExportToClipboard" on page 105
"ExportUse0s" on page 106
"ExportUse1s" on page 107
"ExportUseAll1s0s" on page 108
"ExportUseExtrapolatedFlag" on page 109

ExportFileName

ActiveX syntax

```
Object.ExportFileName = [sFileName]
```

For Visual C:

```
Object.SetExportFileName(sFileName)
sFileName = Object.GetExportFileName()
```

Wrapper dll syntax

```
BERGetExportFileName(hMeasurement,
                     bufferSize,
                     *sFileName)
BERSetExportFileName(hMeasurement,
                     sFileName)
```

Description

Sets the file name and the directory data will be exported to.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

sFileName Directory and name of the file that data will be exported to (data type: `String`). For the wrapper dll GET function, this parameter is a pointer.

Example

To export data to the file `"C:\Temp\export.txt"`:

```
m_BERCTRL.ExportFileName = "C:\\Temp\\export.txt"
```

Related functions and methods

“ExportDataType” on page 101
“ExecuteExport” on page 100
“ExportDelimiter” on page 102
“ExportLocale” on page 104
“ExportToClipboard” on page 105
“ExportUse0s” on page 106
“ExportUse1s” on page 107
“ExportUseAll1s0s” on page 108
“ExportUseExtrapolatedFlag” on page 109

ExportLocale

ActiveX syntax

```
Object.ExportLocale = [sLocale]
```

For Visual C:

```
Object.SetExportLocale(sLocale)
sLocale = Object.GetExportLocale()
```

Wrapper dll syntax

```
BERGetExportLocale(hMeasurement,
                   bufferSize,
                   *sLocale)
BERSetExportLocale(hMeasurement,
                   sLocale)
```

Description

Sets the language for the export which will define the date format and the default delimiter.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

bufferSize Only for the wrapper dll: Specifies the size of the data buffer for the returned data (data type: `ViInt32`).

sLocale Name of the language that the export format will default to. This will define the date format and the default delimiter (data type: `String`). For the wrapper dll GET function, this parameter is a pointer.

Related functions and methods

"ExecuteExport" on page 100
"ExportDataType" on page 101
"ExportDelimiter" on page 102
"ExportFileName" on page 103
"ExportToClipboard" on page 105
"ExportUse0s" on page 106
"ExportUse1s" on page 107
"ExportUseAll1s0s" on page 108
"ExportUseExtrapolatedFlag" on page 109

ExportToClipboard

ActiveX syntax

```
Object.ExportToClipboard = [boolean]
```

For Visual C:

```
Object.SetExportToClipboard(boolean)
boolean = Object.GetExportToClipboard()
```

Wrapper dll syntax

```
BERGetExportToClipboard(hMeasurement,
                        *boolean)
BERSetExportToClipboard(hMeasurement,
                        boolean)
```

Description

Sets whether the export will go to the clipboard or not.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

boolean The settings for boolean (data type: `Boolean`) are:

Constant	Description
`True`	Turns on the export to the clipboard.
`False`	Turns off the export to the clipboard (default setting). Data will be exported to file.

For the wrapper dll GET function, this parameter is a pointer.

Example

To export data to the clipboard:

```
m_BERCTRL.ExportToClipboard = True
```

Related functions and methods

"ExecuteExport" on page 100
"ExportDataType" on page 101
"ExportDelimiter" on page 102
"ExportFileName" on page 103
"ExportUse0s" on page 106
"ExportUse1s" on page 107
"ExportUseAll1s0s" on page 108
"ExportUseExtrapolatedFlag" on page 109

ExportUse0s

ActiveX syntax

```
Object.ExportUse0s = [boolean]
```

For Visual C:

```
Object.SetExportUse0s(boolean)
boolean = Object.GetExportUse0s()
```

Wrapper dll syntax

```
BERGetExportUse0s(hMeasurement,
                  *boolean)
BERSetExportUse0s(hMeasurement,
                  boolean)
```

Description

Sets whether the export will contain data elements for errors from expected 0s.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

boolean The settings for boolean (data type: `Boolean`) are:

Constant	Description
`True`	BER (calculated from errors from 0s) and the errors from expected 0s will be included in the export.
`False`	BER (calculated from errors from 0s) and errors from 0s will *not* be included in the export (default setting).

For the wrapper dll GET function, this parameter is a pointer.

Related functions and methods

"ExecuteExport" on page 100
"ExportDataType" on page 101
"ExportDelimiter" on page 102
"ExportFileName" on page 103
"ExportToClipboard" on page 105
"ExportUse1s" on page 107
"ExportUseAll1s0s" on page 108
"ExportUseExtrapolatedFlag" on page 109

ExportUse1s

ActiveX syntax

```
Object.ExportUse1s = [boolean]
```

For Visual C:

```
Object.SetExportUse1s(boolean)
boolean = Object.GetExportUse1s()
```

Wrapper dll syntax

```
BERGetExportUse1s(hMeasurement,
                  *boolean)
BERSetExportUse1s(hMeasurement,
                  boolean)
```

Description

Sets whether the export will contain data elements for errors from 1s.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

boolean The settings for boolean (data type: `Boolean`) are:

Constant	Description
`True`	The BER (calculated from errors from 1s) and errors from 1s will be included in the export.
`False`	The BER (calculated from errors from 1s) and errors from 1s will not be included in the export (default setting).

For the wrapper dll GET function, this parameter is a pointer.

Related functions and methods

"ExecuteExport" on page 100
"ExportDataType" on page 101
"ExportDelimiter" on page 102
"ExportFileName" on page 103
"ExportUse0s" on page 106
"ExportToClipboard" on page 105
"ExportUseAll1s0s" on page 108
"ExportUseExtrapolatedFlag" on page 109

ExportUseAll1s0s

ActiveX syntax

```
Object.ExportUseAll1s0s = [boolean]
```

For Visual C:

```
Object.SetExportUseAll1s0s(boolean)
boolean = Object.GetExportUseAll1s0s()
```

Wrapper dll syntax

```
BERGetExportUseAll1s0s(hMeasurement,
                       *boolean)
BERSetExportUseAll1s0s(hMeasurement,
                       boolean)
```

Description

Sets whether the export will contain data elements for all errors.

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

boolean The settings for boolean (data type: `Boolean`) are:

Constant	Description
`True`	Total BER and all errors will be included in the export (default setting).
`False`	Total BER and all errors will not be included in the export.

For the wrapper dll GET function, this parameter is a pointer.

Related functions and methods

"ExecuteExport" on page 100
"ExportDataType" on page 101
"ExportDelimiter" on page 102
"ExportFileName" on page 103
"ExportUse0s" on page 106
"ExportToClipboard" on page 105
"ExportUse1s" on page 107
"ExportUseExtrapolatedFlag" on page 109

ExportUseExtrapolatedFlag

ActiveX syntax

```
Object.ExportUseExtrapolatedFlag = [boolean]
```

For Visual C:

```
Object.SetExportUseExtrapolatedFlag(boolean)
boolean = Object.GetExportUseExtrapolatedFlag()
```

Wrapper dll syntax

```
BERGetExportUseExtrapolatedFlag(hMeasurement,
                                *boolean)
BERSetExportUseExtrapolatedFlag(hMeasurement,
                                boolean)
```

Description

Sets whether the export will contain the extrapolated flag indicating that the data point was not calculated but was extrapolated by the firmware server.

NOTE

The *Extrapolated Flag* is for future use. For the moment, it is ignored (always 0).

Input parameters

hMeasurement Only for the wrapper dll: Handle to the measurement created by `InitMeasurement` (data type: `ViSession`).

boolean The following constants (data type: `Boolean`) are defined:

Constant	Description
`True`	The extrapolated flag will be included in the export.
`False`	The extrapolated flag will not be included in the export (default setting).

For the wrapper dll GET function, this parameter is a pointer.

Related functions and methods

"ExecuteExport" on page 100
"ExportDataType" on page 101
"ExportDelimiter" on page 102
"ExportFileName" on page 103
"ExportUse0s" on page 106
"ExportToClipboard" on page 105
"ExportUseAll1s0s" on page 108
"ExportUse1s" on page 107

Functions for General Purposes

The following table gives an overview on the methods, events and properties available to set background and text color for the display:

Purpose	Refer to...
To set the background color of the graphical view.	*"BackColor" on page 111*
To set the text color of the graphical display.	*"ForeColor" on page 112*
To set/return the name and the path of the help file.	*"MeasHelpPath" on page 113*
To set the help id for the measurement window.	*"MeasureWinHelp" on page 114*

BackColor

ActiveX syntax

```
Object.BackColor = [nColor]
```

For Visual C:

```
Object.SetBackColor(nColor)
nColor = Object.GetBackColor()
```

NOTE This property is not available for the wrapper dll.

Description Sets/returns the color of the background of the graphical view.

Parameter **nColor** Sets the background color (data type: `integer`). The following table lists typical color values:

Color	RGB Values	nColor Value
White	255, 255, 255	16777215
Black	0, 0, 0	0
Gray	192, 192, 192	12632256
Dark gray	128, 128, 128	8421504
Red	255, 0, 0	255
Dark red	128, 0, 0	128
Yellow	255, 255, 0	65535
Dark yellow	128, 128, 0	32896
Green	0, 255, 0	65280
Dark green	0, 128, 0	32768
Cyan	0, 255, 255	16776960
Dark cyan	0, 128, 128	8421376
Blue	0, 0, 255	16711680
Dark blue	0, 0, 128	8388608
Magenta	255, 0, 255	16711935
Dark magenta	128, 0, 128	8388736

Example To set the background color of the graphical view to "Blue":

```
m_BERCTRL.BackColor = 16711680
```

Related functions and methods *"BackColor" on page 111*

ForeColor

ActiveX syntax

```
Object.ForeColor = [nColor]
```

For Visual C:

```
Object.SetForeColor(nColor)
nColor = Object.GetForeColor()
```

NOTE This property is not available for the wrapper dll.

Description Sets the color of the foreground (text) on the graph.

Input parameter **nColor** Sets the foreground color (data type: `integer`). The following table lists typical color values:

Color	RGB Values	nColor Value
White	255, 255, 255	16777215
Black	0, 0, 0	0
Gray	192, 192, 192	12632256
Dark gray	128, 128, 128	8421504
Red	255, 0, 0	255
Dark red	128, 0, 0	128
Yellow	255, 255, 0	65535
Dark yellow	128, 128, 0	32896
Green	0, 255, 0	65280
Dark green	0, 128, 0	32768
Cyan	0, 255, 255	16776960
Dark cyan	0, 128, 128	8421376
Blue	0, 0, 255	16711680
Dark blue	0, 0, 128	8388608
Magenta	255, 0, 255	16711935
Dark magenta	128, 0, 128	8388736

Example To set the text color of the graphical view to "Black":

```
m_BER.ForeColor = 0
```

Related functions and methods *"BackColor" on page 111*

MeasHelpPath

ActiveX syntax

```
Object.MeasHelpPath = [sHelpPath]
```

For Visual C:

```
Object.SetMeasHelpPath(sHelpPath)
sHelpPath = Object.GetMeasHelpPath()
```

NOTE This property is not available for the wrapper dll.

Description Sets/returns the default help file name.

Parameter **sHelpPath** Full path and name of help file (data type: `String`).

Example To set the path to the help file to "`C:\Temp\tmBERM.chm`":

```
m_BERCTRL.MeasHelpPath = "C:\\Temp\\tmBERM.chm"
```

MeasureWinHelp

ActiveX syntax

```
Object.MeasureWinHelp(lWndHandle,
                      sHelpPath,
                      lCommand,
                      lData)
```

NOTE This method is not available for the wrapper dll.

Input parameters

lWndHandle Window handle for the parent window (data type: `Long`).

sHelpPath Path and name of the controls help file (data type: `String`). The file must have a ".chm" extension. Can be obtained using the `MeasHelpPath` property.

lCommand Value should be set to 1 for context help support (data type: `Long`). Not supporting any other help at this time.

lData Default control help ID, 131172 (data type: `Long`).

Example

To call the online help with the default help id:

```
Dim helpstr as String
Dim helpID as Long
Dim frmMain as Form
helpstr = m_BERCTRL.MeasHelpPath
helpID = m_BERCTRL.DefaultHelpID
m_BERCTRL.MeasureWinHelp(frmMain.hWnd, helpstr, 1, helpID)
```

Related functions and methods

"MeasHelpPath" on page 113

Index