Agilent J-BERT N4903 High-Performance Serial BERT

Programming Guide

Agilent Technologies

Notices

Manual Part Number

N4903-91030

Edition

Draft edition, January 2006

Printed in Germany

Agilent Technologies, Deutschland GmbH
Herrenberger Str. 130
71034 Böblingen, Germany

Warranty

Technology Licenses

Restricted Rights Legend

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.

Contents

1
Programming Basics

Programming Basics - Concepts

This document provides the information you need for programming the Serial BERT using the Agilent IO Libraries Suite. Familiarity with the Agilent IO Libraries Suite is instrumental in understanding remote programming of the Serial BERT .

See the user documentation delivered with the Agilent IO Libraries Suite for information on how to use them.

NOTE

Depending on the options of your Serial BERT, some of the following functions may not be valid for your instrument. See the online Help or the User's Guide for a description of the available options.

CAUTION

The following pattern generator ports must be terminated with 50 Ω if they are not connected:

- Data Out
- $\overline{\text{Data Out}}$
- Clock Out
- $\overline{\text{Clock Out}}$

Before You Begin

Before You Begin - Concepts

This section provides background information that you need before you start with remote programming. It contains the following subjects:

- "Communication Overview" on page 8
- "Connecting to the Serial BERT" on page 9

Communication Overview

Communication with the Serial BERT is based on a host-client protocol. The server is the Serial BERT itself, the host is the remote client. The host requests the server to carry out specific actions; the Serial BERT carries out the actions and returns the results (if a query was sent).

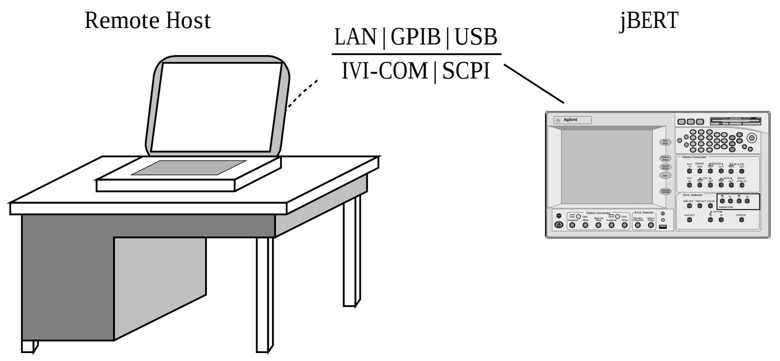


Figure 1 Serial BERT Remote Communication

The Serial BERT uses either a SCPI interface or an IVI-COM interface for communicating with the outside world. See "A Typical Serial BERT Program - Concepts" on page 13 for information on getting started with remote programming for the Serial BERT .

Depending on the options, your Serial BERT may come with a set of features for advanced measurements (such as the DUT Output measurement). These advanced measurements can only be accessed over the LAN interface. See the *Measurement Software*

Programming Guide for more information on programming the measurements. In the online Help you find a description of the available options.

Connecting to the Serial BERT

NOTE

To communicate with the Serial BERT from a remote computer, the Agilent IO Libraries Suite must be installed on this computer.

The following descriptions only provide you with the information you need from the Serial BERT. For complete instructions on how to establish connections to the Serial BERT, refer to the user documentation delivered with the Agilent IO Libraries Suite.

The Agilent IO Libraries Suite offers the following possibilities for remotely connecting to and controlling the Serial BERT :

LAN

The Serial BERT 's network settings are managed by the operating system. You can use the `IPCONFIG` command in the command window to get the network settings.

The steps for setting up the network connection are OS-dependent (Serial BERT 's OS is Windows XP). Contact your network administrator if you need help in defining the network settings.

GPIB

To connect to the Serial BERT via GPIB, you have to have the Serial BERT's GPIB address.

The address is displayed on the user interface. The default address is 14. See the online Help for details on how to set the GPIB address.

NOTE

When setting the GPIB address, it is recommended that you *do not* use the GPIB address 21. This address is reserved for GPIB controllers.

USB

The Serial BERT has a USB port on the rear of the instrument that you can use to connect it to a PC. This is the non-flat USB port below the GPIB port.

To connect to the Serial BERT via USB, you need the Serial BERT's USB ID. You can either use the full VISA resource string or assign an alias. See the Agilent IO Libraries Suite documentation for details.

Instrument Behavior

Instrument Behavior - Reference

The Serial BERT behaves as follows when it is turned on (or after a power-cycle):

Instrument Mode

At power on, the Serial BERT will return to the same mode as it was powered down. Normally, once it has booted, the Serial BERT is ready for either front panel operation or remote operation.

Registers and Filters

At power-on, the state of the registers and filters is:

- Normal operation

  The initial state of the registers and transition filters will be saved in the event of a power failure.

- Initial power-on

  All registers and filters are disabled except the PON, CME and EXE bits of the Standard Event Status Register and its summary bit in the Status Byte.

  The transition filters will be set to allow all conditions and events to pass.

The event registers and the error queue are cleared at each and every power-up.

Overheat Protection

The Serial BERT protects itself from damage by overheating by shutting itself down in such cases.

If the temperature of the pattern generator or error detector generator exceeds a certain threshold, the OVERHEAT bit in the Operation register is set.

There are two thresholds: caution and warning. These both set the same bit: you cannot programmatically get the threshold.

The caution threshold is not critical. You have enough time to save your current settings and gracefully shut down the instrument.

The warning threshold is critical. If this threshold is reached, the instrument will immediately shut itself down.

Overtemperature can be programmatically detected by querying the Status byte (*STB). In case of overheating by either the error detector or pattern generator, the Operation bit (bit 7) in the Status byte will be high, as will the OVERHEAT bit in the Operation register. See "How the Serial BERT Uses Status Registers" on page 25 for details on reading the status registers.

You can identify whether the error detector or pattern generator is overheating by running a self-test on both devices. To run a self-test:

See also the Serial BERT User Guide (or online Help) for details.

Operation Modes

The Serial BERT can be operated in one of two modes: local or remote.

Local Mode

In *local* mode, all the front panel controls are responsive and control the instrument.

Remote Mode

In *remote* mode, the front panel controls are inoperative and the instrument is controlled by the client. The front panel display reflects the remote programming commands received.

The Serial BERT automatically enters remote mode when a command has been received from the client. This is indicated at the top of the front panel (the **RMT** status light).

Returning to Local Mode

To return to local mode, press the front panel **Local** key. When you power-cycle the instrument, it will also start in local mode.

2
A Typical SerialBERT Program

A Typical Serial BERT Program - Concepts

The Serial BERT can be controlled by a remote program using the IVI-COM driver.

The sections of this Help provide you with information you can use to quickly get started with your first program. The examples here are written for Visual Basic 6.0, but can also be ported to any programming language supported by IVI-COM.

You can use the following links to find Agilent's IVI-COM help resources in the internet:

- ADN Introducing IVI-COM Drivers:

  www.agilent.com/find/adnivicominfo

- ADN IVI-COM Briefs and Papers:

  www.agilent.com/find/adnivicompapers

- ADN IVI-COM Drivers and Components Downloads:

  www.agilent.com/find/adnivicomdrivers

- ADN Drivers and Software Downloads:

  www.agilent.com/find/adndownloads

Prerequisites

Prerequisites - Concepts

Before you can control a Serial BERT remotely, the client computer (your PC, the Serial BERT is the host) must meet the following prerequisites:

- Agilent IO Libraries Suite installed
- IVI-COM driver installed
- Configured IO connection to the Serial BERT (you should be able to find the Serial BERT with the I/O libraries VISA assistant)

Initializing the Connection to the SerialBERT

Initializing the Connection - Concepts

The first step in setting up a program for controlling the Serial BERT is to create an object that corresponds to the instrument. You can either use the Serial BERT class (AgilentN490x), or you can use the IVI-compliant Agilent BERT class (AgilentBert).

TIP

If you set up your code using the AgilentBert class, you can easily port your programs to other IVI-compliant Agilent instruments. As Agilent's fulfillment of the IVI-compliance requirements, this class is shared by all other Agilent IVI-compliant instruments.

The examples in this document show how to set up a reference to the AgilentBert class and use this class.

Initializing the Connection - Procedures

The following code shows you how you would establish the connection to the instrument. The ResourceName (`"TCPIP1::10.0.0.207::inst0::INSTR"`) must be replaced by the instrument's address string from the VISA Assistant.

```
' First our declarations...
Private myN490X As AgilentN490x
Private myBERT As AgilentBert
Private myPG As AgilentBertLib.IAgilentBertPG
Private myPGClock As AgilentBertLib.IAgilentBertPGClock
Private myPGOut As AgilentBertLib.IAgilentBertPGOutput
Private myEDDataIn As AgilentBertLib.IAgilentBertEDDataIn

Private Sub Form_Load()
Set myN490X = New AgilentN490x
Set myBERT = myN490x.IAgilentBert
myBERT.Initialize "TCPIP1::10.0.0.207::inst0::INSTR",
True, True
End Sub

Private Sub Form_Unload(Cancel As Integer)
myBERT.Close
End Sub
```

Working with the IVI-COM Objects

Working with the IVI-COM Objects - Concepts

The Serial BERT IVI-COM driver uses a hierarchical class structure that follows the build up of the instrument. For example, the instrument itself is represented by the class AgilentN490x. The pattern generator is represented by the class IAgilentN490xPG.

To access the instrument's pattern generator, you have to first access the object, then the object's pattern generator collection, and finally the actual pattern generator.

The items in the collections are accessed by the name. The easiest way to get the name (if you do not know it) is through the collection's Name property.

Working with the IVI-COM Objects - Procedures

The following example shows you how to set up different objects for controlling the Serial BERT . These objects are used in the following examples.

```
Private Sub InitializeObjects()
' TIP: Call this sub from the Form_Load sub.
Dim EDName as String
With myBERT
    ' Get the pattern generator using the naming conventions
    Set myPG = .PGs.Item("PG1")
    ' Use the myPG object to get sub-items
    Set myPGClock = myPG.Clock
    Set myPGOut = myPG.Outputs.Item("PGOutput1")


    ' Get the error detector by catching and using its name:
    EDName =.EDs.Name(1)
    Set myED = .EDs.Item(EDName)
    Set myEDDataIn = myED.Input.DataIns.Item("EDDataIn1")


End With
End Sub
```

Changing Instrument Parameters

Changing Instrument Parameters - Procedures

The following examples show you how to:

- Change the pattern generator's clock rate and voltages
- Trigger auto-synchronization
- Set up a pattern

Changing the Pattern Generator's Clock Rate and Output Voltages

The following code sets the pattern generator's clock frequency and toggles as example the offset voltage between 0 and 0.5 Volts.

```
Private Sub SetUpPG
' Set the clock frequency
myPGClock.Frequency = 3 * 10 ^ 9
' Toggle the offset voltage (for demo purposes)
If myPGOut.OutVoltage.VOffset = 0 Then
    myPGOut.OutVoltage.VOffset = 0.5
Else
    myPGOut.OutVoltage.VOffset = 0
    End If
    End Sub
```

Analyzer Synchronization

The following code causes the error detector to synchronize.

```
Private Sub RunSync()
' First run the synchronization
myEDDataIn.Sampling.AutoAlign
' And then align the data
myEDDataIn.Synchronisation.SyncNow
End Sub
```

Setting Up a Pattern

The following code shows you how to set up a pattern. It additionally shows a small function that converts strings into arrays that Visual Basic can handle.

```
Private Sub SetUpPattern()
Dim myPattern As AgilentBertLib.IAgilentBertLocalPatternfile

' Use local pattern 13 to save the pattern files
' to a different location
Set myPattern = myBERT.LocalPatternfiles._
            Item(myBERT.LocalPatternfiles.Name(13))

With myPattern
    ' Set the length and description
    .Length = 8
    .Description = "Test pattern"
    ' Define the pattern to be alternate, set the data
    ' For VB, we have to convert the data to an array of doubles
    ' See function below for details
    .Alternate = True
    .SetData 1, AgilentLocalPatternFormatBin, _
      SetPatternData("00001111", AgilentLocalPatternFormatBin)
    .SetData 2, AgilentLocalPatternFormatBin,
      SetPatternData("11111111", AgilentLocalPatternFormatBin)
End With
```

```
' And now load the pattern to the pattern generator
myPGOut.SelectData AgilentN490xPGOutputSelectFile, _
        myPattern.Location

' And to the error detector
myEDDataIn.SelectData AgilentBertEDDataInSelectFile, _
        myPattern.Location


End Sub

Private Function SetPatternData(DataString As String, _
    Format As AgilentBertLib.AgilentBertEDPatternFormatEnum)
Dim myPattern() As Double
Dim ix As Integer
ReDim myPattern(Len(DataString) - 1)

For ix = 1 To Len(DataString)
    Select Case Format
    ' How to interpret the string depends on the format
        Case AgilentBertEDPatternFormatBin
            myPattern(ix - 1) = CByte(Mid(DataString, ix, 1))
        Case AgilentBertEDPatternFormatHex
          myPattern(ix - 1) = CByte("&H" & Mid(DataString, ix, 1))
        Case AgilentBertEDPatternFormatRaw
            myPattern(ix - 1) = CByte(Mid(DataString, ix, 1))
    End Select
Next
SetPatternData = myPattern
End Function
```

3
Recommended Programming Techniques

Recommended Programming Techniques - Concepts

This chapter provides some recommended techniques you should use when programming the Serial BERT.

Output Protection

Output Protection

The pattern generator's Data and Clock Out ports must be terminated with 50 Ω if they are not connected. Termination of output ports improves the test performance.

The following sections describe a protection algorithm and how you can handle the algorithm's delay time in an automated test environment.

Output Protection Algorithm

The instrument has an internal protection algorithm that protects the instrument from improper termination of the pattern generator's output ports.

The algorithm checks for an open condition on these ports every 100ms. If the ports are not correctly terminated, the algorithm adjusts the port's output levels to safe levels:

- V_{Term} remains unchanged.
- $V_{High} = V_{Term} + 1$ V
- $V_{Low} = V_{Term} + 0.9$ V

If the port is correctly terminated while in this state, the output levels are returned to the original levels.

NOTE

If V_{Term} is greater than 1.5V, the protection algorithm is not active.

In an automated test environment, the algorithm may introduce up to 200ms delay time when switching the DUT. You can avoid the protection algorithm from becoming active when switching the DUT (and thus enhance the test throughput).

Speed DUT Switching

At the end of a test when the DUT is ready to be changed, proceed as follows:

1 If $V_{Term} < +1.5$V, adjust a high level that is less than 1V below V_{Term} ($V_{High} < V_{Term} + 1$V). This prevents the protection algorithm from becoming active.

For example, if $V_{Term} = 1.0$V, you have to make sure that the high level is 2.0V or less. The following command shows how you set the high level of the Data Out port to 1.25 V:

```
SOURce1:VOLTage:HIGH 1.25
```

If the termination voltage is higher than +1.5V, no voltage levels need to be adjusted (the algorithm is not active).

2 Remove the tested DUT and connect the next DUT.

3 Restore both high level and low level.

4 Start testing the new DUT.

NOTE

Make sure that all Data Out and Clock Out ports are terminated. If not, the protection algorithm may become active.

Controlling the Output Levels

Controlling the Output Levels - Concepts

When the output levels are changed at the Serial BERT's data and clock output ports, four parameters are changed:

- V_{hi}
- V_{lo}
- V_{ampt}
- V_{offs}

The Serial BERT groups these parameters into "pairs" (V_{ampt}/V_{offs}, V_{hi}/V_{lo}). If one of these values is modified, its "partner" remains constant, and the values in the other pair are modified accordingly. For example, if V_{ampt} is changed, V_{offs} stays constant, and V_{hi} and V_{lo} are modified accordingly.

Controlling the Output Levels - Procedures

Changing the Voltages with IVI-COM

The IVI-COM driver provides a convenient function for setting V_{ampt} and V_{offs}: `Configure`. To set the pattern generator's data output voltage:

```
Private Sub SetPGDataOutVolt()
Dim myPG As AgilentN490xLib.IAgilentN490xPG
Dim myPGOut As AgilentN490xLib.IAgilentN490xPGOutput
Set myPG = myBERT.PGs.Item("PG1")
Set myPGOut = myPG.Outputs.Item("PGOutput1")
myPGOut.OutVoltage.Configure 1.5, 0.5, _
            myPGOut.OutVoltage.VTermination
End Sub
```

Changing the Voltages with SCPI

The following command shows how you would set the data output so that it has an amplitude of 1.5V and an offset of 0.5 V:

```
SOUR:VOLT:AMPT 1.5; OFFS 0.5
```

This sets the output accordingly (V_{Hi} = 1.25 V, V_{Lo} = -0.25).

Conversely, you could set V_{Hi} and V_{Lo} directly:

```
    SOUR:VOLT:HIGH 1.25; LOW -0.25
```

This has the same effect.

Allowing the Serial BERT to Settle

Allowing Serial BERT to Settle - Concepts

When patterns are sent to the pattern generator or error detector , the Serial BERT requires some time to settle before. The following topics explain how the instruments react to pattern changes.

How Pattern Changes Affect the Pattern Generator

The Serial BERT requires some time to change the patterns at the pattern generator and error detector . This is particularly true for large text-based (ASCII) patterns that have to be loaded from the file system. In such a case, it is a recommended technique to always query the Serial BERT 's Operation Complete status after changing the pattern.

How Pattern Changes Affect the Error Detector

When the pattern changes, the error detector has to resync to the new incoming signal. Depending on the signal, the alignment method used, and the desired BER threshold, this procedure can take up to half a minute or more.

Allowing Serial BERT to Settle - Procedures

When patterns have been changed, you should check the status registers to make sure that the operation is complete before continuing with your program.

Checking the Settling with IVI-COM

The following example illustrates how to check the synchronization status using IVI-COM.

```
Private Sub CheckSyncStatus()
Dim BERTStatus As AgilentN490xLib.IAgilentN490xStatus
Dim myED As AgilentN490xLib.IAgilentN490xED
Dim myPG As AgilentN490xLib.IAgilentN490xPG
Set BERTStatus = myBERT.Status
Set myED = myBERT.EDs.Item("ED1")
Set myPG = myBERT.PGs.Item("PG1")
' First enable the register of interest:
' Operation register, bit 10, positive transition
BERTStatus.Register(AgilentN490xStatusRegisterOperation, _
        AgilentN490xStatusSubRegisterEnable) = &H400
BERTStatus.Register(AgilentN490xStatusRegisterOperation, _
      AgilentN490xStatusSubRegisterPositiveTransition) = &H400
BERTStatus.Register(AgilentN490xStatusRegisterOperation, _
      AgilentN490xStatusSubRegisterNegativeTransition) = &H400
' Standard Event register, bit 0, positive transition
BERTStatus.Register(AgilentN490xStatusRegisterStandardEvent, _
        AgilentN490xStatusSubRegisterEnable) = 1
' Now clear the registers
BERTStatus.Clear
' ED should track the PG
myED.Input.DataIns.Item("EDDataIn1").TrackingEnabled = True
' Load the pattern
myPG.Outputs.Item("PGOutput1").SelectData
AgilentN490xPGOutputSelectFile, "testptr.ptrn"
' Just wait until the Operation bit goes low
Do While BERTStatus.SerialPoll And &H80
    DoEvents
Loop
End Sub
```

Checking the Analyzer Sync with SCPI Using Agilent I/O Libraries

The following example illustrates how to check the synchronization status using SCPI.

```
/* We need to check sync loss bit
    of the Questionable register (bit 10) */
const unsigned int QUESTION_REG_10 = 2^10;
unsigned int question_reg;
unsigned int opc_stat;
/* Make sure the error detector tracks
    the pattern generator */
viPrintf(vi, "SENSe1:PATTern:TRACk ON\n");
/* Load a pattern file to the instruments */
viPrintf(vi, "SOURce1:PATTern:SELect FILename, testfile.ptrn\n");
/* Wait until the instrument is in operational state */
viQueryf(vi, "*OPC?\n", "%d", &opc_stat);
do
{
    /* Get the Questionable register */
    viQueryf (vi, "STATus:QUEStionable:CONDition?\n", "%d",
                &question_reg);
    /* Loop until the sync loss bit goes low */
}
while(question_reg && QUESTION_REG_10);
```

If `Question_con_reg` returns a value that includes bit 10 ("1024"), this is an indication that the error detector has not yet synchronized to the new pattern. In this case, the instrument has not yet settled.

NOTE

File accessing, especially for large files can take some time. Control programs must be prepared for time-outs of this size.

Reading the Serial BERT's Status

Reading the Serial BERT's Status - Concepts

The Serial BERT has a set of status registers that you can use to monitor the status of the hardware, software, and any running tests.

Overview of Registers

Specifically, it has the following registers:

- Status Byte

  The Status Byte is a single register that stores the events occurring on the other registers.

- Standard Event Status Register

  The Standard Event Status Register monitors some non-critical errors and basic operations.

- Questionable Data Status Register

  The bits in the Questionable Data Status Register are set when certain events occur in the Serial BERT that can lead to questionable results.

- Operation Status Register

  The Operation Status Register indicates when certain operations have been completed.

- Clock Loss Status Register

  The Clock Loss Status Register indicates if the Serial BERT's pattern generator or error detector have lost the clock signal.

How the Serial BERT Uses Status Registers

You can determine the state of certain instrument hardware and firmware events and conditions by programming the status register system.

The following subsections provide you with details about the Serial BERT's status system.

Overview of the Serial BERT's Status System

The Serial BERT has status reporting features that give important information about events and conditions within the instrument. For example, a flag may be set to indicate the end of a measurement or perhaps a command error. To access this information, it is necessary to query a set of registers using SCPI.

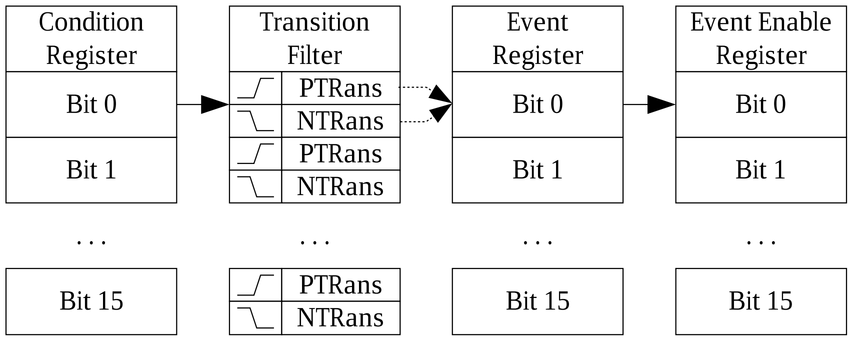
Serial BERT's Status System Structure

The Serial BERT's status system is comprised of multiple registers that are arranged in a hierarchical order. The lower-level status registers propagate their data to the higher-level registers in the data structures by means of summary bits. The Status Byte register is at the top of the hierarchy and contains general status information for the Serial BERT's events and conditions. All other individual registers are used to determine the specific events or conditions.

For figures showing Serial BERT's status registers, see "Serial BERT Register Model" on page 28.

Status Register Group Model

The following figure illustrates the typical structure of a status register.



As shown in this figure, most status registers actually consist of five registers:

- Condition

  The condition register continuously monitors the hardware and firmware status of the instrument. There is no latching or buffering for a condition register. It is updated in real time.

  This register is read by the CONDition? SCPI commands.

- Negative Transition

The negative transition register specifies the bits in the condition register that will set corresponding bits in the event register when the condition bit changes from 1 to 0.

This register is set and read by the NTRAnsition[?] SCPI commands.

- Positive Transition

  A positive transition register specifies the bits in the condition register that will set corresponding bits in the event register when the condition bit changes from 0 to 1.

- Event

  An event register latches transition events from the condition register as specified by the positive and negative transition filters. Bits in the event register are latched, and once set, they remain set until cleared by either querying the register contents or sending the *CLS command.

- Event Enable

  An enable register specifies the bits in the event register that can generate a summary bit. Summary bits are, in turn, used by the next higher register.

The registers work together as follows:

1 The *Condition Register* corresponds to a condition on the hardware or in the software. If the monitored condition is present, the corresponding bit is high.

2 The *Transition Registers* monitor changes in the Condition Register. If the Positive Transition Register is configured to watch for a condition, when this condition occurs, and the bit in the Condition Register goes high, the Positive Transition Register passes this event to the *Event Register*.

3 If this bit is enabled in the *Enable Event Register*, a summary bit is generated in the next higher register. For the higher register, this is the Condition Register, and the event is handled the same way as described here.

NOTE

The transition and enable registers for the Failure Status register (and its subregisters) cannot be modified.

Reading the Serial BERT's Status - Procedures

See “Starting ELOC and Verifying that it is Running” on page 48 for an example of how to access the registers in IVI-COM. See “Preparing the Registers (SCPI)” on page 51 for a SCPI example.

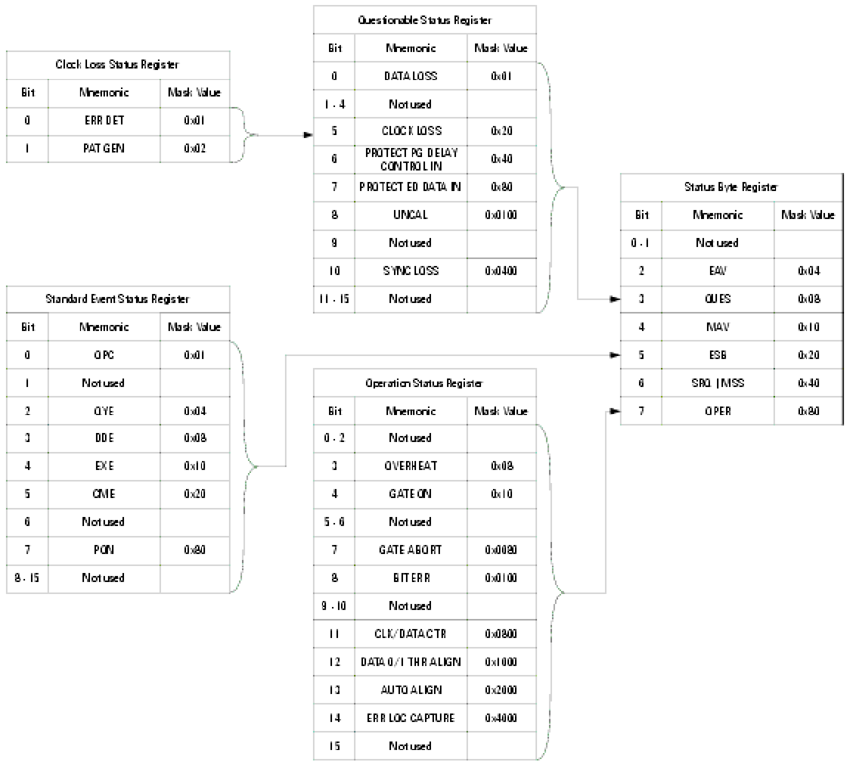
Reading the Serial BERT's Status - Reference

NOTE

Depending on the options of your Serial BERT, some of the status bits may not be valid for your instrument. See the online Help or the User's Guide for a description of the available options.

Serial BERT Register Model

The following figure shows the Serial BERT 's status register hierarchy.



Clock Loss Status Register

Bit	Mnemonic	Mask Value
0	ERR DET	0x01
1	PAT GEN	0x02

Questionable Status Register

Bit	Mnemonic	Mask Value
0	DATA LOSS	0x01
1 - 4	Not used	
5	CLOCK LOSS	0x20
6	PROTECT PG DELAY CONTROL IN	0x40
7	PROTECTED DATA IN	0x80
8	UNCAL	0x0100
9	Not used	
10	SYNC LOSS	0x0400
11 - 15	Not used	

Status Byte Register

Bit	Mnemonic	Mask Value
0 - 1	Not used	
2	EAV	0x04
3	QUES	0x08
4	MAV	0x10
5	ESB	0x20
6	SRQ | MSS	0x40
7	OPER	0x80

Standard Event Status Register

Bit	Mnemonic	Mask Value
0	OPC	0x01
1	Not used	
2	QYE	0x04
3	DDE	0x08
4	EXE	0x10
5	CME	0x20
6	Not used	
7	PON	0x80
8 - 15	Not used	

Operation Status Register

Bit	Mnemonic	Mask Value
0 - 2	Not used	
3	OVERHEAT	0x08
4	GATE ON	0x10
5 - 6	Not used	
7	GATE ABORT	0x0080
8	BIT ERR	0x0100
9 - 10	Not used	
11	CLK/DATA CTR	0x0800
12	DATA 0/1 THR ALIGN	0x1000
13	AUTO ALIGN	0x2000
14	ERR LOC CAPTURE	0x4000
15	Not used	

Status Byte

The Status Byte is the summary register to which the other registers report. Each reporting register is assigned a bit in the Status Byte Register.

The bits in the Status Byte byte have the following meaning:

Table 1

Bit	Mnemonic	Description
0	Not used	
1	Not used	
2	EAV	Error available: The error queue contains at least one message.
3	QUES	A bit has been set in the Questionable Data Status register (indicates that a signal is of questionable quality).
4	MAV	Message available: There is at least one message in the message queue.
5	ESB	A bit in the Standard Event Register has been set.
6	SRQ or MSS	Value depends on the polling method; see below for details.
7	OPER	A bit in the Operation Status Register has been set.

Bit 6 Bit 6 has two definitions, depending on how the access is polled:

- Serial Poll

  If the value of the register is read using the serial poll (SPOLL), bit 6 is referred to as the Service Request (SRQ) Bit. It is used to interrupt and inform the active controller that the instrument has set the service request control line, SRQ.

- *STB?

  If the register is read using the common command *STB? , bit 6 is referred to as the master summary bit or MSS bit. This bit indicates that the instrument has requested service. The MSS bit is not cleared when the register is read using the *STB? command. It always reflects the current status of all the instrument's status registers.

Standard Event Status Register

The Standard Event Status register is a 16-bit register group that gives general-purpose information about the instrument. It sets bit 5 in the Status Byte.

Table 2

Bit	Mnemonic	Description
0	OPC	Operation Complete bit. It is set in response to the *OPC command, but only if the instrument has completed all its pending operations.
1	Not used	
2	QYE	Query error bit. It indicates that there is a problem with the output data queue. There has been an attempt to read the queue when it is empty, the output data has been lost, or the query command has been interrupted.
3	DDE	Device-dependent error bit. It is set when an instrument-specific error has occurred.

Table 2

Bit	Mnemonic	Description
4	EXE	Execution error bit. It is set when a command (GPIB instrument specific) cannot be executed due to an out of range parameter or some instrument condition that prevents execution.
5	CME	Command error bit. It is set whenever the instrument detects an error in the format or content of the program message (usually a bad header, missing argument, or wrong data type etc.).
6	Not used	
7	PON	Power-on bit. It is set each time the instrument is powered from off to on.
8-15	Not used	

NOTE

This register is compatible with the generalized status register model. It is comprised of an event and enable register, but no condition register or transition filter. All positive transitions in this register are latched.

Clock Loss Register

The Clock Loss Register group indicates whether the pattern generator or error detector has experienced a clock signal loss. The output of this register sets bits 5 and 9 (Clock Loss) in the Questionable Status Register.

Table 3

Bit	Mnemonic	Description
0	ERR DET	Clock loss condition at the error detector.
1	PAT GEN	Clock loss condition at the pattern generator.
2-15	Not used	

Questionable Status Register

The Questionable Status Register indicates that a currently running process is of questionable quality. The output of this register sets bit 3 of the Status Byte.

Table 4

Bit	Mnemonic	Description
0	DATA LOSS	This bit is set when the data source is turned off, not connected, or the cables or device is faulty. This bit can also set when the 0/1 threshold is not in the eye limits of the incoming data signal. In this last case, use Auto Align or select Avg 0/1 Threshold.
1-4	Not used	

Table 4

Bit	Mnemonic	Description
5	CLOCK LOSS	This bit is set when the pattern generator receives no external clock signal or the error detector receives no clock input signal. To find out which of the 2 events is causing this bit to set, you must poll the Clock Loss Status Register; see “Clock Loss Register” on page 31.
6	PROTECT ED DATA IN	This bit indicates that the protection mechanism for the Data Input port of the error detector was activated, e.g. the voltage or current measured at this port was out of range.
7	PROTECT PG DELAY CTRL IN	This bit indicates that the protection mechanism for the Delay Control Input port of the pattern generator was activated, e.g. the voltage or current measured at this port was out of range.
8	UNCAL	This bit is set when the serial number of the installed pattern generator or error detector does not match the calibration file in the instrument.
9	Not used	

Table 4

Bit	Mnemonic	Description
10	SYNC LOSS	This bit is set when the error detector pattern does not match the incoming data pattern or the BER of your device is higher than the sync threshold.
11-15	Not used	

NOTE Depending on the options of your Serial BERT, some of the status bits may not be valid for your instrument. See the User Guide for a description of the available options.

Operation Status Register

The output of this register gives information about the current operation the instrument is performing. It sets bit 7 of the Status Byte.

Table 5

Bit	Mnemonic	Description
0-2	Not used	
3	OVERHEAT	The pattern generator or error detector has a higher-than-normal temperature.
4	GATE ON	An accumulated measurement is in progress.
5-6	Not used	
7	GATE ABORT	Indicates that the repetitive accumulation period was aborted.

Table 5

Bit	Mnemonic	Description
8	BIT ERR	The instrument has detected a bit error.
9-10	Not used	
11	CLK/DATA CTR	Indicates that the clock/ data alignment is in progress.
12	DATA 0/1 THR ALIGN	Indicates that the 0/1 threshold alignment is in progress.
13	AUTO ALIGN	Indicates that the auto alignment is in progress.
14	ERR LOC CAPTURE	Indicates that there is an Error Location Capture measurement in progress.
15	Block Change	When a user-defined sequence is generated: Reports a transition from one block to another.

NOTE

Depending on the options of your Serial BERT, some of the status bits may not be valid for your instrument. See the User Guide for a description of the available options.

Running the Fast Eye Mask

Running the Fast Eye Mask - Concepts

NOTE This topic is only valid for instruments with the Fast Eye Mask option. See the User Guide for a description of the available options for your instrument.

The Fast Eye Mask measurement is first of all meant for production and screening tests. It allows to determine very quickly whether the eye opening seen at the output signal of a device is within specifications, that is, within certain timing and voltage limits.

The following topics inform you on how you can programmatically set up Fast Eye Mask measurements.

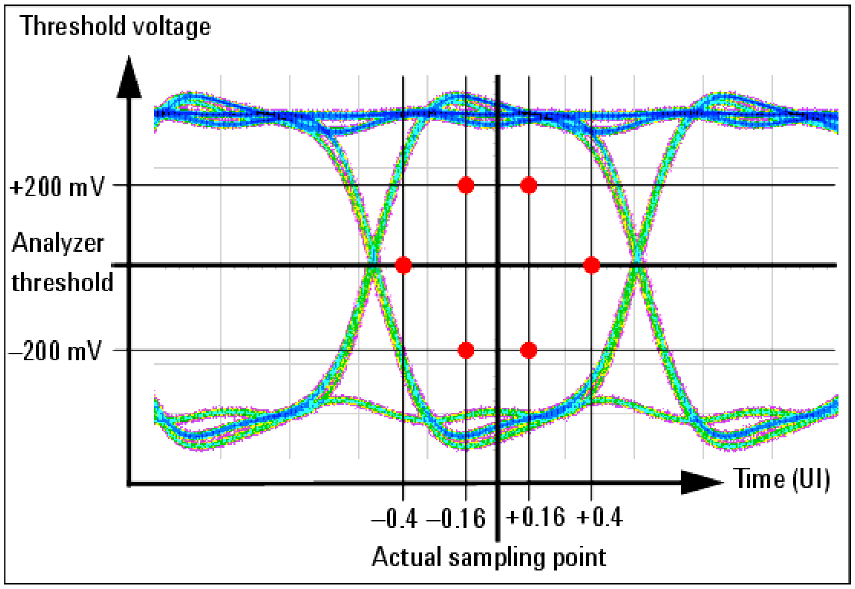
See the online Help or the User's Guide   for details on working with the Fast Eye Mask.

To be able to set up and run a Fast Eye Mask measurement using SCPI, you should be aware that the Fast Eye Mask measurement is programmatically implemented as an object on the instrument. You must first create an instance of the object, and then use the handle returned to address this object.

As good programming practice, also be sure to close the object when you are finished with your test.

Setting Up Data Points

When you create a measurement object, the object is set up by default with six symmetrically placed data points, as shown in the following figure:



These settings can be changed, and up to 32 measurement points can be defined. For this, you would first change the number of measurement points, and then specify each of the points as required.

Running the Fast Eye Mask - Procedures

The following code indicates how you could set up and run the Fast Eye Mask using SCPI.

NOTE

The Fast Eye Mask measurement displayed in the user interface cannot be controlled remotely. When you set up a program to run a Fast Eye Mask measurement remotely, a separate measurement object is created. Any programmatical changes to this measurement object will not be reflected in the user interface.

Prepare the Measurement

To create the measurement object and prepare the measurement:

1 Create the session:

```
meas:fem:cre?
```

```
2
```

Note that the returned number (handle) has to be used as a suffix in each of the subsequent SCPI commands to address the measurement.

2 Define how many bits are to be compared:

```
meas:fem2:par:mcb 1.0E+6
```

3 Specify how many errors at one point move the measurement to the next point:

```
meas:fem2:par:merr 1
```

4 Now enable error mode:

```
meas:fem2:par:merr:mode DIS
```

5 Specify the allowed bit error rate:

```
meas:fem2:par:pfcr 1.0E-9
```

Run the Measurement

To run the measurement:

1 Start the measurement:

```
meas:gen2:go
```

2 Check if the measurement has finished:

```
meas:gen2:opc? BLOCk
```

```
1
```

Or check the measurement progress:

```
meas:gen2:prog?
```

```
1.0E+0
```

Evaluate the Results

To evaluate the results:

1 Now check if the measurement was passed:

```
meas:gen2:pass?
```

```
0
```

2 Get the number of points measured:

```
meas:fem2:par:point:numb?
```

```
6
```

3 Query the six points to see which one failed:

```
meas:fem2:poin:pass? 1 0
```

...

```
meas:fem2:poin:pass? 6
```

```
0
```

In our example, points 1 and 6 failed, the other points passed.

4 Specify how the timing resolution is to be reported:

```
meas:fem2:par:tres:type UINT
```

5 Specify how the threshold is to be reported:

```
meas:fem2:par:thr:type ABS
```

6 Check the timing resolution and threshold at points 1 and 6:

```
meas:fem2:par:point? 1 -4.0E-1,1.525000035763E-1
```

```
meas:fem2:par:point? 6 1.6E-1,-4.749999642372E-2
```

7 And finally close the object:

```
meas:gen2:clos
```

Running the Eye Diagram

Running the Eye Diagram - Concepts

The Eye Diagram measurement is generally used for production and screening tests. It helps in determining very quickly whether the eye opening seen at the output signal of a device is within specifications, that is, within certain timing and voltage limits.

The following topics inform you on how you can run the Eye Diagram measurements.

To be able to set up and run an Eye Diagram measurement using SCPI, you should know that this measurement is implemented as an object on the instrument. You must first create an instance of the object, and then use the returned handle to address this object.

As a good programming practice, remember to close the object when you are finished with your test.

Running the Eye Diagram - Procedures

The following code indicates how you could set up and run the Eye Mask using SCPI.

NOTE

The Eye Diagram measurement displayed in the user interface cannot be controlled remotely. When you set up a program to run an Eye Diagram measurement remotely, a separate measurement object is created. The user interface does not reflect any change in program for this measurement object.

Example to Running the Measurement

To run the measurement:

1 Create the session: `:meas:emas(*):cre?` 2

The return value 2 is the handle.

2 Start the measurement: `meas:gen2:go`

3 To get the measured data: `:meas:gen2:fetc:data?`

Running JTol Characterization

Running JTol Characterization - Concepts

The Jitter Tolerance Characterization test determines the amplitude of jitter a data receiving circuit can cope with before it produces an untolerable number of errors.

The following topics inform you on how you can programmatically set up Jitter Tolerance Characterization measurements.

See the online Help or the User's Guide for details on working with the Jitter Tolerance Characterization.

To be able to set up and run a Jitter Tolerance Characterization measurement using SCPI, you should be aware that the measurement is programmatically implemented as an object on the instrument. You must first create an instance of the object, and then use the handle returned to address this object.

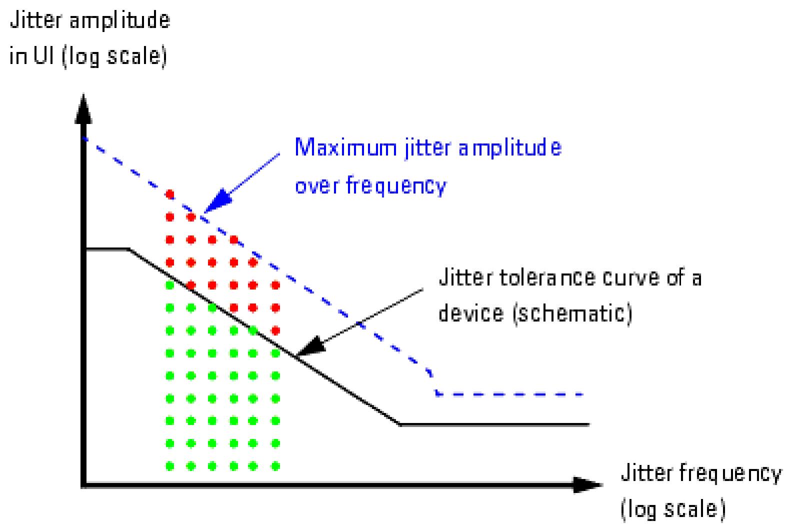
As good programming practice, also be sure to close the object when you are finished with your test.

Setting Up Data Points

The Jitter Tolerance Characterization test generates jitter with varying amplitudes. The condition for proceeding from one amplitude to the next and the direction (upwards or downwards) can be specified, as well as the step size. The test for a given frequency stops when the measured BER exceeds (or falls below) the target BER.

The test can be swept automatically over a wide frequency range. Test result is the jitter tolerance curve of the device. This curve is constructed from the pass/fail transitions of the measured points.

The following figure shows an example.



Besides the target bit error ratio, you need to specify the verification method and the positioning of the measured points. The latter can be done in manifold ways.

Running JTol Characterization - Procedures

The following code indicates how you could set up and run the Jitter Tolerance Characterization test using SCPI.

NOTE

The Jitter Tolerance Characterization measurement displayed in the user interface cannot be controlled remotely. When you set up a program to run a Jitter Tolerance Characterization measurement remotely, a separate measurement object is created. Any programmatical changes to this measurement object will not be reflected in the user interface.

Prepare the Measurement

To create the measurement object and prepare the measurement:

1 Create the session:

```
meas:jtol:char:cre?
```

```
7
```

Note that the returned number (handle) has to be used as a suffix in each of the subsequent SCPI commands to address the measurement.

2 Specify the target bit error ratio:

```
meas:jtol:char7:par:btes:tber 1.0E-12
```

3 Specify how the target BER shall be verified:

```
meas:jtol:char7:par:btes CLEV
```

4 For confidence mode (CLEVel), set the confidence level:

```
meas:jtol:char7:par:btes:clev 95E-2
```

5 Specify the frequencies to be measured. The following command creates a list of 10 logarithmically equidistant steps between 10 kHz and 10 MHz:

```
meas:jtol:char7:par:flis:fill 1.0E+4, 1.0E+7, 10
```

6 Decide on the search method. The following establishes a downwards logarithmic search:

```
meas:jtol:char7:par:sear DLOG
```

7 Set the stop value. This is the minimum jitter amplitude in UI:

```
meas:jtol:char7:par:sear:dlog:stop 1.0E-1
```

8 Set the ratio between two vertical amplitude steps:

```
meas:jtol:char7:par:sear:dlog:rati 6.6E-1
```

Run the Measurement

To run the measurement:

1. Start the measurement:

   ```
   meas:gen7:go
   ```

2. Check if the measurement has finished:

   ```
   meas:gen7:opc? BLOC
   ```

   ```
   1
   ```

   Or check the measurement progress:

   ```
   meas:gen7:prog?
   ```

   ```
   1.0E+0
   ```

Evaluate the Results

To evaluate the results:

1. Get the number of points measured:

   ```
   meas:jtol:char7:fetc:data:avai?
   ```

   54

2. Ask for the DUT capability:

   ```
   meas:jtol:char7:fetc:data:cap? DUT
   ```

   This returns for each of the measured frequencies the maximum jitter amplitude the DUT could stand without exceeding the target BER. These are the values that form the jitter tolerance curve.

3. Query the measured points to inspect the results in detail:

   ```
   meas:jtol:char7:fetc:data:poin? 54, 0
   ```

   This returns the settings and results for the leftmost 54 points.

4. Finally close the object:

   ```
   meas:gen7:clos
   ```

Using Error Location Capture

Using Error Location Capture - Concepts

What is Error Location Capture?

Error Location Capture allows to capture the position of an errored bit in a memory-based pattern. The instrument saves the position of the errored bit and writes a bit sequence neighbouring the errored bit to a file.

This feature can be used to find rare or random errors. A DUT could have problems handling long series of zeroes. Error Location Capture can be used to locate the bit errors in such cases.

NOTE

Error Location Capture may not be available to all instruments or options. See the online Help or the User's Guide for details on the available features of your instrument.

Restrictions for Error Location Capture

Error Location Capture is subject to the following restrictions:

- Only memory-based patterns with a unique 48-bit pattern (detect word) are allowed.
- The error detector must be aligned to the incoming stream.
- No alignment features can run during error location capture: Auto Align, 0/1 Threshold Center, Data Center
- No other advanced measurement (Output Timing, Output Levels, etc.) can be running.
- Error Location Capture can only run when the BER Location Mode is set to more than one bit (for example, all bits, or a block with a length > 1).

How to Run Error Location Capture

The following steps are recommended for running Error Location Capture:

1 Reset the instrument.

2 Load a pattern to the Pattern Generator and to the Error Detector.

3 Synchronize the Error Detector.

4 Clear the status registers.

5 Set up the Operation Status register so that it triggers when Error Location Capture starts.

   The Operation Status register should catch positive transitions on the ERR LOC CAPTURE bit (bit 14).

6 Start the Error Location Capture. This is an overlapped command.

7 Set up a loop that queries the Operation Status register until the Error Location bit goes high. This indicates that the Error Location Capture has started.

NOTE

Because Error Location Capture would run forever if no errors are detected, it is recommended to also set up a time-out in your program.

How to See if Error Location Capture has Stopped

To see if Error Location Capture has stopped:

1 Set up the Operation Status register so that it triggers when Error Location Capture stops.

   The Operation Status register should catch negative transitions on the ERR LOC CAPTURE bit (bit 14).

2 Check the status registers to see if Error Location Capture has stopped.

3 If it has stopped, check the status of Error Location Capture. It could have either been aborted or successful.

See also "Handling the Results" on page 47 for more information.

How to Abort Error Location Capture

Error Location Capture runs until it detects an error and stops. If there are no errors in the data stream, it would run forever. It can also be interrupted by a remote program (or user) by the following actions:

- Disabling error location
  - IVI-COM: `IAgilentN490xEDErrorLocation.Mode = AgilentN490xEDErrorLocationModeOff`

  - SCPI: `SENS:ELOC OFF`
- Incorrect sample point (faulty measurement)
- Action not compatible with error location currently being performed, for example:
  - Selecting a new pattern
  - Changing the current pattern
  - Starting synchronization or alignment

There are various other actions that also abort the run. Changing the sampling point has no effect (as long as the sampling point does not leave the eye).

Understanding the Status

Error Location Capture is not immediately able to detect bit errors when the start command is given, and does not immediately terminate when the stop command is given. There is a specific delay that must be heeded. For remote programming, it is sufficient to wait 400ms.

You can check both, the status of the command and the status of the measurement.

Command Status

To query the status of the Error Location Capture *command*:

- IVI-COM: `IAgilentN490xEDErrorLocation.CaptureErrors`
- SCPI: `SENSe1:ELOCation?`

Measurement Status

To query the status of the Error Location Capture *measurement*:

- Use the following commands:
  - IVI-COM: `IAgilentN490xEDErrorLocation.ReadState`
  - SCPI: `SENSe1:ELOCation:VERBose?`
- In the Operation Status register, check the Error Location Capture bit (bit 14) :
  - If the bit is high, the measurement is running.
  - If the bit is low, the measurement is not running. It may be not started yet, successfully finished, or aborted.

Handling the Results

Once an Error Location Capture has been successfully finished, you can get the following results:

- Location of first errored bit
- Number of all errored bits in the stored bit sequence
- Comparison pattern between expected pattern and received data

  A bit sequence neighbouring the errored bit is written to a file. The captured data is saved as an alternating pattern:

  – Pattern A contains the expected data.

    This is the pattern downloaded to the error detector.

  – Pattern B contains the errored data:

    This is a bit sequence where 1s mark the bit positions where an error occurred and 0s mark the positions where the expected bits were received.

  The captured pattern is the bit stream detected at the analyzers data input. To calculate the captured pattern, XOR the bits from pattern A with the bits from pattern B.

  The pattern description contains the first error, the error count, date and time.

  The name of the pattern file is ELOC_RESULT_CURRENT.ptrn for the current capture and ELOC_RESULT_PREVIOUS.ptrn for the previous capture. These patterns are saved under C:\<Instrument Model>\Pattern on the machine with the firmware server.

TIP The position of the first errored bit is automatically taken over as the bit position for BER location mode.

How to Handle Run Errors

Errors in Error Location Capture are handled differently than standard instrument errors:

- Errors caused by starting or stopping Error Location Capture are put in the standard error queue.
- Internal run errors caused during Error Location Capture are neither put into the standard error queue nor reported by the status register's error flag. In such a case, the response to SENS:ELOC:VERB? is ELOC__FAILED.

Using Error Location Capture - Procedures

There is a slight delay (~400ms) after error location capture is triggered before the measurement actually starts. There are two ways to handle this:

- For simplicity, just wait in your program 400ms before continuing.
- For precision, monitor the error location status bit until it registers that error location has started.

The following code examples show how to set the registers and then run Error Location Capture.

NOTE

See the online Help or the User's Guide for a list of requirements that have to be met to perform Error Location Capture measurements.

Running ELOC in IVI-COM

The following subroutines show you how you can prepare the registers and then run error location.

Defining the Required Variables

```
' The following variables are required in the code.
' Note that myBERT is a Serial BERT object that has already been
' created and initialized, for example through
' myBERT.Initialize "GPIB2::11::INSTR", True, False, ""

Private myED As IAgilentN490xED
Private myEDDataIn As IAgilentN490xEDDataIn
Private myPG As IAgilentN490xPG
Private myPGDataOut As IAgilentN490xPGOutput
Private myELOC As IAgilentN490xEDErrorLocation
Private myStatus As IAgilentN490xStatus
Private myELOC_CountedErrors As Integer
Private myELOC_FirstBitError As Long
Private myStatusByte As Integer
Private myOperReg As Long
```

Starting ELOC and Verifying that it is Running

```
Private Sub Start_And_Verify_ELOC()

Set myED = myBERT.EDs.Item("ED1")
Set myEDDataIn = myED.Input.DataIns.Item("EDDataIn1")
Set myPG = myBERT.PGs.Item("PG1")
Set myPGDataOut = myPG.Outputs.Item("PGOutput1")
Set myELOC = myEDDataIn.ErrorLocation
Set myStatus = myBERT.Status
```

```
' Reset the instrument:
myBERT.Utility.Reset
' Load a pattern to the pattern generator and _
  error detector:
myPGDataOut.SelectData AgilentN490xPGOutputSelectPRBN, "7"
myEDDataIn.SelectData AgilentN490xEDDataInSelectPRBN, "7"

' Synchronize the error detector:
myEDDataIn.Synchronisation.SyncNow

' Clear the status registers:
myStatus.Clear

' Set the status registers to trigger when Error Location _
    Capture starts:
With myStatus
' First clear the registers:
        .Clear
        .Register(AgilentN490xStatusRegisterOperation, _
            AgilentN490xStatusSubRegisterEnable) = &H4000
        .Register(AgilentN490xStatusRegisterOperation, _
            AgilentN490xStatusSubRegisterPositiveTransition)
            = &H4000
        .Register(AgilentN490xStatusRegisterOperation, _
          AgilentN490xStatusSubRegisterNegativeTransition) = 0
End With

' Error location only runs for all bits
myELOC.Mode = AgilentN490xEDErrorLocationModeAllBits

' Start the actual capture
myELOC.CaptureErrors
' Wait until the ELOC register goes high:
Do
        myStatusByte = myStatus.SerialPoll
        If myStatusByte <> 0 Then
            myOperReg = _
             myStatus.Register(AgilentN490xStatusRegisterOperation, _
                                AgilentN490xStatusSubRegisterCondition)
            If myOperReg And &H4000 <> 0 Then Exit Do
        End If
        DoEvents
Loop While True
' We now know that Error Location Capture is running.
End Sub
```

Setting Up the Status Registers

```
Private Sub Set_Up_Registers_For_Stop()

' Set the operation register to trigger when Error Location _
    Capture stops:
With myStatus
        .Register(AgilentN490xStatusRegisterOperation, _
            AgilentN490xStatusSubRegisterEnable) = &H4000
        .Register(AgilentN490xStatusRegisterOperation, _
          AgilentN490xStatusSubRegisterPositiveTransition) = 0
        .Register(AgilentN490xStatusRegisterOperation, _
         AgilentN490xStatusSubRegisterNegativeTransition) = &H4000
End With
End Sub
```

Manually Stopping ELOC

```
Private Sub Manually_Stop_ELOC()

myELOC.Mode = AgilentN490xEDErrorLocationModeOff

' Loop until ELOC has actually stopped:
Do
    myStatusByte = myStatus.SerialPoll
    If myStatusByte <> 0 Then
        myOperReg = _
          myStatus.Register(AgilentN490xStatusRegisterOperation, _
                            AgilentN490xStatusSubRegisterCondition)
        If myOperReg And &H4000 <> 0 Then Exit Do
    End If
    DoEvents
Loop While True
' Verify the ELOC status:
If myELOC.ReadState = AgilentN490xEDErrorLocationStateAborted Then
    ' Any code for verifying the manual stopping
Else
    ' Any code for handling other states
End If

End Sub
```

Manually Inserting an Error

```
Private Sub Insert_Error_and_Get_Results()
' Add an error to the bit stream:
myPG.Input.ErrorAddition.InsertManually

' Now wait until Error Location Capture stops:
Do
    myStatusByte = myStatus.SerialPoll
    If myStatusByte <> 0 Then
      myOperReg = myStatus.Register(AgilentN490xStatusRegisterOperation, _
AgilentN490xStatusSubRegisterCondition)
        If myOperReg And &H4000 <> 0 Then Exit Do
    End If
    DoEvents
Loop While True


' Verify that Error Location Capture has finished successfully:
If myELOC.ReadState = AgilentN490xEDErrorLocationStateSuccess Then
    ' Read the results
    myELOC_CountedErrors = myELOC.ReadCount   ' Should be 1
   myELOC_FirstBitError = myELOC.BitAddress  ' Location of errored bit
Else
    ' Any code for handling problems
End If


End Sub
```

Running ELOC in SCPI

The following is an example for running ELOC in SCPI with a memory based pattern and a stimulated error.

Preparing the Registers (SCPI)

To prepare the instrument and to verify that the instrument is ready by monitoring the error location status bit:

1 Reset the instrument:

   `*RST`

2 Select a memory-based pattern:

   SOURce1:PATTern:SELect PRBN7

3 Align the error detector to the pulse generator:

   SENSe1:EYE:ALIGN:AUTO 1

4 Clear the status registers:

   `*CLS`

5 Set up the Operation status register so that it triggers when Error Location Capture starts:

   ```
   STATus:OPERation:NTRansition 0 STATus:OPERation:PTRansition
   16384
   ```

6 Activate the error location bit (bit 14) in the Operation register:

   `STATus:OPERation:ENABle 16384`

Starting Error Location Capture

To run error location capture in SCPI:

1 Start error location capture:

```
SENSe1:ELOCation ONCE
```

2 Set up a loop in your program and wait until the measurement starts:

```
SENSe1:ELOCation:VERBose?
```

As soon as the measurement is running, this should return `ELOC__RUNNING`.

Alternatively, you can set up a loop and wait until the OPER bit (bit7) of the status byte is set (`*STB?`). Then check the Operation status register (`STATus:OPERation?`). If bit14 is set, ELOC has started.

What to do if errors are captured

To check whether errors are captured and to view the results:

1 Clear the status registers:

```
*CLS
```

2 Set up the Operation Status register so that it triggers when Error Location Capture stops:

```
STATus:OPERation:NTRansition 16384
STATus:OPERation:PTRansition 0
```

3 Stimulate an error in the bit stream:

```
SOURce1:PATTern:EADDition ONCe
```

4 Set up a loop in your program and wait until the measurement stops:

```
SENSe1:ELOCation:VERBose?
```

As soon as the measurement is successfully finished, this should return `ELOC__SUCCESS.`

Alternatively, you can set up a loop and wait until the OPER bit (bit7) of the status byte is set (`*STB?`). Then check the Operation status register (`STATus:OPERation?`). If bit14 is set, ELOC is no longer running. However, checking the status bytes will not tell you the reason why the measurement is not running.

If Error Location Capture is stopped or aborted the program should inform you or take appropriate steps.

5 Get the number of errored bits found:

```
SENSe1:ELOCation:ECOunt?
```

This should return 1 (since we added only one error).

6 Query the location of the errored bit:

```
SENSe1:ELOCation:BEADdress?
```

Files containing the captured data are saved under C:\<InstrumentModel>\Pattern.

7 Read the expected data of the last run (pattern A):

```
SOURce1:PATTern:UFILe:DATA? A,C:\<InstrumentModel>
\ELOC_RESULT_CURRENT.ptrn
```

8 Read the errored bits of the last run (pattern B):

```
SOURce1:PATTern:UFILe:DATA? B,C:\<InstrumentModel>
\ELOC_RESULT_CURRENT.ptrn
```

To calculate the captured pattern, XOR the bits from pattern A with the bits from pattern B. See also "Handling the Results" on page 47 for more information.

Aborting Error Location Capture

To abort error location capture in SCPI:

1 Clear the status registers:

```
*CLS
```

2 Set up the Operation status register so that it triggers when Error Location Capture stops:

```
STATus:OPERation:NTRansition 16384
STATus:OPERation:PTRansition 0
```

3 Stop error location capture:

```
SENSe1:ELOCation OFF
```

4 Set up a loop in your program and wait until the measurement stops:

```
SENSe1:ELOCation:VERBose?
```

As soon as the measurement is aborted, this should return `ELOC__ABORTED`.

Alternatively, you can setup a loop and wait until the OPER bit (bit7) of the status byte is set (`*STB?`). Then check the Operation status register (`STATus:OPERation?`). If bit14 is set, ELOC is no longer running. However, checking the status bytes will not tell you the reason why the measurement is not running.

Note that for an aborted measurement, no result files are generated.

Using Interrupts

Using Interrupts - Concepts

You may want to know when a particular event occurs, without having to continually poll the reporting register. The best way to do this is with the use of interrupts.

Service Request Example

Interrupts or Service Requests (SRQ) allow the instrument to pause the controller when the contents of a particular register change. The controller can then suspend its present task, service the instrument, and return to its initial task.

The basic steps involved in generating a service request (SRQ) are as follows:

- Decide which particular event should trigger a service request.
- Locate the corresponding status register.
- Set the transition filter to pass the chosen transition of that event.
- Set the enable register from that register group to pass that event to set the summary bit in the Status Byte Register.
- Set the Status Byte Enable Register to generate an SRQ on the chosen summary bit being set.

Using Interrupts - Procedures

The process of using interrupts is best explained by looking at an example. The following example generates an SRQ from an event in the Operation Status group.

Specifically, it causes the error detector to generate a service request upon detection of a bit error. This is done by catching the positive transition of the Serial BERT 's BIT ERR event (bit 8 in the Operation Status register).

Using Interrupts with SCPI

Note that you can only test this example if the error detector is aligned to an incoming bit stream. For testing purposes, it is recommended that you have connected the pattern generator's Data Out port to the error detector's Data In port using a shielded cable and performed an alignment (bit error rate should be 0).

To generate and detect an interrupt upon detection of a bit error:

1 Reset the instrument.

```
*RST
```

2 Align the error detector to the pulse generator.

```
SENSe1:EYE:ALIGN:AUTO 1
```

3 Reset the status subsystem.

```
*CLS
```

Note: This resetting and aligning must be carried out before setup of the operation and status bytes. Otherwise the errors generated during alignment would be reflected in the status system.

4 Set the Operation Status register's transition registers so that the positive transition of the BIT ERR bit (bit 8) is caught:

```
STATus:OPERation:PTRAnsition 256
STATus:OPERation:NTRAnsition 0
```

Note: The default setting of the transition registers is to pass only positive transitions.

5 Set the enable mask in the Operation Status register on bit 8:

```
STATus:OPERation:ENABle 256
```

With this setting, any bit error (bit 8: BIT ERR) is reported in the Status Byte register.

6 Program the Service Request Enable Register to generate a service request when the Operation Status summary bit (OPER) is set in the Status Byte Register:

```
*SRE 128
```

7 Now add an error to the bit stream:

```
SOURce1:PATTern:EADDition ONCE
```

As soon as the error is added to the bit stream, bit 8 in the Operation Status register is set. Because positive transitions of bit BIT ERR are caught and reported in the OPER bit of the Status Byte register, a service request is generated.

The current program is left and the interrupt handler is started.

To handle the service request:

1 Read the status byte.

   `*STB?`

   The current value of the status byte is returned, which is 192.

   (Because we just generated a bit error, the SRQ/MSS (bit 6) and the OPER (bit 7) of the status byte should be high.)

2 Check the bits in the returned value to see which bits are high.

   The set OPER bit tells us that an event was detected within the Operation Status register.

3 Now check the Operation Status event register.

   `STATus:OPERation:EVENt?`

   The current value of the Operation Status event register is returned, which is 256.

   The active BIT ERR (bit 8) tells us that a bit error was detected.

   Note that when you query the Operation Status *event* register, its value is cleared (the second of two consecutive queries will return0).

Working With User Patterns

Working With User Patterns - Concepts

The following topics provide information on the recommended techniques for working with user patterns.

Techniques for Editing User Patterns

The recommended way to edit a user pattern in IVI-COM is as follows:

- Define the pattern

  This includes the length, description, and whether the pattern is alternate or standard.

- Insert the data

  The data format and the data itself must be defined.

- Send the pattern to the pattern generator and/or error detector
- Set up a trigger on the pattern generator to be sent with the pattern

NOTE

The Serial BERT can use 12 user patterns (UPATtern<n>) and any number of user pattern files (UFILe). There is absolutely no difference between these patterns. The user patterns are stored in the same format on the file system, with the name UPAT<n>.ptrn (for example, upat12.ptrn).

The user patterns are provided for backwards compatibility. It is recommended that you use user pattern files, and *not*the user patterns.

NOTE

UPAT0 is a synonym for the pattern currently executed by the instrument.

How the Serial BERT Uses Alternate Patterns

These patterns are used to define the pattern generator's data output signal. Various commands can be used to define which pattern is sent at any one time. These commands, and how they interact, are described below.

Source

The source defines how the Serial BERT determines what should be output. The following alternatives are available:

- Internal

  Alternate pattern output is determined internally by the instrument (for example, from the user interface or remote program).

- External

  Alternate pattern output is determined by the signal at AUX IN. This can either be edge-sensitive or level-sensitive.

- Blanking

  Output can be shut off according to the level at AUX IN. If AUX IN high, output is generated, if AUX IN low, no output.

NOTE

It is important to understand this setting regarding the usage of the signal at the AUX IN port. If you select External, the signal at AUX IN defines *which* pattern (A or B) is sent. If you select Blanking, the signal at AUX IN defines *if* a signal is sent at all. The latter also works for standard (not alternating) patterns.

Mode

Mode defines how the output is generated. Alternatives are:

- Alternate

  Output signal is defined by the selected pattern.

- Oneshot

  One instance of pattern B is inserted into the output pattern upon trigger.

- LLevel

  Output depends on signal level at AUX IN. If high, pattern B is output, if low, pattern A is output.

- REdge

  Output depends on the signal edge at AUX IN. At rising edge, one instance of pattern B is output.

AlternatePattern/Select

AlternatePattern/Select defines what pattern is output. It is only applicable to Alternate patterns with the Source set to INTErnal or output Blanking. The following options are available:

- A Half

  Only pattern A is output.

- B Half

  Only pattern B is output.

- AB Half

  Pattern A and pattern B are sent alternatively (one instance A, one instance B, and so on).

The following table shows how these commands work together:

Table 6

Source	Mode	Description
External	LLevel	The signal at Aux In controls which half of the pattern is output. If Aux In=logic high, pattern B is sent. If Aux In=logic low, pattern A is sent.
	REdge	When a rising edge occurs at the Aux In, a single occurrence of pattern B is inserted into a continuous pattern A output.
Internal	Alternate	The AlternatePattern/ Select command controls which half of the pattern is output. The options are: pattern A only (A half) pattern B only (B half) alternating A B (AB half)
	Oneshot	IVI-COM: The BShot command inserts one instance of pattern B into the output. SCPI: The :APCHange:IBHalf command inserts one instance of pattern B into the output.

Table 6

Source	Mode	Description
Blanking	Alternate	The signal at Aux In controls whether output is generated: If Aux In=logic high, output is generated. If Aux In=logic low, no output is generated. The generated output depends on the Select command (A Half, B half, AB Half).

How the Serial BERT Sends Triggers

The Serial BERT can repeatedly send trigger signals either according to a clock divider, or according to the output pattern.

Triggering upon Divided Clock

The trigger pulse is sent from the pattern generator's TRIG OUT port. If the trigger mode is **Divided Clock**, the trigger is sent according to the clock ratio.

Triggering upon Pattern

If the trigger mode is **Pattern**, the trigger is sent according to the selected pattern.

Depending on the selected pattern, you have the following possibilities for setting the position of the trigger:

- PRBS and PRBN patterns

  You can define the pattern, the occurrence of which sends the trigger.
- Mark Density and Zero Substitution patterns

  You can define the bit position that causes a trigger to be sent.
- User patterns

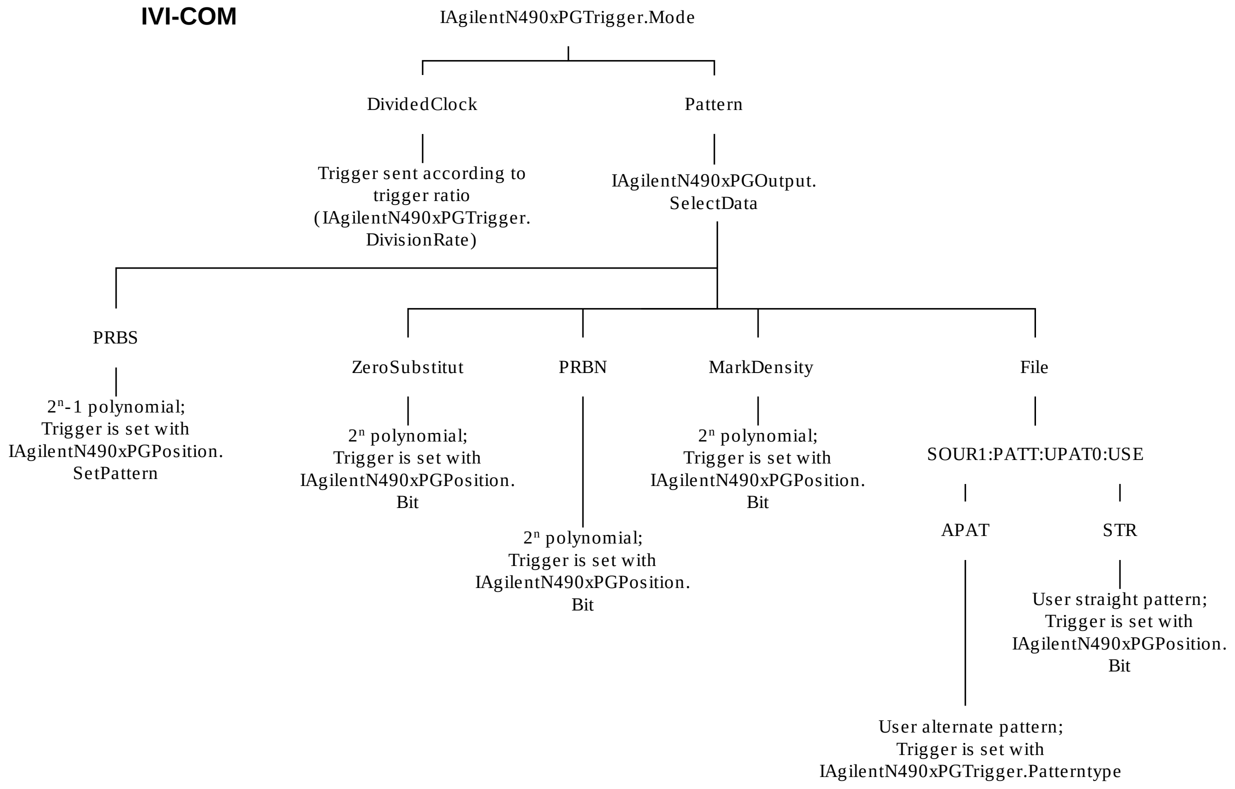
  You can define whether a trigger is sent every time a pattern is sent, or every time a pattern is changed (for alternate patterns).

Triggering upon Alternate Patterns

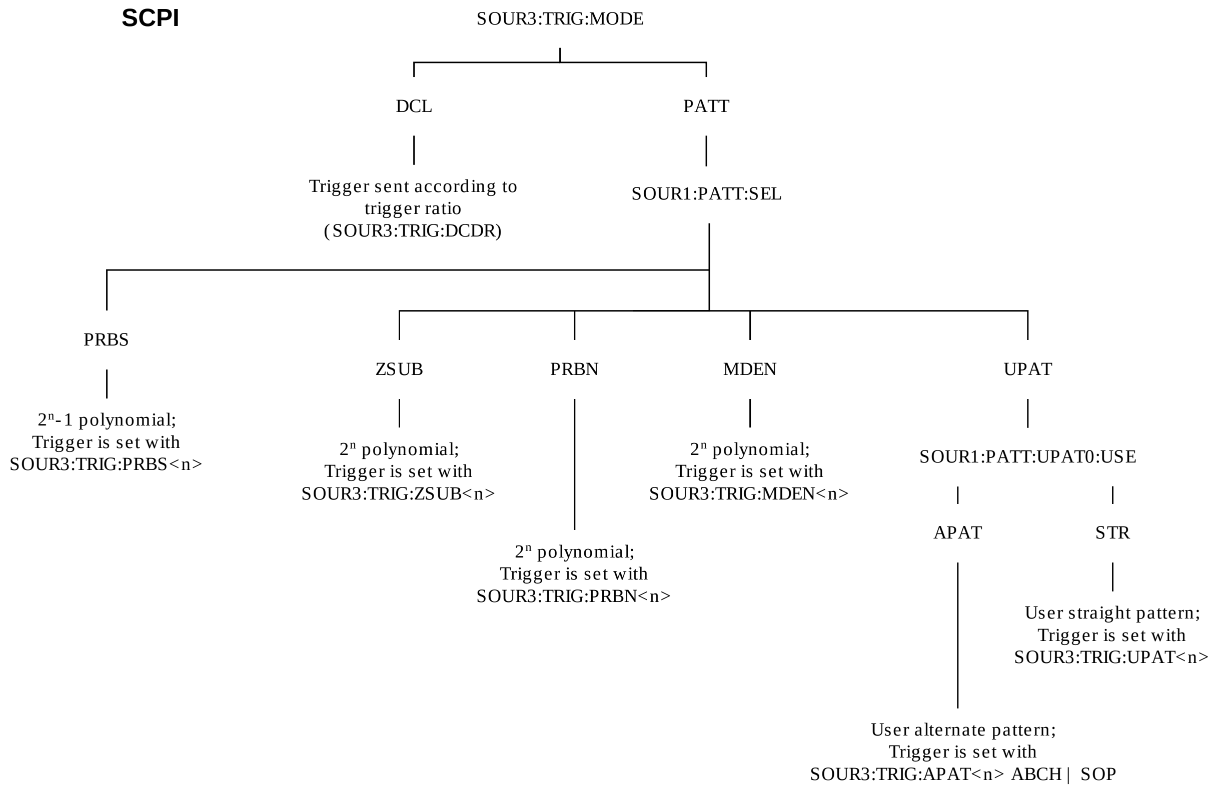
Alternate patterns are composed of two halves. The half that is sent out can be defined according to input at the Aux In port, triggered by the instrument internally, or can be triggered by the user. This is defined according the mode.

The following graphics shows the dependencies for sending patterns.

IVI-COM


IAgilentN490xPGTrigger.Mode
DividedClock
Pattern
Trigger sent according to trigger ratio (IAgilentN490xPGTrigger. DivisionRate)
IAgilentN490xPGOutput. SelectData
PRBS
ZeroSubstitut
PRBN
MarkDensity
File
2ⁿ-1 polynomial; Trigger is set with IAgilentN490xPGPosition. SetPattern
2ⁿ polynomial; Trigger is set with IAgilentN490xPGPosition. Bit
2ⁿ polynomial; Trigger is set with IAgilentN490xPGPosition. Bit
2ⁿ polynomial; Trigger is set with IAgilentN490xPGPosition. Bit
SOUR1:PATT:UPAT0:USE
APAT
STR
User straight pattern; Trigger is set with IAgilentN490xPGPosition. Bit
User alternate pattern; Trigger is set with IAgilentN490xPGTrigger.Patterntype

SCPI


SOUR3:TRIG:MODE
DCL
PATT
Trigger sent according to trigger ratio (SOUR3:TRIG:DCDR)
SOUR1:PATT:SEL
PRBS
ZSUB
PRBN
MDEN
UPAT
2ⁿ-1 polynomial; Trigger is set with SOUR3:TRIG:PRBS<n>
2ⁿ polynomial; Trigger is set with SOUR3:TRIG:ZSUB<n>
2ⁿ polynomial; Trigger is set with SOUR3:TRIG:MDEN<n>
SOUR1:PATT:UPAT0:USE
APAT
STR
2ⁿ polynomial; Trigger is set with SOUR3:TRIG:PRBN<n>
User straight pattern; Trigger is set with SOUR3:TRIG:UPAT<n>
User alternate pattern; Trigger is set with SOUR3:TRIG:APAT<n> ABCH | SOP

Triggering in Sequence Mode

If a user-defined sequence of patterns is loaded to be generated, the trigger mode can be set to **Divided Clock** or **Sequence**.

If the trigger mode is **Sequence**, a trigger can be generated whenever a block of the user-defined sequence starts or restarts. Whether that really happens for a particular block is defined for each block in the SequenceExpression. For more information see "SequenceExpression for User-Defined Sequences" on page 107.

Working With User Patterns - Procedures

The following topics show you how to set up a program in IVI-COM and SCPI that does the following:

- Sets up an alternate pattern file and sends it to the pattern generator and error detector
- Sends triggers according to the input at AUX IN
- Sends a PRBS pattern to the pattern generator and error detector

Working with User Patterns in IVI-COM

Creating Alternate Patterns

The following code provides an example of how to set up an alternate pattern.

```
Private Sub DefinePatternFile()
' Define the classes;
' myBERT is the already created Serial BERT object
Dim myPG As IAgilentN490xPG
Dim myED As IAgilentN490xED
Dim myPGTrig As IAgilentN490xPGTrigger
Dim myPatternFile As IAgilentN490xPGPatternfile
Dim myData1() As String
Dim myData2() As String
Dim ix As Integer


Set myPG = myBERT.PGs.Item("PG1")
Set myED = myBERT.EDs.Item("ED1")
Set myPGTrig = myPG.Trigger
Set myPatternFile = myPG.Patternfiles.Item("PGPatternfile1")
' Set up one array with alternating 1s and 0s
' and one with only 0s
ReDim myData1(32)
ReDim myData2(32)
For ix = 1 To 32
    If (ix And 2) = 2 Then
        myData1(ix) = "1"
    Else
        myData1(ix) = "0"
    End If
    myData2(ix) = "0"
Next


With myPatternFile
    ' Define the pattern
    .Length = 32
    .Description = "Test pattern"
    .Alternate = True


    ' Set the pattern's data
    .SetData 1, AgilentN490xPGPatternFormatBin, myData1
    .SetData 2, AgilentN490xPGPatternFormatBin, myData2


    ' Error detector should track the pattern generator
    myED.Input.DataIns.Item("EDDataIn1"). _
            TrackingEnabled = True
    ' Now send the pattern to the instrument
    .SelectData


End With
' And finally send a trigger upon pattern change
myPGTrig.Mode = AgilentN490xPGTriggerModePattern
myPGTrig.Patterntype = AgilentN490xPGTriggerPatterntypeABChange
myPGTrig.Position.Bit = 32
End Sub
```

Triggering on AUX IN

The following example shows how to set up the Serial BERT to send pattern B upon a rising edge at AUX IN:

```
Private Sub AlternatePatterns()
' Define the classes;
#39; myBERT is the already created Serial BERT object
Dim myPG As IAgilentN490xPG
Dim myPGAuxIn As IAgilentN490xPGAuxIn
Dim myPGDataOut As IAgilentN490xPGOutput
Dim myPatternFile As IAgilentN490xPGPatternfile

Set myPG = myBERT.PGs.Item("PG1")
Set myPGAuxIn = myPG.Input.AuxIn
Set myPGOut = myPG.Outputs.Item("PGOutput1")

' Now send the pattern generator's
' pattern file 1 to the pattern generator
Set myPatternFile = myPG.Patternfiles.Item("PGPatternfile1")
myPatternFile.SelectData

' Set the source to be external
myPGAuxIn.Source = AgilentN490xPGAuxInSourceExternal

' We want alternate patterns
myPGAuxIn.AlternatePattern = _
        AgilentN490xPGAuxInAlternatePatternABHalf

' With B sent at the rising edge
myPGAuxIn.Mode = AgilentN490xPGAuxInModeREdge


End Sub
```

PRBS Patterns

The following code shows you how to set up a PRBS pattern and send it to the instrument:

```
Private Sub UsePRBS()
Dim myPG As IAgilentN490xPG
Dim myPGOut As IAgilentN490xPGOutput
Dim myPGTrig As IAgilentN490xPGTrigger
Dim myPGTrigPos As IAgilentN490xPGPosition
Dim myED As IAgilentN490xED
Dim myPattern() As String
Dim ix As Integer

Set myPG = myBERT.PGs("PG1")
Set myED = myBERT.EDs("ED1")
Set myPGOut = myPG.Outputs.Item("PGOutput1")
Set myPGTrig = myPG.Trigger
Set myPGTrigPos = myPG.Trigger.Position
myED.Input.DataIns.Item("EDDataIn1").TrackingEnabled = True
myPGOut.SelectData AgilentN490xPGOutputSelectPRBN, "7"
' Create an array for the trigger pattern
' We want to trigger on "0011111"
ReDim myPattern(7)
myPattern(1) = "0"
myPattern(2) = "0"
For ix = 3 To 7
    myPattern(ix) = "1"
Next
' And set the trigger
myPGTrig.Mode = AgilentN490xPGTriggerModePattern
myPGTrig.Position.SetPattern myPattern
End Sub
```

Working with User Patterns in SCPI

When creating user patterns in SCPI, it is necessary to format the data. You can use the PATTern:FORMat[:DATA] command to define the format for entering the data. This command allows you to define how the block data should be entered: as standard ASCII data (256 characters), hex data (4 bits per character), or binary data (1s and 0s).

Editing Straight Patterns

For user patterns in the STRaight mode, it is recommended that the following commands be executed *in order*:

1 Define that a STRaight pattern be used.

```
SOURce1:PATTern:UPATtern<n>:USE STRaight
```

2 Set the length of the pattern.

```
SOURce1:PATTern:UPATtern<n>[:LENGth] <NR1>
```

3 Define how the data is to be packed.

```
SOURce1:PATTern:FORMat:DATA PACKed, 1|4|8
```

4 Define the pattern data.

```
SOURce1:PATTern:UPATtern<n>:DATA <block data>
```

Defining a Trigger

Note that you can optionally define a trigger for a specific bit in the pattern:

1 Define the trigger out mode.

```
SOURce3:TRIGger:MODE PATTern
```

2 Set the bit on which the trigger is sent.

```
SOURce3:TRIGger:UPATtern<n> <NR1>
```

Editing Alternate Patterns

For user-patterns in the APATtern mode, it is recommended that the following commands be executed *in order*:

1 Define that an Alternate PATtern be used.

```
SOURce1:PATTern:UPATtern<n>:USE APATtern
```

2 Define the length of the pattern.

```
SOURce1:PATTern:UPATtern<n>[:LENGth] <NR1>
```

3 Define how the data is to be packed.

```
SOURce1:PATTern:FORMat:DATA PACKed, 1|4|8
```

4 Define the data in pattern A.

```
SOURce1:PATTern:UPATtern<n>:DATA A, <block data>
```

5 Define the data in the pattern B.

```
SOURce1:PATTern:UPATtern<n>:DATA B, <block data>
```

Defining a Trigger

Note that you can optionally define a trigger when there is a pattern change:

1 Define the trigger out mode.

```
SOURce3:TRIGger:MODE PATTern
```

2 Optionally define a trigger when there is a pattern change.

```
SOURce3:TRIGger:APATtern<n> ABCHange
```

Using Alternate Patterns

It is recommended that the following commands be executed *in order*:

1 Select the pattern to be used. This has to be an alternate pattern.

```
SOURce1:PATTern:SELect UPATtern<n>
```

2 Define the source for switching.

```
SOURce1:PATTern:APCHange:SOURce EXTernal | INTernal | BLANking
```

3 Define the mode for switching.

```
SOURce1:PATTern:APCHange:MODE
ALTernate | ONEShot | LLEVel | REDGe
```

4 Use the following command to define which half of the pattern should be sent.

```
SOURce1:PATTern:APCHange:SELect AHALf | BHALf | ABHAlf
```

Examples for Using User Patterns in SCPI

To set up a user pattern using SCPI:

1 Set the error detector to track the pattern generator (that is, to use the same pattern).

```
SENSe1:PATTern:TRACk  ON
```

2 Define the file 'ALT1s0s.ptrn' to be an alternate pattern.

```
SOURce1:PATTern:UFILe:USE 'ALT1s0s.ptrn', APATtern
```

3 Define the input data format to be binary (1s and 0s).

```
SOURce1:PATTern:FORMat:DATA PACKed, 1
```

4 Set the pattern length to 8 bits.

```
SOURce1:PATTern:UFILe:LENGth 'ALT1s0s.ptrn', 8
```

5 Define pattern A.

```
SOURce1:PATTern:UFILe:DATA A, 'ALT1s0s.ptrn', #1810101010
```

6 Define pattern B.

```
SOURce1:PATTern:UFILe:DATA B, 'ALT1s0s.ptrn', #1800000000
```

7 Load the pattern to the pattern generator.

```
SOURce1:PATTern:SELect FILENAME, 'ALT1s0s.ptrn'
```

NOTE

When the pattern is loaded to the pattern generator, it is also loaded to the error detector (TRACking ON). Keep in mind that the error detector can only track pattern A. When pattern B is sent, the error detector will still expect pattern A.

Switching at Aux In

With these commands, pattern A is sent when the input at **Aux In** is low. When the input is high, pattern B is sent.

1 Load the previously defined pattern to the pattern generator.

```
SOURce1:PATTern:SELect FILename, 'ALT1s0s.ptrn'
```

2 Select the source for switching patterns to **Aux In**.

```
SOURce1:APCHange:SOURce EXTernal
```

3 Define that alternate patterns should be sent.

```
SOURce1:APCHange:MODE ALTernate
```

Generating a Trigger

The following commands expand on the previous example. They cause a a trigger to be generated on the **Trigger Out** port whenever the user pattern is changed (from pattern A to pattern B).

1 Define the trigger output mode.

```
SOURce3:TRIGger:MODE PATTern
```

2 Set up the trigger for pattern changes.

```
SOURce3:TRIGger:APATtern ABCHange
```

Switching on the Rising Edge

With these commands, pattern A is sent until a rising edge is detected at **Aux In**. When the rising edge is detected, pattern B is sent.

1 Load the pattern to the pattern generator.

```
SOURce1:PATTern:SELect FILENAME, 'ALT1s0s.ptrn'
```

2 Set the source for switching patterns to **Aux In**.

```
SOURce1:APCHange:SOURce EXTernal
```

3 Define that pattern B should be sent upon the rising edge.

```
SOURce1:APCHange:MODE REDGe
```

Programmatically Switching

These commands allow the programmer to manually set which pattern should be sent.

1 Load the pattern to the pattern generator.

```
SOURce1:PATTern:SELect FILENAME, 'ALT1s0s.ptrn'
```

2 Select the source for changing patterns to be internal.

```
SOURce1:APCHange:SOURce INTernal
```

3 Define that alternate patterns should be sent.

```
SOURce1:APCHange:MODE ALTernate
```

4 Send pattern A continuously.

```
SOURce1:PATTern:APCHange:SELect AHALf
```

5 After some event occurs, change to pattern continuous B.

```
SOURce1:PATTern:APCHange:SELect BHALf
```

6 And then set up output to automatically alternate between pattern A and pattern B.

```
SOURce1:PATTern:APCHange:SELect ABHAlf
```

Inserting Pattern B

These commands allow one instance of pattern B to be inserted into the output when the Insert B button in the user interface is pressed.

1 Load the pattern to the pattern generator.

```
SOURce1:PATTern:SELect FILENAME, 'ALT1s0s.ptrn'
```

2 Select the source for changing patterns to be internal.

```
SOURce1:PATTern:APCHange:SOURce INTernal
```

3 Select the mode to insert a single instance of pattern B.

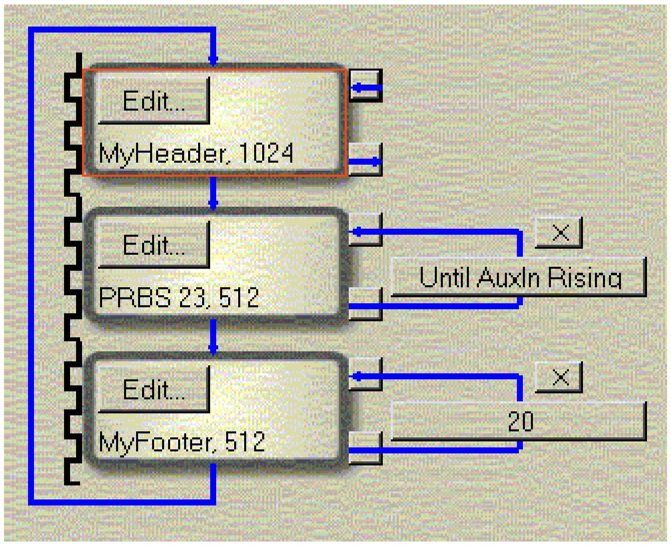
```
SOURce1:PATTern:APCHange:MODE ONEShot
```

4 Use Insert B button in GUI or use remote command in order to insert pattern B in the data output.

```
SOURce1:PATTern:APCHange:IBHalf ONCe
```

Working With User-Defined Sequences - Procedures

The following examples show you how to set up a program in IVI-COM and SCPI that generates the following sequence:



The first block references a user pattern file named `MyHeader.ptrn`. It is used to initialize the device.

The second block generates a PRBS of polynomial 2^23 -1 as payload data. It is repeated until the Auxiliary In port of the pattern generator receives a rising edge.

The third block generates the contents of a user pattern file named `MyFooter.ptrn`. It is used to reset the device and repeated 20 times.

The pattern editor can be used to specify and store the user patterns.

Setting up a Sequence in IVI-COM

This section provides a compressed example of how to set up the sequence in IVI-COM.

```
using  System;

using  Agilent.AgilentN490x.Interop;

namespace IviSequencing

{

/// <summary>

/// Summary description for Class1.

/// </summary>

class IviSequencing

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]
static void  Main (string[]  args )

{

//get an Agilen8114x driver:

Agilent8114xClass pg= new

Agilent.Agilent8114x.Interop.Agilent8114xClass();

// Connect to the instrument.

// The connection string might be different for your instrument.
Check the string with your Visa Assistant.

pg.Initialize("GPIB2::11::INSTR",false,false,"");

//For accessing the output settings we need the PGOutput interface

AgilentN490xPGOutput pgOutput = pg.Outputs.get_Item("Output1");

//Set the SequenceExpression of the pattern generator:

pg.Sequence.Expression = @"Version= 1.0; Start= IMM; Block 1=
C:\81141A\Pattern\MyHeader.ptrn; Block 2= PRBS23, 512, TrigOn;
Block 3= C:\81141A\Pattern\MyFooter.ptrn; Loop= B3, B3, 20; Loop=
B3, B3, 20; Loop= B2, B2, AuxInRising";
```

//select Sequence as data output mode:

```
pgOutput.SelectData(Agilent8114xOutputSelectEnum.
Agilent8114xOutputSelectSequence,"");
```

//Some more code...

//At the end we have to close the connection:

```
pg.Close();

}

}

}
```

Setting up a Sequence in SCPI

Setting up a Sequence in SCPI requires only two steps:

1 Create the sequence:

```
:SOUR:PATT:SEQ:DATA (Version= 1.0; Start= IMM;
Block 1= C:\N4903A\Pattern\MyHeader.ptrn; Block 2= PRBS23, 512, TrigOn;
Block 3= C:\N4903A\Pattern\MyFooter.ptrn; Loop= B3, B3, 20;
Loop= B2, B2, AuxInRising)
```

2 Enable sequence execution:

```
:SOUR1:PATT:SEL SEQ
```

If you wish to change the Trigger Out port from Divided Clock to Sequence Trigger, use the command:

```
:SOUR3:TRIG SEQ
```

By default, each block is set to TrigOn, and a trigger will be output whenever block execution starts.

4
SCPI Command Language

SCPI Command Language - Concepts

The Serial BERT is compatible with the standard language for remote control of instruments. **S**tandard **C**ommands for **P**rogrammable **I**nstruments (SCPI) is the universal programming language for instrument control.

SCPI can be subdivided into the following command sets:

- SCPI Common Commands
- SCPI Instrument Control Commands
- IEEE 488.2 Mandatory and Optional Commands

SCPI Common Commands

This is a common command set. It is compatible with IEEE 488.2 and contains general housekeeping commands. The common commands are always headed by an asterisk. A typical example is the reset command: *RST

The IEEE 488.2 command set also contains query commands. Query commands always end with a question mark.

SCPI Instrument Control Commands

The programming commands are compatible with the Standard Commands for Programmable Instruments (SCPI) standard. For more detailed information regarding the GPIB, the IEEE 488.2 standard, or the SCPI standard, refer to the following books:

- SCPI Consortium. SCPI-Standard Commands for Programmable Instruments, 1997 (http://www.scpiconsortium.org).

- International Institute of Electrical and Electronics Engineers. IEEE Standard 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation. New York, NY, 1987.
- International Institute of Electrical and Electronics Engineers. IEEE Standard 488.2-1987, IEEE Standard Codes, Formats, Protocols and Common commands For Use with ANSI/IEEE Std 488.1-1987. New York, NY, 1987.

IEEE 488.2 Mandatory and Optional Commands

In order to comply with the SCPI model as described in IEEE 488.2, the Serial BERT implements certain mandatory commands. Other commands are implemented optionally. For more detail on the IEEE 488.2 mandatory and optional commands, see "IEEE Commands – Reference" on page 85 and "Optional Commands" on page 91.

Overlapped and Sequential Commands

IEEE 488.2 defines the distinction between overlapped and sequential commands. A sequential command is one which finishes executing before the next command starts executing. An overlapped command is one which does not finish executing before the next command starts executing.

The Serial BERT has the following overlapped commands:

- SENSe[1]:GATE[:STATe] ON | 1

  (when GATE:MODE SINGle)
- SENSe[1]:EYE:TCENter|:TCENter ONCE | ON | 1
- SENSe[1]:EYE:ACENter|:ACENter ONCE | ON | 1
- SENSe[1]:EYE:ALIGn:AUTO ONCE | ON | 1
- SENSe[1]:EYE:QUICk:TCENter ONCE | ON | 1
- SENSe[1]:EYE:QUICk:ACENter ONCE | ON | 1
- SENSe[1]:EYE:QUICk:ALIGN:AUTO ONCE | ON | 1
- SENSe[1]:SYNChronizat ONCE
- SENSe[1]:ELOCation ONCE

NOTE

It is not reliable to use wait statements in the control program to facilitate the use of overlapped commands.

Because these commands may allow the execution of more than one command at a time, special programming techniques must be used to ensure valid results. The common commands *OPC, *WAI, and *OPC? can be used for this purpose. They help synchronize a device controller with the execution of overlapped commands.

The behaviors of these commands, in brief, are as follows:

- *OPC

  The *OPC command sets the Operation Complete (OPC) bit of the Standard Event Status Register (SESR) when the No Operation Pending flag is TRUE (No Operation Pending flag is attached to each overlapped command). Until that time, the controller may continue to parse and execute previous commands. It is good technique, then, to periodically poll the OPC bit to determine if the overlapped command has completed.

- *WAI

  The *WAI command allows no further execution of commands or queries until the No Operation Pending flag is true, or receipt of a Device Clear (dcas) message, or a power on.

- *OPC?

  The *OPC? query returns the ASCII character "1" in the Output Queue when the No Operation Pending flag is TRUE. At the same time, it also sets the Message Available (MAV) bit in the Status Byte Register. The *OPC? will not allow further execution of commands or queries until the No Operation Pending flag is true, or receipt of a Device Clear (dcas) message, or a power on.

NOTE

The command behaviors described above are for overlapped commands. When the same commands are used with sequential commands, the behaviors may be different.

Operation Pending Events

For the Serial BERT , six conditions can change an operation pending flag. Notice that the first four correspond to the four overlapped commands:

- A single timed accumulation period has expired.
- The automatic eye-time-centering operation has expired.
- The automatic eye-amplitude-centering operation has expired.
- An automatic alignment has occurred.
- The requested operation failed.
- The operation was aborted by the user.

Data Types

The Serial BERT has the capability of receiving and returning data in the following formats:

- STRING

  A string of human-readable ASCII characters, either quoted or non-quoted.

- NUMERIC

  The Serial BERT handles three numeric formats:

  - **<NR1>**: Integer (0, 1, 2, -1, etc.)
  - **<NR2>**: Number with an embedded decimal point (0.1, 0.001. 3.3, etc.)
  - **<NR3>**: Number with an embedded decimal point and exponent (1e33, 1.3e-12, etc.)
  - Hex preceded by #h (#hff, #hFF, etc.)

- BOOLEAN

  Boolean values can be sent to the Serial BERT as either TRUE | FALSe or 0 | 1. The Serial BERT answers queries with 0 | 1.

- BLOCK DATA

  Block data is used when a large quantity of related data is being returned. A definite length block is suitable for sending blocks of 8-bit binary information when the length is known beforehand. An indefinite length block is suitable for sending blocks of 8-bit binary information when the length is not known beforehand or when computing the length beforehand is undesirable.

  It has the following format:

  #<Length of length><Length of data><data>

  <Length of length> is a single integer that contains the number of digits in <Length of data>, which in turn contains the length of the data. So, for example, a 512-byte pattern would be defined as:

  #3512<data>

Important Points about SCPI

Important Points about SCPI - Concepts

There are a number of key areas to consider when using SCPI for the first time. These are as follows:

- Instrument Model
- Command Syntax
- Optional Parts of Commands
- Sending Commands
- Command Separators
- SCPI Command Structure

Instrument Model

SCPI guidelines require that the Serial BERT is compatible with an instrument model. This ensures that when using SCPI, functional compatibility is achieved between instruments that perform the same tasks. For example, if two different instruments have a programmable clock frequency setting, then both instruments would use the same SCPI commands to set their frequency. The instrument model is made up of a number of subsystems.

The sub-system defines a group of functions within a module and has a unique identifier under SCPI, which is called the Root Keyword.

For more details on the instrument model, see "Serial BERT Register Model" on page 28.

Command Syntax

Commands may be up to twelve characters long. A short-form version is also available which has a preferred length of four characters or less. In this document the long-form and short-form versions are shown as a single word with the short-form being shown in upper-case letters.

For example, the long-form node command SOURce has the short-form SOUR. Using the short form saves time when entering a program, however, using the long form makes a program more descriptive and easier to understand.

SCPI commands may be commands only, commands and queries, or queries only. A question mark at the end of a command indicates that it is a query. If the question mark appears in brackets ([?]), the command has a command and query form.

Note that some queries can be combined with an additional keyword that impacts the response.

`For example:`

```
SOURce9:SSCLocking:FREQuency?
```

returns the present value.

```
SOURce9:SSCLocking:FREQuency? MIN
```

returns the minimum acceptable value.

```
SOURce9:SSCLocking:FREQuency? MAX
```

returns the maximum acceptable value.

In this manual, the syntax

```
SOURce9:SIN:FREQ? MIN | MAX
```

is used for this type of queries.

Optional Command Keywords

Some layers in the SCPI command structure are optional. These optional keywords are indicated by square brackets ([ ]). A typical use for these types of keywords is with a command that is unique to one module. In this case, the top layer (Root Keyword) of the command structure may be omitted.

For example, the following command code segments are functionally identical:

```
[SOURce[1]:]PATTern:MDENsity[:DENSity] <numeric value>

SOURce:PATTERN:MDENSITY <numeric value>

PATTern:MDENsity <numeric value>

PATT:MDEN <numeric value>

patt:mden <numeric value>
```

Note that it is not necessary to include the syntax inside the square brackets ([ ]).

Sending Commands

Commands are sent over the GPIB in the same way that GPIB and IEEE 488.2 common commands are sent. The difference is that the SCPI command is "nested" into the programming language of choice. The programming language of choice may be a language such as Visual Basic, C++, or SICL.

For an examples of how commands are sent, see "Sending Commands to the Serial BERT - Concepts" on page 81.

Query Responses

It is possible to interrogate the individual settings and status of a device using query commands. Retrieving data is a two-stage operation.

The query command is sent from the controller using the OUTPUT statement and the data is read from the device using the ENTER statement. A typical example, using the SCPI IEEE 488.2 Common Command *IDN? which queries the identity of a device.

See "Sending Commands using VISA" on page 81 for an example in the C programming language of how to query the identity.

NOTE

When sending strings to the instrument, either the double quote (") or the single quote may be used ('), the latter being more suited to PASCAL programs, which make use of a single quote; the former being more suited to use in BASIC programs, which use a double quote as a delimiter. In this manual, the double quote has been used throughout.

Command Separators

The SCPI command structure is hierarchical and is governed by commas, semicolons and colons:

- Commas are used to separate parameters in one command.
- Colons are used to separate levels.
- Semicolons are used to send more than one command to the instrument at a time.

```
SENSe[1]:PATTern:UPATtern<n>:IDATa     [A|B,]
<start_bit>, <length_in_bits>, <block_data>
```

Note that the command hierarchy is indicated by colons and that the parameters (beginning with [A|B,]), are separated by commas.

Multiple Commands

It is possible to send several commands in one pass, as long as the commands all belong to the same node in the SCPI tree. The commands have to be separated by semicolons.

The following SCPI commands provide examples of this. Note that the optional characters and keywords have been removed.

```
SOURce1:VOLTage:LEVel:IMMediate:OFFSet    1.5
SOURce1:VOLTage:LEVel:IMMediate:AMPLitude 2
```

These commands can also be sent as follows:

```
VOLT:OFFS 1.5; AMPL 2.0
```

SCPI Command Structure Example

The SCPI command structure can be best examined by means of an example. For example, the command to select the pattern generator's pattern is:

```
[SOURce[1]]:PATTern[:SELect] PRBS7
```

The structure of this command can be illustrated as follows:

[SOURce [1]:]	This is the top layer of the command structure and identifies the pattern generator source subsystem.
PATTern	This is the next layer and defines subnode for setting up the pattern.
[:SELect]	This is the command itself, and is the equivalent of setting the front panel pattern selection field.
PRBS(n)	This is the parameter required by the PATTern command keyword.

NOTE

Any optional commands are enclosed in square brackets [ ] and any optional characters are shown in lower case.

A colon indicates a change of level in the command hierarchy. Commands at the same level in the hierarchy may be included in the same command line, if separated by a semi-colon.

The bar symbol (|) indicates mutually exclusive commands.

To translate this syntax into a command line, follow the convention described above. Remember, however, that the command line can be created in several different ways. It can be created with or without optional keywords, and in a long or short form. The following example gives three possible forms of the command line; all are acceptable.

In long form:

```
SOURce1:PATTern:SELect PRBS7
```

In short form:

```
SOUR1:PATT:SEL PRBS7
```

With the optional commands removed:

```
PATT PRBS7
```

The long form is the most descriptive form of programming commands in SCPI. It is used for the examples in this manual.

Sending Commands to the SerialBERT

Sending Commands to the Serial BERT - Concepts

A command is invalid and will be rejected if:

- It contains a syntax error.
- It cannot be identified.
- It has too few or too many parameters.
- A parameter is out of range.
- It is out of context.

Sending Commands using VISA

The following code example shows how to use the Agilent IO Libraries Suite to connect to the instrument via GPIB. This code also contains commented examples for USB and LAN.

This example queries the device for the identification string and prints the results.

```
#include <visa.h>
#include <stdio.h>
```

```
void main () {
    ViSession defaultRM, vi;
    char buf [256] = {0};

    /* Open session to GPIB device at address 14 */
    viOpenDefaultRM (&defaultRM);
   viOpen (defaultRM, "GPIB0::14::INSTR", VI_NULL,VI_NULL, &vi);

    /* Alternatively open a session to the device at
        IP address 10.0.1.255 */
    /*    viOpen (defaultRM,
        "TCPIP0::10.0.1.255::INSTR", VI_NULL,VI_NULL, &vi); */

    /* Or open a session to the USB device */
    /* viOpen (defaultRM,
                "usb0[2391::20496::SNN4900AXXXDE::0::INSTR]",
                VI_NULL,VI_NULL, &vi); */

    /* Or if you have assigned an alias N4903A-Lab */
   /* viOpen (defaultRM, "N4903A-Lab", VI_NULL, VI_NULL, &vi); */

    /* Initialize device */
    viPrintf (vi, "*RST\n");

    /* Send an *IDN? string to the device */
    viPrintf (vi, "*IDN?\n");

/* Read results */
    viScanf (vi, "%t", &buf);

/* Print results */
    printf ("Instrument identification string: %s\n", buf);

/* Close session */
    viClose (vi);
    viClose (defaultRM);
}
```

This returns the identity string "AGILENT TECHNOLOGIES,N4903A,3331U00101,A.01.01".

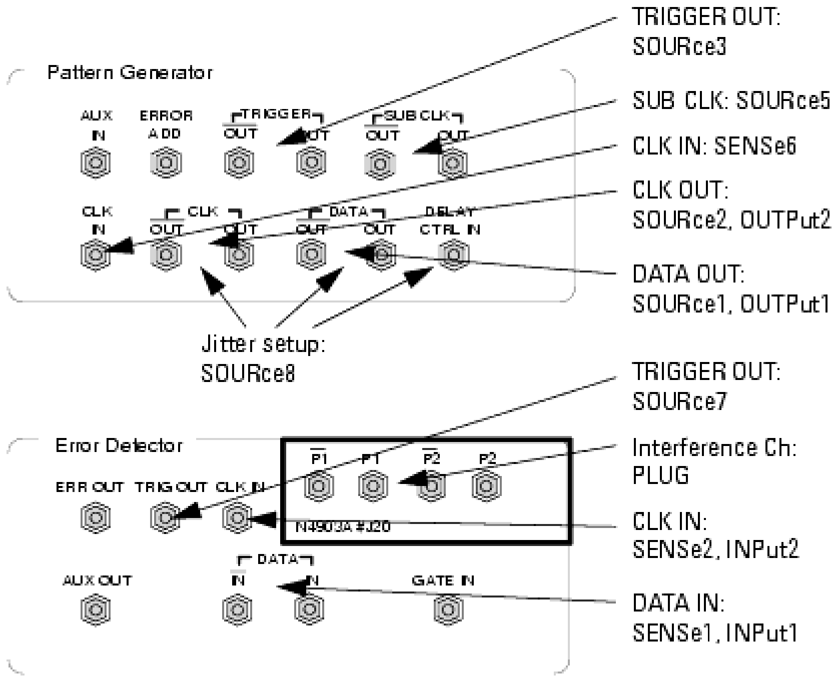
5
SCPI Command Reference

Serial BERT Subsystems

TIP

You can use the Output Window in the instrument's user interface to monitor the SCPI commands and queries. This can make it easier to find out which command is responsible for which action.

The SCPI commands are divided into *subsystems*, which reflect the various functionality of the instrument. The following figure shows where the port-related subsystems are located.



SOURce The SOURce subsystems control output signals (for example, for defining output patterns and levels). The OUTPut subsystems control the electrical port connection (for example, to disconnect the port or set the terminations).

SENSe The SENSe subsystems control the expected input signal. They correspond to the SOURce subsystems. The INPut subsystems correspond to the OUTPut subsystems; they are responsible for the electrical port connection.

NOTE

The inverted clock and data outputs track the standard outputs. For example, the pattern generator's $\overline{\text{DATA OUT}}$ port tracks the DATA OUT port. Any changes to the standard output automatically modifies the inverted output (and vice versa). Therefore, only the commands of the standard outputs are documented here.

Besides the subsystems shown above, the following subsystems are available:

- STATus

  This subsystem controls the SCPI-compatible status reporting structures.

  IVI-COM Equivalent: IAgilentN490xStatus

- SYSTem

  This subsystem controls functions such as general housekeeping and global configurations. It controls also the installation and activation of licensed options.

  IVI-COM Equivalent: IAgilentN490xSystem

- TEST

  This subsystem verifies specific hardware components for basic functionality.

  IVI-COM Equivalent: IIviDriverUtility.SelfTest

All subsystems and commands are described in this chapter.

TIP

You can find the SCPI commands and their corresponding IVI-COM commands in the online help:

1 Open the online help (Help Contents menu).

2 Open the Index tab and search for the root keyword, for example, SENSe[1], and click on it.

3 Search in the reference area for the complete command and click on the command.

A description of the command and both SCPI and IVI-COM syntax is displayed.

IEEE Commands

IEEE Commands – Reference

Mandatory Commands

The following *mandatory* IEEE 488.2 commands are implemented:

Table 8

Name	Description under
*CLS	"*CLS " on page 85
*ESE[?]	"*ESE[?] " on page 86
*ESR?	"*ESR? " on page 86
*IDN?	"*IDN? " on page 87
*OPC	"*OPC " on page 87
*OPC?	"*OPC? " on page 88
*RST	"*RST " on page 89
*SRE[?]	"*SRE[?] " on page 89
*STB?	"*STB? " on page 90
*TST?	"*TST? " on page 90
*WAI	"*WAI " on page 90

*CLS

IVI-COM Equivalent IAgilentN490xStatus.Clear (not IVI-compliant)

Syntax *CLS

Description

This command clears all status data structures in a device. For the Serial BERT , these registers include:

SESR	IEEE 488.2
OPERation Status Register	SCPI
QUEStionable Status Register	SCPI

Execution of *CLS also clears any additional status data structures implemented in the device. The corresponding enable registers are unaffected.

See “Reading the Serial BERT's Status - Concepts” on page 25 for more information about the Status Byte.

*ESE[?]

Syntax

*ESE <Num.>

*ESE?

IVI-COM Equivalent

IAgilentN490xStatus.Register (not IVI-compliant)

Description

The Standard Event Status Enable Command (*ESE) sets the Standard Event Enable Register. This register acts like a mask, so that the next time a selected bit goes high, the ESB bit in the status byte is set. See “Reading the Serial BERT's Status - Reference” on page 28 for details.

For example, if bit 0 is set in the Standard Event Enable Register, then when the OPC bit in the Standard Event register goes true, the ESB summary bit is set in the Status Byte.

The query (*ESE?) returns the contents of the Standard Event Enable Register.

*ESR?

IVI-COM Equivalent

IAgilentN490xStatus.SerialPoll (not IVI-compliant)

Syntax

*ESR?

Description This query interrogates the Standard Event Status Register. The register is cleared after it is read.

*IDN?

IVI-COM Equivalent IIviDriverIdentity (IVI-compliant)

Syntax *IDN?

Description For the Serial BERT , the Identification Query (*IDN?) response semantics are organized into four fields, separated by commas. The field definitions are as follows:

Table 10

Field	Value
Manufacturer	Agilent Technologies
Model	N490xx
Serial Number	DExxxxxxxx
Firmware Level	A.x.x.xxx

*OPC

IVI-COM Equivalent IAgilentN490xSystem.WaitForOperationComplete (not IVI-compliant)

Syntax *OPC

Description A device is in the Operation Complete Command Active State (OCAS) after *OPC has been executed. The device returns to the Operation Complete Command Idle State (OCIS) whenever the No Operation Pending flag is TRUE, while at the same time setting the OPC bit of the ESR TRUE.

The following events force the device into OCIS without setting the No Operation Pending flag to TRUE and without setting the OPC bit of the ESR:

- power on

- receipt of a DCAS message (device clear)
- execution of *CLS
- execution of *RST

Implementation of the *OPC command is straightforward in devices that implement only sequential commands. When executing *OPC, the device simply sets the OPC bit of the ESR.

In devices that implement overlapped commands, the implementation of *OPC is more complicated. After executing *OPC, the device must not set the OPC bit of ESR until the device returns to OCIS, even though it continues to parse and execute commands.

NOTE

For the Serial BERT, *OPC can be used with overlapped commands. For more information, see "Overlapped and Sequential Commands " on page 74 the .

*OPC?

IVI-COM Equivalent: IAgilentN490xSystem.WaitForOperationComplete (not IVI-compliant)

Syntax: *OPC? Command

Description: A device is in the Operation Complete Query Active State (OQAS) after it has executed *OPC?. The device returns to the Operation Complete Query Idle State (OQIS) whenever the No Operation Pending flag is TRUE, at the same time placing a "1" in the Output Queue.

The following events force the device into OQIS without setting the No Operation Pending flag TRUE and without placing a "1" in the Output Queue:

- power on
- receipt of the dcas message (device clear)

Implementation of the *OPC? query is straightforward in devices which implement only sequential commands. When executing *OPC? the device simply places a "1" in the Output Queue.

The implementation of overlapped commands in a device complicates the implementation of *OPC? and places some restrictions on the implementation of the Message Exchange Protocol (MEP). IEEE 488.2 dictates that devices shall send query responses in the order that they receive the corresponding queries. Although IEEE 488.2 recommends that *OPC? be the last query in a program message, there is nothing to prevent a controller program from ignoring this suggestion. This is why *OPC? must be sequential.

NOTE

For the Serial BERT, *OPC(?) can be used with overlapped commands. For more information, see "Overlapped and Sequential Commands " on page 74 the .

*RST

IVI-COM Equivalent IIviDriverUtility.Reset (IVI-compliant)

Syntax *RST

Description The Reset Command (*RST) sets the device-specific functions to a known state that is independent of the past-use history of the device. The command has the same effect as the front-panel PRESET key.

In addition, receipt of *RST by the error detector will cause all past results to be reset to zero.

*SRE[?]

IVI-COM Equivalent IAgilentN490xStatus.ConfigureServiceRequest (not IVI-compliant)

Syntax *SRE <Num.>

*SRE?

Description The Service Request Enable Command (*SRE) sets the Service Request Enable Register. This acts as a mask on the Status Byte, defining when the instrument can issue a service request. For a service request to be issued, the summary bit in the Status Byte must match the bit in the Service Request Enable Register. More than one bit may be set by the *SRE command.

The query returns the current contents of the Service Request Enable Register.

See "Reading the Serial BERT's Status - Reference" on page 28 for details.

*STB?

IVI-COM Equivalent IAgilentN490XStatus.SerialPoll

Syntax *STB?

Description The Read Status Byte Query (*STB?) allows the programmer to read the status byte and Master Summary Status bit. When the status byte is read using the *STB command, bit 6 of the status byte is referred to as the Master Summary (MSS) bit. With this query, the status byte is not cleared when the value is read. It always reflects the current status of all the instrument's status registers.

See "Reading the Serial BERT's Status - Reference" on page 28 for details.

*TST?

IVI-COM Equivalent IIviDriverUtility.SelfTest (not IVI-compliant)

Syntax *TST?

Description The self-test query starts all internal self-tests and places a response into the output queue indicating whether or not the device completed the self-tests without any detected errors. It returns a 0 for success; a 1 if a failure was detected.

Upon successful completion of *TST?, the device settings are restored to their values prior to the *TST?

For more precise self-test results, use "TEST:EXECute? " on page 318.

*WAI

Syntax *WAI

Description The *WAI commands allows no further execution of commands or queries until the No Operation Pending flag is true, or receipt of a Device Clear (dcas) message, or a power on.

The *WAI command can be used for overlapped commands. It stops the program execution until any pending overlapped commands have finished. Specifically, it waits until the No Operation Pending flag is TRUE, or receipt of a dcas message, or a power on.

Optional Commands

The following *optional* IEEE 488.2 commands are implemented:

Table 11

Command	Description
*OPT?	Option Identification Query
*PSC	Power On Status Clear Command
*PSC?	Power On Status Clear Query
*RCL	Recall device setup
*SAV	Save device setup

*OPT?

Syntax *OPT?

Description The Option Identification query is for identifying the instrument's options. It returns a string, for example:

- Option U13 (Upgrade 13.5 Gbit/s)
- Option A01 (Bit Recovery Mode)
- Option J12 (Jitter Compliance Suite)

See the online Help or the User's Guide for detailed information on the options and the corresponding features.

*PSC

Syntax *PSC

Description The Power-on Status Clear command controls the automatic power-on clearing of the Service Request Enable Register, the Standard Event Status Enable Register, and the Parallel Poll Enable Register.

This is a standard SCPI command. Please refer to the SCPI specification for details.

*RCL

IVI-COM Equivalent	IAgilentN490xSystem.RecallState (IVI-compliant)
Syntax	*RCL <numeric value | string>
Description	This command loads the setup from a numbered store or from a full path filename that was previously stored with "*SAV " on page 92. The range of store numbers is 0 through 9. In addition, upon receipt of *RCL, the error detector will reset all past results to zero.

NOTE: Depending on the patterns that are saved with the setup, the instrument may require up to half a minute to settle. See "Allowing Serial BERT to Settle - Procedures" on page 23 for details.

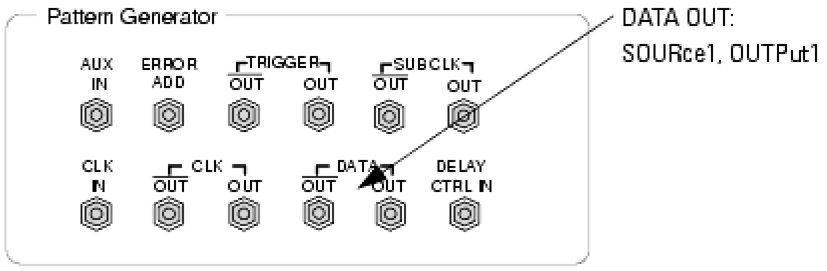
*SAV

IVI-COM Equivalent	IAgilentN490xSystem.SaveState (IVI-compliant)
Syntax	*SAV <numeric value | string>
Description	This command saves the current instrument setup into a numbered store or into a full path filename. The range of store numbers is 0 through 9. The command "*RCL " on page 92 restores the setup. The setup saves the currently used patterns, signal definitions, and other user interface settings.

SOURce[1] Subsystem

SOURce[1] Subsystem - Reference

The SOURce[1] subsystem controls the pattern generator's Data Out port.



This subsystem has the following SCPI structure:

```
[:SOURce[1]:]
 ├ :PM
 │  └ [STATe][?]
 ├ PATTern
 │  └ ...
 └ :VOLTage
    └ ...
```

This subsystem has the following commands and subnodes:

Table 12

Name	Description under
Commands	
:PM	“[SOURce[1]]:PM[:STATe][?] ” on page 94
Subnodes	

Table 12

Name	Description under
:PATTern	“[SOURce[1]]:PATTern Subnode” on page 95
:VOLTage	“[SOURce[1]]:VOLTage Subnode” on page 124

[SOURce[1]]:PM[:STATe][?]

IVI-COM Equivalent IAgilentN490xPGDelayControlInput.Enabled (not IVI-compliant)

Syntax [SOURce[1]]:PM:STATe ON | OFF | 0 | 1

[SOURce[1]]:PM:STATe?

Description Enables/disables delay control input. The query returns the state of the delay control input (0 | 1).

[SOURce[1]]:PATTern Subnode

This subnode has the following SCPI structure:

```
[:SOURce[1]:]
└ PATTern
  ├ :APChange
  │ └ . . .
  ├ :EADDition[?]
  │ ├ :RATE[?]
  │ └ :SOURce[?]
  ├ :FORMat
  │ └ [:DATA][?]
  ├ :MDENsity
  │ └ [:DENSity]
  ├ [:SELect][?]
  ├ :SEQuence
  │ └ . . .
  ├ :UFILe
  │ └ . . .
  ├ :UPATtern
  │ └ . . .
  └ :ZSUBstitut[:ZRUN][?]
```

This subnode has the following commands and subnodes:

Table 13

Name	Description under
Commands	
:EADDition[?]	"[SOURce[1]]:PATTern:EADDition[?] " on page 96
:EADDition:RATE[?]	"[SOURce[1]]:PATTern:EADDition:RATE [?] " on page 97
:EADDition:SOURce[?]	"[SOURce [1]]:PATTern:EADDition:SOURce[?] " on page 97

Table 13

Name	Description under
:FORMat[:DATA][?]	“[SOURce[1]]:PATTern:FORMat[:DATA][?] ” on page 98
:MDENsity[:DENSity][?]	“[SOURce[1]]:PATTern:MDENsity[:DENSity][?] ” on page 99
[:SELect][?]	“[SOURce[1]]:PATTern[:SELect][?] ” on page 99
:ZSUBstitut[:ZRUN][?]	“[SOURce[1]]:PATTern:ZSUBstitut[:ZRUN][?]” on page 101
Subnodes	
:APCHange	“[SOURce[1]]:PATTern:APCHange Subnode” on page 102
SEQuence	“[SOURce[1]]:PATTern:SEQuence Subnode” on page 106
:UFILe	“[SOURce[1]]:PATTern:UFILe Subnode” on page 113
:UPATtern<n>	“[SOURce[1]]:PATTern:UPATTern Subnode” on page 118

This subnode has the following commands:

[SOURce[1]]:PATTern:EADDition[?]

IVI-COM Equivalent IAgilentN490xPGErrorAddition.InsertManually (not IVI-compliant)

Syntax [SOURce[1]]:PATTern:EADDition <EADD>

[SOURce[1]]:PATTern:EADDition?

Input Parameters **<EADD>**: ONCE | 0 | 1 | OFF | ON

Return Range 0 | 1

Description

This command is a contraction of the phrase **E**rror **ADD**ition. It is used to control the addition of errors into the generated pattern.

The parameter ONCe causes a single bit error to be added to the pattern. It depends on the previous status of this command and the selected source (see "[SOURce[1]]:PATTern:EADDition:SOURce[?]" on page 97). The following table lists the dependencies:

Table 14

:EADD	:EADD:SOUR	:EADD ONCe
0	EXT	Active
	FIX	Active
1	EXT	Active
	FIX	Not active (command has no effect)

The query returns the current state of error addition.

[SOURce[1]]:PATTern:EADDition:RATE[?]

IVI-COM Equivalent

IAgilentN490xPGErrorAddition.PresetRate (not IVI-compliant)

Syntax

[SOURce[1]]:PATTern:EADDition:RATE <RATE> 10^(-3, -4,... -9)

[SOURce[1]]:PATTern:EADDition:RATE?

Return Range

10^(-3, -4,... -9)

Description

The command controls the rate of internal fixed error addition. Values between 10^{3} and 10^{9} in decade steps are permitted.

The query returns the current error add rate.

[SOURce[1]]:PATTern:EADDition:SOURce[?]

IVI-COM Equivalent

IAgilentN490xPGErrorAddition.Mode (not IVI-compliant)

Syntax

[SOURce[1]]:PATTern:EADDition:SOURce EXTernal | FIXed

[SOURce[1]]:PATTern:EADDition:SOURce?

Return Range

EXT | FIX

Description

The command controls the source of injected errors:

- EXTernal (and :EADDition[:STATe] is ON)

  Each pulse at the **Error Add** port causes an error to be added to the data stream.
- FIXed (and :EADDition[:STATe] is ON)

  Repetitive errors are internally added to the data stream. The rate of error addition is controlled by the :EADDition:RATE command.

The query returns the current error addition mode.

[SOURce[1]]:PATTern:FORMat[:DATA][?]

IVI-COM Equivalent

IAgilentN490xPGPatternfile.SetData (IVI-compliant)

Syntax

[SOURce[1]]:PATTern:FORMat:DATA <PACKed>, <Num.>

[SOURce[1]]:PATTern:FORMat:DATA?

Input Parameters

<PACKed>: permits the packing of bits within a byte to be set.

<NR1>: Can be 1, 4, or 8.

Return Range

1 | 4 | 8

Description

The command controls the format of data transfer for the :PATTern:UPATtern<n>:DATA, :PATTern:UPATtern<n>:IDATa, :PATTern:UFILe:DATA and :PATTern:UFILe:IDATa commands. The following values are possible:

- 1

  The data is sent as a string of 1s and 0s.
- 4

  The data is sent as a string of hex characters.
- 8

The data is sent as a string of full 8-bit ASCII characters.

The query returns the current value of the data pack.

See "Working with User Patterns in SCPI" on page 65 for descriptions on how to use the data packing.

[SOURce[1]]:PATTern:MDENsity[:DENSity][?]

IVI-COM Equivalent IAgilentN490xPGOutput.MarkDensity (not IVI-compliant)

Syntax [SOURce[1]]:PATTern:MDENsity[:DENSity] <Num.>

[SOURce[1]]:PATTern:MDENsity[:DENSity]?

Input Parameters **<NR2>**: 0.125, 0.25, 0.5, 0.75, 0.875

Description The command sets the ratio of high bits to the total number of bits in the pattern. The ratio may be varied in eighths, from one to seven (eighths), but excluding three and five.

The query returns the mark density in eighths.

[SOURce[1]]:PATTern[:SELect][?]

IVI-COM Equivalent IAgilentN490xPGOutput.SelectData (IVI-compliant)

Syntax [SOURce[1]]:PATTern[:SELect] <Source>

[SOURce[1]]:PATTern[:SELect]?

Input Parameters **<Source>**: PRBS<n> | PRBN<n> | ZSUBstitut<n> | MDENsity<n> | UPATtern<n> | FILename, <string> | SEQuence

Return Range PRBS<n> | PRBN<n> | ZSUB<n> | MDEN<n> | UPAT | SEQ

Description This command defines the type of pattern being generated. The parameter is retained for backwards compatibility and may be one of the following:

PRBS<n>	<n> = 7, 10, 11, 15, 23, 31
PRBN<n>	<n> = 7, 10,11,13, 15, 23

ZSUBstitut<n>	<n> = 7, 10,11,13, 15, 23
UPATtern<n>	<n> = 1 through 12
MDENsity<n>	<n> = 7, 10,11,13, 15, 23
FILename,	<string>
SEQuence	

ZSUBstitut: **Z**ero **SUB**stitution; used for defining PRBN patterns in which a block of bits is replaced by a block of zeros. The length of the block is defined by "[SOURce[1]]:PATTern:ZSUBstitut[:ZRUN][?]" on page 101.

MDENsity: **M**ark **DEN**sity; used for defining a PRBN pattern in which the user can set the mark density. The mark density is set with "[SOURce[1]]:PATTern:MDENsity[:DENSity][?] " on page 99.

UPATtern<n>: User **PAT**tern; used to define the contents of a pattern store. For the Serial BERT , <n> can be 1 to12.

FILename: A parameter that allows the remote user to load a user pattern from the instrument's disk drive. This is the preferred mechanism for loading user patterns in the Serial BERT .

NOTE

If the pattern generator and error detector are coupled, setting the pattern by using the SOURce1:PATTern:SELect command will cause the pattern to be set in both the pattern generator and the error detector. If the pattern generator and error detector are not coupled, then the error detector pattern must be selected using the SENSe [1]:PATTern:SELect command.

The query form returns the pattern's type in short form.

NOTE

If a user-defined pattern is selected and the query command is used, the response is UPAT. The particular value of <n> or the name of the file specified in the command form is not returned.

To get the path of a user pattern file, use the UFILe:NAME? command.

SEQuence: Downloads a user-defined sequence to the pattern generator and enables it. Such a sequence can be defined with the command "[SOURce[1]]:PATTern:SEQuence:DATA[?] " on page 106 or loaded from a file with "[SOURce[1]]:PATTern:SEQuence:RCL " on page 111.

[SOURce[1]]:PATTern:ZSUBstitut[:ZRUN][?]

IVI-COM Equivalent: IAgilentN490xPGOutput.ZeroSub (not IVI-compliant)

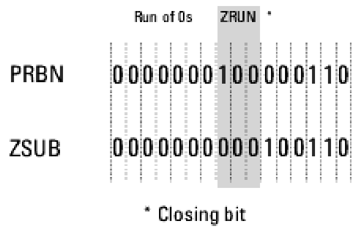
Syntax: [SOURce[1]]:PATTern:ZSUBstitut[:ZRUN] MINimum | MAXimum | <numeric value>

[SOURce[1]]:PATTern:ZSUBstitut[:ZRUN]?

Return Range: <NR3>

Description: ZSUB patterns are PRBN patterns, where a number of bits are replaced by zeroes. The zero substitution starts after the longest runs of zeroes in the pattern (for example, for PRBN 2^7, after the run of 7 zeroes). This command allows you to define the length of the run of zeroes. For example, to produce 10 zeroes in a PRBN 2^7 pattern, three additional bits after the run of 7 zeroes must be replaced by zeroes. The bit after the run of zeroes (the closing bit) is set to 1.

The following figure shows an example, where a run of 10 zeroes is inserted into a PRBN 2^7 pattern.



This command is only active when a ZSUB pattern has been selected (see "[SOURce[1]]:PATTern[:SELect][?] " on page 99).

Range: The minimum value is the PRBN value. The maximum value is length of the pattern - 1. So, for a PRBN 2^7 pattern, the minimum value is 7, and the maximum value is 127 (2^7 - 1).

[SOURce[1]]:PATTern:APCHange Subnode

This subnode has the following SCPI structure:

```
[SOURce[1]:]
└ :PATTern
   └ :APCHange
      ├ :IBHalf
      ├ :MODE[?]
      ├ :SELect[?]
      └ :SOURce[?]
```

This subnode has the following commands:

Table 16

Name	Description under
:IBHalf	"[SOURce [1]]:PATTern:APCHange:IBHalf " on page 102
:MODE[?]	"[SOURce [1]]:PATTern:APCHange:MODE[?] " on page 103
:SELect[?]	"[SOURce [1]]:PATTern:APCHange:SELect[?] " on page 104
:SOURce[?]	"[SOURce [1]]:PATTern:APCHange:SOURce[?] " on page 105

[SOURce[1]]:PATTern:APCHange:IBHalf

IVI-COM Equivalent IAgilentN490xPGAuxIn.BShot (not IVI-compliant)

Syntax [SOURce[1]]:PATTern:APCHange:IBHalf ONCe

Description This command is short for **I**nsert **BHalf**. It causes the insertion of a number of instances of pattern B. It is valid only when :APCHange:SOURce is set to INTernal and :APCHange:MODE is set to ONEShot. It is an event command, and as such has no query form.

Pattern B is repeated as necessary to reach the next 512-bit boundary in the memory. So, for example, if pattern B is 4 bits long, it is repeated 128 times. Or if it is 7 bits long, it is repeated 512 times.

See "How the Serial BERT Uses Alternate Patterns" on page 57 for more information.

[SOURce[1]]:PATTern:APCHange:MODE[?]

IVI-COM Equivalent: IAgilentN490xPGAuxIn.Mode (not IVI-compliant)

Syntax: [SOURce[1]]:PATTern:APCHange:MODE <MODE>

[SOURce[1]]:PATTern:APCHange:MODE?

Input Parameters: **<MODE>**: ALTernate | ONEShot | LLEVel | REDGe

Return Range: ALT | ONES | LLEV | REDG

*RST Setting: ALTernate

Description: This command controls the mode of operation of the alternate pattern output. The query returns the current mode of operation.

The parameters have the following meanings:

- ALTernate

  Alternate patterns are used. The pattern that is output must be defined with "[SOURce[1]]:PATTern:APCHange:SELect[?] " on page 104.

- ONEShot

  A single instance of pattern B is inserted into the output stream. This can be triggered either programmatically (with "[SOURce [1]]:PATTern:APCHange:IBHalf " on page 102, or from the user interface (with the Insert B button).

- LLEVel

  The output pattern is determined by the level of the signal at the **Aux In** port.

- REDGe

The output pattern is determined by the rising edge of the signal at the **Aux In** port.

NOTE

This command must be used together with the "[SOURce[1]]:PATTern:APCHange:SELect[?] " on page 104 and "[SOURce[1]]:PATTern:APCHange:SOURce[?] " on page 105.

For instructions on how to use these commands, refer to "How the Serial BERT Uses Alternate Patterns" on page 57.

[SOURce[1]]:PATTern:APCHange:SELect[?]

IVI-COM Equivalent	IAgilentN490xPGAuxIn.AlternatePattern (not IVI-compliant)
Syntax	[SOURce[1]]:PATTern:APCHange:SELect AHALf | BHALf | ABHalf [SOURce[1]]:PATTern:APCHange:SELect?
Return Range	AHAL | BHAL | ABH
*RST Setting	AHALf

Description

This command defines what pattern is output. It is only applicable to ALTernate patterns. The following options are available:

- AHALf

  Only pattern A is output.
- BHALf

  Only pattern B is output.
- ABHalf

  Pattern A and pattern B are sent alternatively (one instance A, one instance B, and so on).

This command must be used together with the "[SOURce[1]]:PATTern:APCHange:MODE[?] " on page 103 and "[SOURce[1]]:PATTern:APCHange:SOURce[?] " on page 105.

For instructions on how to use these commands, refer to "How the Serial BERT Uses Alternate Patterns" on page 57.

The selection ABHalf is new for the Serial BERT .

[SOURce[1]]:PATTern:APCHange:SOURce[?]

IVI-COM Equivalent	IAgilentN490xPGAuxIn.Source (not IVI-compliant)
Syntax	[SOURce[1]]:PATTern:APCHange:SOURce EXTernal | INTernal | BLANking [SOURce[1]]:PATTern:APCHange:SOURce?
Return Range	EXT | INT | BLAN
*RST Value	EXTernal

Description

This command defines how the Serial BERT determines the pattern to be output. The following alternatives are available:

- INTernal

  Alternate pattern output is determined internally by the instrument (for example, from the user interface or SCPI commands).

- EXTernal

  Alternate pattern output is determined by the signal at **Aux In**. This can either be edge-sensitive or level-sensitive.

- BLANking

  Output can be shut off according to the level at **Aux In**. If **Aux In** high, output is generated, if **Aux In** low, no output.

The query returns the current control of the alternate pattern output.

NOTE

This command must be used together with the "[SOURce[1]]:PATTern:APCHange:MODE[?] " on page 103 and "[SOURce[1]]:PATTern:APCHange:SELect[?] " on page 104.

For instructions on how to use these commands, refer to "How the Serial BERT Uses Alternate Patterns" on page 57.

[SOURce[1]]:PATTern:SEQuence Subnode

This subnode has the following SCPI structure:

```
[SOURce[1]:]
└ :PATTern
   └ :SEQuence
      ├ :DATa[?]
      ├ :EVENt
      ├ :RCL
      ├ :SAVe
      └ :STATe?
```

This subnode has the following commands:

Table 17

Name	Description under
:DATA[?]	"[SOURce [1]]:PATTern:SEQuence:DATA[?] " on page 106
:EVENt	"[SOURce [1]]:PATTern:SEQuence:EVENt " on page 111
:RCL	"[SOURce[1]]:PATTern:SEQuence:RCL " on page 111
:SAVE	"SOURce[1]:PATTern:SEQuence:SAVE " on page 112
:STATe	"[SOURce [1]]:PATTern:SEQuence:STATe? " on page 112

[SOURce[1]]:PATTern:SEQuence:DATA[?]

IVI-COM Equivalent IAgilentN490xPG2.Sequence.Expression

Syntax SOURce[1]:PATTern:SEQuence:DATA <SequenceExpression>

Description

This command is used for generating a user-defined sequence of up to four blocks.

The sequence is defined by a SequenceExpression, which is formulated in its own language. This SequenceExpression can be set up with this command (which performs also the syntax and semantic checks).

The SequenceExpression must be enclosed in parentheses.

Once you have created or changed a SequenceExpression, you can download the sequence to the pattern generator with the command SOURce[1]:PATTern:SELect SEQuence.

NOTE

If the SOURce[1]:PATTern:SEQuence:RCL command is used to recall a saved sequence from a file, the present SequenceExpression is overwritten.

If a user-defined sequence is used, pattern tracking is automatically disabled (see also "SENSe[1]:PATTern:TRACk[?] " on page 193).

During the initialization phase of the user sequence, NRZ and RZ pulse formats generate pure zeros, R1 generates pure ones.

The query returns the current SequenceExpression.

SequenceExpression for User-Defined Sequences

The SequenceExpression specifies:

- the sequence start condition
- the blocks, their contents, and triggers
- the loops

You can inspect the contents of the sequence expression in the Properties window of the Sequence Editor.

The SequenceExpression uses the following keywords:

Version=	optional
Description=	optional
Start=	optional
Block #=	repeated for each block, numbered

Loop=	repeated for every loop, not numbered

The data following a keyword must be terminated by CR/LF or semicolon. In the following description, optional parameters are given in brackets.

Example of a Sequence Expression:

```
Version= 1.0
Start= IMM
Block 1= PRBS11, 1024, TrigOn
Block 2= C:\N4903A\Pattern\Upat10.ptrn
Block 3= P0, 512, TrigOff
Loop= B1, B1, 2
```

Explanation of the Keywords:

Version= Language version to allow future extensions. If no version is entered, version 1.0 is assumed.

Example: `Version= 1.0`

Description= Descriptive text to be stored with the sequence, given as a double-quoted string.

Example: `Description= "Sequence for testing A81397B"`

Start= Sets the start condition of the sequence.

Syntax:

Start= IMM[edate] | AuxInHi | AuxInLo | AuxInRising | AuxInFalling | Man[ual]

Input parameters:

- IMM: Sequence starts immediately after sequence download. This is the default.
- AuxInHi | AuxInLo: Sequence starts when Auxiliary Input is high or low.
- AuxInRising | AuxInFalling: Sequence starts with a rising or falling edge at the Auxiliary Input.
- Manual: Sequence starts when the [SOURce [1]]:PATTern:SEQ:EVENt ONCE command is received.

Example: `Start= AuxInRising`

Block #= Defines the contents of a block and the On/Off of the associated sequence trigger.

Syntax:

Block #= <PatternType> [, TrigOn |, TrigOff]

The range of "#" is 1 to 4.

Input parameters:

- PatternType is one of the following:

  None| PRBS#[, <Length>] | <UserFilename>[,A|,B] | P0[, <Length>] | P1, [<Length>] | CL/#[, <Length>]

  The default length of a block is 512 bits. Therefore, this parameter can be omitted.

  - None: Makes a block empty. Empty blocks may be present, but are completely ignored.
  - PRBS#: Pseudo random bit stream with the specified polynomial n (format 2^n-1).
  - UserFilename: The path and name of the file that contains the pattern (for example, `C:\N4903A\Pattern\Upat1.ptrn`). Flavor A|B: The first or second half of the pattern to be generated from that file (see "How the Serial BERT Uses Alternate Patterns" on page 57 for more information).
  - P0: Pause0. This is the default.
  - P1: Pause1.
  - CL/<NR1>: Generates a clock pulse by dividing the system clock frequency. The range of the divider <NR1> is 2 to 127.

    The output starts with ones. For example: CL/2 yields 101010...; CL/4 yields 11001100... . Odd divider factors generate one "1" more than zeros.

- TrigOn | TrigOff specifies whether a trigger pulse shall be generated whenever the execution of the block is started (or repeated). To enable the sequence trigger mode, use the command SOURce3:TRIGger[:MODE] SEQuence.

  The setting of TrigOn | TrigOff is ignored when the Trigger Out port is put into divided clock mode (with the SOURce3:TRIGger DCLock command).

Example: `Block 2= PRBS15, 2048, TrigOn`

NOTE

The block length resolution is 512 bits.

If a user-defined pattern from a file contains less or more bits and the block is part of a counted loop, then <PatternLength> × <LoopCount> must match 512 bits or a multiple thereof.

If a user-defined pattern from a file contains less or more bits and the block is infinitely looped, the pattern is "rolled out". That means, it starts from the beginning until a multiple of 512 bits is generated.

Loop= Defines a loop. The keyword must be repeated for every loop.

Syntax:

Loop= B<EndBlock#>, B<StartBlock#>, <LoopCondition>

Input parameters:

- EndBlock#, StartBlock#: Integer numbers between 1 and 4.
- LoopCondition: One of the following:

  LoopCount | INF | AuxInHi | AuxInLo | AuxInRising | AuxInFalling | Manual |

  - LoopCount: The number of iterations of a counted loop (NR1).
  - INFinite: Specifies an endless loop. This is the default.
  - AuxInHi | AuxInLo: Loop continues until Auxiliary Input is high or low. Then, sequence execution continues with the next block.
  - AuxInRising | AuxInFalling: Loop continues until Auxiliary Input receives a rising or falling edge. Then, sequence execution continues with the next block.
  - Manual: Loop continues until the command SOURce [1]:PATTern:SEQ:EVENt:ONCE is received. Then, sequence execution continues with the next block.

    You can use the query SOURce[1]:PATTern:SEQuence:STATe? to determine which block is currently executed.

Example: `Loop= B4, B2, 102`

NOTE

Loops always define the transition from the end of a block to the beginning of the same or a previous block. It is not possible to jump into an existing loop. It is also not possible to specify loops within loops (exept the default overall loop).

NOTE

It is possible to interrupt and re-initialize a running sequence. This is done with the command SOURce[1]:PATTern:SEQ:EVENt:RESume.

Whether the sequence execution restarts immediately or waits for an event depends on the sequence start condition.

[SOURce[1]]:PATTern:SEQuence:EVENt

IVI-COM Equivalent	IAgilentN490xPGSequence.Event()
Syntax	SOURce[1]:PATTern:SEQuence:EVENt ONCE | RESume
Description	This command is used to break a loop which is in "manual" mode. It is also used to stop and re-initialize the sequence. For details about LoopConditions see "SequenceExpression for User-Defined Sequences" on page 107.
Input parameters	ONCE: Breaks the loop-the pattern generator proceeds to the next block. RESume: Re-initializes the sequence. Sequence execution restarts as soon as the sequence start condition is met: immediately, triggered by Aux In, or upon command.

[SOURce[1]]:PATTern:SEQuence:RCL

IVI-COM Equivalent	IAgilentN490xPGSequence.Load()
Syntax	SOURce[1]:PATTern:SEQuence:RCL <FileIdentifier>
Description	This command recalls (loads) a sequence that has been stored in a file by means of the SOURce[1]:PATTern:SEQuence:SAVE command. The FileIdentifier must include path and file name (for example, `C:\N4903A\Sequences\seq01.seq`).

NOTE

The contents of the specified file overwrites the present SequenceExpression. See also "SequenceExpression for User-Defined Sequences" on page 107. You can download the new SequenceExpression to the pattern generator with the command SOURce[1]:PATTern:SELect SEQuence.

Recall may fail if the SequenceExpression references a pattern file that is not available.

SOURce[1]:PATTern:SEQuence:SAVE

IVI-COM Equivalent IAgilentN490xPGSequence.Save()

Syntax SOURce[1]:PATTern:SEQuence:SAVE <FileIdentifier>

Description This command saves a SequenceExpression in a file.

The FileIdentifier must include path and file name (for example, `C:\N4903A\Sequences\seq01.seq`).

NOTE The standard extension of a sequence file is `.seq`. The file is ASCII-coded and contains the SequenceExpression. See also "SequenceExpression for User-Defined Sequences" on page 107.

[SOURce[1]]:PATTern:SEQuence:STATe?

IVI-COM Equivalent IAgilentN490xPGSequence.State()

Syntax SOURce[1]:PATTern:SEQuence:STATe?

Description This query returns the number of the sequence block that is currently executed. It can be used to determine whether the command SOURce[1]:PATTern:SEQuence:EVENt is adequate.

Query results:

-1	No user sequence activated
0	Sequence start condition not fulfilled (init state, no data)
1 ... 4	The number of the block currently executed

NOTE

The pattern generator is polled every 200 ms. The answer can be incorrect if the execution time of a block is not sufficiently long.

The transition from one block to another is also signaled with a flag in the status subsystem (bit #15 in the Operation Status Register).

[SOURce[1]]:PATTern:UFILe Subnode

This subnode has the following SCPI structure:

```
[SOURce[1]:]
└ PATTern
   └ :UFILe
      ├ :DATA[?]
      ├ :IDATa[?]
      ├ [:LENGth][?]
      ├ :LABel[?]
      ├ :NAME?
      └ :USE[?]
```

This subnode has the following commands:

Table 20

Name	Description under
:DATA[?]	“[SOURce[1]]:PATTern:UFILe:DATA[?] ” on page 114
:IDATA[?]	“[SOURce[1]]:PATTern:UFILe:IDATa ” on page 115
[:LENGth][?]	“[SOURce[1]]:PATTern:UFILe[:LENGth][?] ” on page 117
:LABel[?]	“[SOURce[1]]:PATTern:UFILe:LABel[?] ” on page 117
:NAME?	“[SOURce[1]]:PATTern:UFILe:NAME? ” on page 117
:USE[?]	“[SOURce[1]]:PATTern:UFILe:USE[?] ” on page 118

[SOURce[1]]:PATTern:UFILe:DATA[?]

IVI-COM Equivalent IAgilentN490xLocalPatternfile.SetData (IVI-compliant)

Syntax [SOURce[1]]:PATTern:UFILe:DATA [A|B,] <filename>, <block data>

[SOURce[1]]:PATTern:UFILe:DATA? [A|B,] <filename>

Return Range The query returns the standard (A) or alternate pattern (B) of the file found under <filename>.

Description This command is used to set the bits in user pattern files. See "Working with User Patterns in SCPI" on page 65for a detailed description on how to edit user patterns.

The parameters have the following meanings:

Table 21

Parameter	Description
[A|B]	Defines for which pattern the data is to be set (A = standard pattern, B = alternate pattern). If the pattern file describes a standard pattern (:USE = STRaight), this parameter cannot be B.
<filename>	Name of the file being defined. If the file does not exist, it is created.
<block data>	The data that describes the pattern (see the following for the description).

<block data> The <block data> parameter contains the actual data for setting the bits of the user pattern. The bits can also be packed using the FORMat[:DATA] command. If the bits are not packed, they are handled as 8-bit data. See "[SOURce[1]]:PATTern:FORMat[:DATA][?] " on page 98.

This command also sets the pattern length to fit the length of the data: If the data block is longer than the pattern, the pattern is extended to fit the data; if the data block is shorter than the pattern, the pattern is truncated to the end of the data.

<block data> starts with a header that indicates the length of the desired resulting data. The length of the <block data> embedded in the header always refers to the length of the data block in bytes.

For example, consider the following header:

#19<data>

#	Start of the header.
1	Number of decimal digits to follow to form the length.
9	Length of the data block (in bytes) that follows.
<data>	The pattern data, packed according the DATA:PACKed command.

- For non-packed data (or 8-bit packed data), the <block data> required to set an 8-bit pattern of alternating 1s and 0s (01010101) would be:

  #11U (Note that "U" is the ASCII representation of 85)

- For 4-bit packed data, the <block data> required to set the same pattern would be:

  #1255

- For 1-bit packed data, the <block data> would be as follows:

  #1801010101

[SOURce[1]]:PATTern:UFILe:IDATa

IVI-COM Equivalent IAgilentN490xLocalPatternfile.SetDataBlock (IVI-compliant)

Syntax [SOURce[1]]:PATTern:UFILe:IDATa [A | B,] <filename>, <start_bit>, <length_in_bits>, <block_data>

[SOURce[1]]:PATTern:UFILe:IDATa? [A | B,] <filename>, <start_bit>, <length_in_bits>

Return Range The query returns the selected bits of the standard (A) or alternate (B) pattern of the file found under <filename>.

Description This command is used to set specific bits in a user pattern. It is similar to the :DATA command. The :IDATa command is a contraction of the phrase **I**ncremental **DATA** and is used to download a part of a user-defined pattern.

The parameters have the following meanings:

Table 23

Parameter	Description
[A|B]	Defines for which pattern the data is to be set (A = standard pattern, B = alternate pattern). If the pattern file describes a standard pattern (:USE = STRaight), this parameter cannot be B.
<filename>	Name of the file being defined. If the file does not exist, it is created.
<start bit>	First bit to be overwritten (starting with 0).
<length_in_bits>	Number of bits to be overwritten.
<block data>	The data that describes the pattern (see "[SOURce[1]]:PATTern:UFILe:DATA[?] " on page 114 for the description).

The use of the parameters can be best illustrated by an example. If we have an alternate 16-bit pattern of 0s only, and we want to set the last four bits to 1s, the IDATa command would appear as follows:

- If the data packing is 8:

  SOURce1:PATTern:UFILe:IDATa B, <filename>, 12, 4, #11(&F0) (where (&F0) is replaced by the ASCII representation of the value)

- If the data packing is 4:

  SOURce1:PATTern:UFILe:IDATa B, <filename>, 12, 4, #11F

- If the data packing is 1:

  SOURce1:PATTern:UFILe:IDATa B, <filename>, 12, 4, #141111

The response form returns <block data> at the specified location.

NOTE

See "Working with User Patterns in SCPI" on page 65 for more information on using this command.

[SOURce[1]]:PATTern:UFILe[:LENGth][?]

IVI-COM Equivalent IAgilentN490xLocalPatternfile.Length (IVI-compliant)

Syntax [SOURce[1]]:PATTern:UFILe[:LENGth] <filename>, <numeric_value>

[SOURce[1]]:PATTern:UFILe[:LENGth]? <filename>

Description This command sets the length of a user pattern file. The query returns the length of the user pattern file. If an alternate pattern is selected (:USE APATtern), the LENGth command sets the length of each half of the pattern.

Note that the :DATA command automatically sets the length of the file.

See “Working with User Patterns in SCPI” on page 65 for information on using this command.

[SOURce[1]]:PATTern:UFILe:LABel[?]

IVI-COM Equivalent IAgilentN490xLocalPatternfile.Description (IVI-compliant)

Syntax [SOURce[1]]:PATTern:UFILe:LABel <filename>, <string>

[SOURce[1]]:PATTern:UFILe:LABel? <filename>

Description This command sets a description for a user pattern file. The query returns the description. See “Working with User Patterns in SCPI” on page 65 for information on using this command.

[SOURce[1]]:PATTern:UFILe:NAME?

IVI-COM Equivalent IAgilentN490xLocalPatternfile.Location (IVI-compliant)

Syntax [SOURce[1]]:PATTern:UFILe:NAME?

Description This query returns the file name of the currently used user pattern. It is only valid if SOURce1:PATTern:SELect? returns UPAT.

[SOURce[1]]:PATTern:UFILe:USE[?]

IVI-COM Equivalent IAgilentN490xLocalPatternfile.Alternate (IVI-compliant)

Syntax [SOURce[1]]:PATTern:UFILe:USE <filename>, STRaight | APATtern

[SOURce[1]]:PATTern:UFILe:USE? <filename>

Return Range STR | APAT

Description This command defines whether a user pattern file should be a straight pattern or an alternate pattern:

- STRaight

  The pattern is repeatedly output.

- APATtern

  The pattern is composed of two halves. The output depends on various other commands; see "How the Serial BERT Uses Alternate Patterns" on page 57 for more information.

The default is set to have a length of 128 bits for each half pattern; all bits are set to zero and the trigger is set to occur on the A/B changeover. See "Working with User Patterns in SCPI" on page 65 for information on using this command.

[SOURce[1]]:PATTern:UPATTern Subnode

This subnode has the following SCPI structure:

```
[SOURce[1]:]
└ :PATTern
  └ :UPATtern<n>
    ├ [:LENGth][?]
    ├ :LABel[?]
    ├ :USE[?]
    ├ :DATA[?]
    └ :IDATa[?]
```

This subnode has the following commands:

Table 24

Name	Description under
[:LENGth][?]	"[SOURce[1]]:PATTern:UPATtern<n>[:LENGth][?] " on page 119
:LABel[?]	"[SOURce [1]]:PATTern:UPATtern<n>:LABel[?] " on page 120
:USE[?]	"[SOURce [1]]:PATTern:UPATtern<n>:USE[?] " on page 120
:DATA[?]	"[SOURce [1]]:PATTern:UPATtern<n>:DATA[?] " on page 120
:IDATa[?]	"[SOURce [1]]:PATTern:UPATtern<n>:IDATa[?] " on page 122

NOTE

For the UPATtern<n> commands, <n> can be in the range 0 - 12. 0 (zero) is used to select the current pattern, 1 - 12 selects one of the user patterns in the memory.

[SOURce[1]]:PATTern:UPATtern<n>[:LENGth][?]

IVI-COM Equivalent IAgilentN490xPGPatternfile.Length (IVI-compliant)

Syntax [SOURce[1]]:PATTern:UPATtern<n>[:LENGth] <numeric value>

[SOURce[1]]:PATTern:UPATtern<n>[:LENGth]?

Description This command sets the length of the selected user pattern. The query returns the length of the user pattern. If an alternate pattern is selected (:USE APATtern), the LENGth command sets the length of each half of the pattern.

Note that the :DATA command automatically sets the length of the pattern.

See "Working with User Patterns in SCPI" on page 65 for information on using this command.

[SOURce[1]]:PATTern:UPATtern<n>:LABel[?]

IVI-COM Equivalent IAgilentN490xPGPatternfile.Description (IVI-compliant)

Syntax [SOURce[1]]:PATTern:UPATtern<n>:LABel <string>

[SOURce[1]]:PATTern:UPATtern<n>:LABel?

Description The command sets the description of the pattern. The query returns the description of the pattern.

See “Working with User Patterns in SCPI” on page 65 for information on using this command.

[SOURce[1]]:PATTern:UPATtern<n>:USE[?]

IVI-COM Equivalent IAgilentN490xPGPatternfile.Alternate (IVI-compliant)

Syntax [SOURce[1]]:PATTern:UPATtern<n>:USE STRaight | APATtern

[SOURce[1]]:PATTern:UPATtern<n>:USE?

Return Range STR | APAT

Description This command defines whether a user pattern file should be a straight pattern or an alternate pattern:

- STRaight

  The pattern is repeatedly output.

- APATtern

  The pattern is composed of two halves. The output depends on various other commands; see “How the Serial BERT Uses Alternate Patterns” on page 57 for more information.

The default is set to have a length of 128 bits for each half pattern; all bits are set to zero and the trigger is set to occur on the A/B changeover. See “Working with User Patterns in SCPI” on page 65 for information on using this command.

[SOURce[1]]:PATTern:UPATtern<n>:DATA[?]

IVI-COM Equivalent IAgilentN490xPGPatternfile.SetData (IVI-compliant)

Syntax

[SOURce[1]]:PATTern:UPATtern<n>:DATA [A | B,] <block_data>

[SOURce[1]]:PATTern:UPATtern<n>:DATA? [A|B,]

Return Range

The query returns the block data for pattern A or pattern B.

Description

This command is used to set the bits in user pattern files. See "Working with User Patterns in SCPI" on page 65 for a detailed description on how to edit user patterns.

The parameters have the following meanings:

Table 25

Parameter	Description
[A|B]	Defines for which pattern the data is to be set (A = standard pattern, B = alternate pattern). If the pattern file describes a standard pattern (:USE = STRaight), this parameter cannot be B.
<filename>	Name of the file being defined. If the file does not exist, it is created.
<block data>	The data that describes the pattern (see the following for the description).

<block data>

The <block data> parameter contains the actual data for setting the bits of the user pattern. The bits can also be packed using the FORMat[:DATA] command. If the bits are not packed, they are handled as 8-bit data. See "[SOURce[1]]:PATTern:FORMat[:DATA][?]" on page 98.

This command also sets the pattern length to fit the length of the data: If the data block is longer than the pattern, the pattern is extended to fit the data; if the data block is shorter than the pattern, the pattern is truncated to the end of the data.

<block data> starts with a header that indicates the length of the desired resulting data. The length of the <block data> embedded in the header always refers to the length of the data block in bytes.

For example, consider the following header:

- #19<data>

#	Start of the header.
1	Number of decimal digits to follow to form the length.
9	Length of the data block (in bytes) that follows.
<data>	The pattern data, packed according the DATA:PACKed command.

For non-packed data (or 8-bit packed data), the <block data> required to set an 8-bit pattern of alternating 1s and 0s (01010101) would be:

- #11U (Note that "U" is the ASCII representation of 85)

For 4-bit packed data, the <block data> required to set the same pattern would be:

- #1255

For 1-bit packed data, the <block data> would be as follows:

- #1801010101

[SOURce[1]]:PATTern:UPATtern<n>:IDATa[?]

IVI-COM Equivalent IAgilentN490xPGPatternfile.SetDataBlock (IVI-compliant)

Syntax [SOURce[1]]:PATTern:UFILe:IDATa [A | B,] <start bit>, <length in bits>, <block data>

[SOURce[1]]:PATTern:UFILe:IDATa? [A|B,] <start bit>, <length in bits>

Return Range The query returns the selected bits of the standard (A) or alternate (B) pattern.

Description This command is used to set specific bits in a user pattern. It is similar to the :DATA command. The :IDATa command is a contraction of the phrase **I**ncremental **DATA** and is used to download part of a user-defined pattern.

The parameters have the following meanings:

Table 27

Parameter	Description
[A|B]	Defines for which pattern the data is to be set (A = standard pattern, B = alternate pattern). If the pattern file describes a standard pattern (:USE = STRaight), this parameter cannot be B.
<filename>	Name of the file being defined. If the file does not exist, it is created.
<start bit>	First bit to be overwritten (starting with 0).
<length_in_bits>	Number of bits to be overwritten.
<block data>	The data that describes the pattern (see "[SOURce[1]]:PATTern:UFILe:DATA[?] " on page 114 for the description).

The use of the parameters can be best illustrated by an example. If we have an alternate 16-bit pattern of 0s only, and we want to set the last four bits to 1s, the IDATa command would appear as follows:

- If the data packing is 8:

  SOURce1:PATTern:UPAT1:IDATa B, <filename>, 12, 4, #11(&F0) (where (&F0) is replaced by the ASCII representation of the value)

- If the data packing is 4:

  SOURce1:PATTern:UPAT1:IDATa B, <filename>, 12, 4, #11F

- If the data packing is 1:

  SOURce1:PATTern:UPAT1:IDATa B, <filename>, 12, 4, #141111

The response form returns <block data> at the specified location.

NOTE

See "Working with User Patterns in SCPI" on page 65 for more information on using this command.

[SOURce[1]]:VOLTage Subnode

This subnode has the following SCPI structure:

```
[SOURce[1]:]
└ :VOLTage
   ├ :ECL
   └ [:LEVel]
      ├ [:IMMediate]
      │  ├ [:AMPLitude][?]
      │  ├ :HIGH[?]
      │  ├ :LOW[?]
      │  └ :OFFSet[?]
      └ :LLEVel[?]
```

This subnode has the following commands:

Table 28

Name	Description under
:ECL	"[SOURce[1]]:VOLTage:ECL " on page 124
[:LEVel][:IMMediate][:AMPLitude][?]	"[SOURce[1]]:VOLTage[:LEVel][:IMMediate][:AMPLitude][?] " on page 125
[:LEVel][:IMMediate]:HIGH[?]	"[SOURce[1]]:VOLTage[:LEVel][:IMMediate]:HIGH[?] " on page 125
[:LEVel][:IMMediate]:LOW[?]	"[SOURce[1]]:VOLTage[:LEVel][:IMMediate]:LOW[?] " on page 125
[:LEVel][:IMMediate]:OFFSet[?]	"[SOURce[1]]:VOLTage[:LEVel][:IMMediate]:OFFSet[?] " on page 126
[:LEVel]:LLEVel[?]	"[SOURce[1]]:VOLTage[:LEVel]:LLEVel[?] " on page 126

[SOURce[1]]:VOLTage:ECL

IVI-COM Equivalent IAgilentN490xPGOutput.LogicLevel (not IVI-compliant)

Syntax	[SOURce[1]]:VOLTage:ECL
Description	This command sets the data output values to those used for the ECL family. Retained for backwards compatibility. Superseded by SOURce1:VOLTage:LLEVel (see "[SOURce[1]]:VOLTage[:LEVel]:LLEVel[?] " on page 126).

[SOURce[1]]:VOLTage[:LEVel][:IMMediate][:AMPLitude][?]

IVI-COM Equivalent	IAgilentN490xPGOutVoltage.VAmplitude (IVI-compliant)
Syntax	[SOURce[1]]:VOLTage[:LEVel][:IMMediate][:AMPLitude] <Num.> [SOURce[1]]:VOLTage[:LEVel][:IMMediate][:AMPLitude]?
Description	The command sets the peak-to-peak value of the data signal in units of Volts. The query returns the peak-to-peak value of the data signal in units of Volts.

[SOURce[1]]:VOLTage[:LEVel][:IMMediate]:HIGH[?]

IVI-COM Equivalent	IAgilentN490xPGOutVoltage.VHigh (IVI-compliant)
Syntax	[SOURce[1]]:VOLTage[:LEVel][:IMMediate]:HIGH <Num.> [SOURce[1]]:VOLTage[:LEVel][:IMMediate]:HIGH?
Description	The command sets the DC low output level in units of Volts. The query returns the DC low output level in units of Volts.

[SOURce[1]]:VOLTage[:LEVel][:IMMediate]:LOW[?]

IVI-COM Equivalent	IAgilentN490xPGOutVoltage.VLow (IVI-compliant)
Syntax	[SOURce[1]]:VOLTage[:LEVel][:IMMediate]:LOW <Num.> [SOURce[1]]:VOLTage[:LEVel][:IMMediate]:LOW?
Description	The command sets the DC low output level in units of Volts. The query returns the DC low output level in units of Volts.

[SOURce[1]]:VOLTage[:LEVel][:IMMediate]:OFFSet[?]

IVI-COM Equivalent	IAgilentN490xPGOutVoltage.VOffset (IVI-compliant)
Syntax	[SOURce[1]]:VOLTage[:LEVel][:IMMediate]:OFFSet <Num.> [SOURce[1]]:VOLTage[:LEVel][:IMMediate]:OFFSet?
Description	The command sets the mean of the high and low DC output level in units of Volts. The query returns the mean of the high and low DC output level in units of Volts.

[SOURce[1]]:VOLTage[:LEVel]:LLEVel[?]

IVI-COM Equivalent	IAgilentN490xPGOutput.LogicLevel (not IVI-compliant)
Syntax	[SOURce[1]]:VOLTage[:LEVel]:LLEVel <Family> [SOURce[1]]:VOLTage[:LEVel]:LLEVel?
Input Parameters	**<Family>**: ECL | LVPECL | SCFL | LVDS | CML | CUSTom
Return Range	ECL | LVPECL | SCFL | LVDS | CML | CUST

NOTE Selecting CUSTom has no effect.

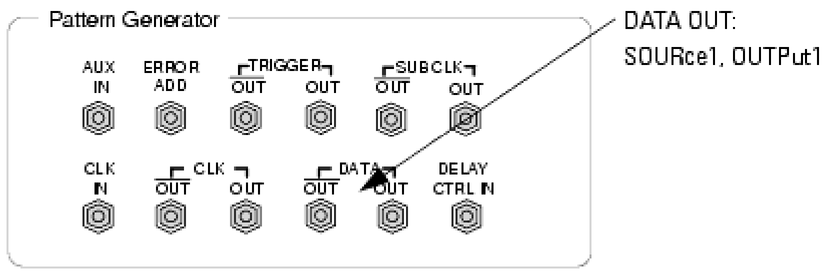
Description	The command sets the output level appropriate for the specified logic family. The query returns the currently used logic family.

NOTE If any of the voltage parameters have been modified, CUSTom will be returned by the query, even if the parameter has been set back to the default.

OUTPut[1] Subsystem

OUTPut[1] Subsystem - Reference

The Output[1] subsystem represents the pattern generator's Data Out port.



This subsystem has the following SCPI structure:

```
OUTPut[1]
 ├ :CENTer
 ├ :COUPling[?]
 ├ :DATA
 │  └ :XOVer[?]
 ├ :DCYCle[?]
 ├ :DELay[?]
 ├ :FORMat[?]
 ├ :HOLD[?]
 ├ :POLarity[?]
 ├ [:STATe][?]
 ├ :TERMination[?]
 └ :WIDTh[?]
```

This subsystem has the following commands:

Table 29

Name	Description under
:CENTer	"OUTPut[1]:CENTer " on page 128

Table 29

Name	Description under
:COUPling[?]	"OUTPut[1]:COUPling " on page 128
:DATA:XOVer[?]	"OUTPut[1]:DATA:XOVer[?] " on page 129
:DCYCle[?]	"OUTPut[1]:DCYCle[?] " on page 129
:DELay[?]	"OUTPut[1]:DELay[?] " on page 129
:FORMat[?]	"OUTPut[1]:FORMat[?] " on page 130
:HOLD[?]	"OUTPut[1]:HOLD[?] " on page 130
:POLarity[?]	"OUTPut[1]:POLarity[?] " on page 131
[:STATe][?]	"OUTPut[1][:STATe][?] " on page 131
:TERMination[?]	"OUTPut[1]:TERMination[?] " on page 131
:WIDTh[?]	"OUTPut[1]:WIDTh[?] " on page 132

OUTPut[1]:CENTer

IVI-COM Equivalent IAgilentN490xPGGlobal.OutputsDisconnect (not IVI-compliant)

Syntax OUTPut[1]:CENTer DISConnect | CONNect

Description The DISConnect command sets the voltage at the pattern generator's Data Out and Clock Out ports to 0 V, the CONNect command re-enables the output (to the normal data pattern).

OUTPut[1]:COUPling

IVI-COM Equivalent IAgilentN490xPGOutput.TerminationEnabled (IVI-compliant)

Syntax OUTPut[1]:COUPling AC | DC

OUTPut[1]:COUPling?

Description The command enables or disables the source of the termination voltage:

- DC: Enables the termination voltage
- AC: Disables the termination voltage

The query returns the current state.

OUTPut[1]:DATA:XOVer[?]

IVI-COM Equivalent IAgilentN490xPGOutput.Crossover (IVI-compliant)

Syntax OUTPut[1]:DATA:XOVer <NR1.>

OUTPut[1]:DATA:XOVer? [MINimum | MAXimum]

Description The command sets the eye crossover of the pattern generator's Data Out port. Crossover can only be changed in NRZ signal mode.

The query returns the current crossover setting.

OUTPut[1]:DCYCle[?]

IVI-COM Equivalent IAgilentBertPGPulse.DutyCycle

Syntax OUTPut[1]:DCYCle <NR1>

OUTPut[1]:DCYCle?

Description Sets the duty cycle of a repetitive pulse waveform (like in RZ or R1 signal modes). Duty cycle value <NR1> in % of the clock period. Valid range is 0 ... 100, default is 50.

The query returns the current setting.

OUTPut[1]:DELay[?]

IVI-COM Equivalent IAgilentN490xPGOutput.Delay (IVI-compliant)

Syntax OUTPut[1]:DELay <Num.>

OUTPut[1]:DELay?

Description

This command sets the delay of the active edge of the clock output relative to the pattern generator's Data Out port. The units are seconds. The value is rounded to the nearest one picosecond. The response returns the current data to clock delay value.

This command has restrictions for frequencies under 620 Mbits/s. See for details.   See the User Guide (or online Help) for details.

OUTPut[1]:FORMat[?]

IVI-COM Equivalent

IAgilentBertPGOutput.SignalMode

Syntax

OUTPut[1]:FORMat <NRZ | RZ | R1>

OUTPut[1]:FORMat?

Description

Sets the pulse format ("Signal Mode") to NRZ, RZ, or R1.

The query returns the current setting as a string.

NOTE

If you wish to use the RZ or R1 formats in conjunction with a directly supplied external clock (not a reference clock), you must specify the external clock as "manual" (instead of "automatic"). For details see "SENSe6:MODE " on page 150.

In RZ or R1 mode, the crossover cannot be changed.

OUTPut[1]:HOLD[?]

IVI-COM Equivalent

IAgilentBertPGPulse.DelayHoldMode

Syntax

OUTPut[1]:HOLD <WIDTh | DCYCle>

OUTPut[1]:HOLD?

Description

Determines whether Width or Duty Cycle shall be kept, if a repetitive pulse waveform (as in RZ or R1 signal modes) is enabled and the generator's signal frequency is changed.

If this command is used to switch from "Hold Width" to "Hold Duty Cycle", the current pulse width is converted to a percentage of the present signal period.

If this command is used to switch from "Hold Duty Cycle" to "Hold Width", the current duty cycle is converted from percent to seconds, according to the present signal period.

The query returns the current setting as a string.

OUTPut[1]:POLarity[?]

IVI-COM Equivalent IAgilentN490xPGOutput.Polarity (IVI-compliant)

Syntax OUTPut[1]:POLarity NORMal | INVerted

OUTPut[1]:POLarity?

Return Range NORM | INV

Description The command sets the polarity of the pattern generator's Data Out port. The query returns the current polarity of the pattern generator's Data Out port.

OUTPut[1][:STATe][?]

IVI-COM Equivalent IAgilentN490xPGOutput.Enabled (IVI-compliant)

Syntax OUTPut[1][:STATe] 0 | 1 | OFF | ON

OUTPut[1][:STATe]?

Description This command is kept for compatibility reasons. Due to the lack of relays, it is not possible to disable the Serial BERT 's output. This command has no effect. The query will always return "1" = ON.

To force the Data Out and Clock Out ports to 0 V, use the command "OUTPut[1]:CENTer " on page 128.

OUTPut[1]:TERMination[?]

IVI-COM Equivalent IAgilentN490xPGOutVoltage.VTermination (IVI-compliant)

Syntax OUTPut[1]:TERMination <NR3>

OUTPut[1]:TERMination?

Description

This command sets the data termination level of the pattern generator's Data Out port. The response form returns the data termination level.

This command is only valid if the coupling is set to DC (see "OUTPut[1]:COUPling" on page 128).

OUTPut[1]:WIDTh[?]

IVI-COM Equivalent

IAgilentBertPGPulse.Width

Syntax

OUTPut[1]:WIDTh <NR3>

OUTPut[1]:WIDTh?

Description

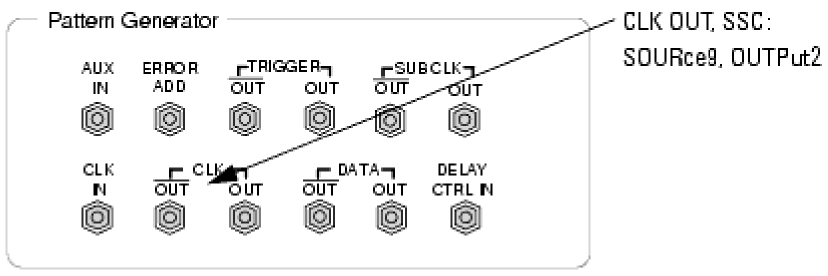
Sets the width or duration of a repetitive pulse (like in RZ or R1 signal modes). The width value is given in seconds (<NR3>). Valid range is 25 ps (e.g. 25 e-12) to one clock period minus 25 ps.

The query returns the current setting.

SOURce9 Subsystem

SOURce9 Subsystem - Reference

The SOURce9 Subsystem represents the pattern generator's Clock Out port. It is also used to control the optional Spread Spectrum Clocking function (SSC).



This subsystem has the following SCPI structure:

```
SOURce9
├─ :FREQuency
│   └─ [:CW|:FIXed][?]
├─ :OUTPut
│   └─ [:STATe][?]
└─ :SSCLocking
    ├─ [:STATE][?]
    ├─ :DEViation[?]
    └─ :FREQuency[?]
```

This subsystem has the following commands:

Table 30

Name	Description under
:FREQuency[:CW|:FIXed][?]	"SOURce9:FREQuency[:CW|FIXed][?] " on page 133
:OUTPut[:STATe][?]	"SOURce9:OUTPut[:STATe][?] " on page 134
:SSCLocking[:STATe][?]	"SOURce9:SSCLocking[:STATe][?] " on page 134
:SSCLocking:DEViation[?]	"SOURce9:SSCLocking:DEViation[?] " on page 134
:SSCLocking:FREQuency[?]	"SOURce2:FREQuency[:CW|:FIXed]? " on page 137

SOURce9:FREQuency[:CW|FIXed][?]

IVI-COM Equivalent IAgilentN490xPGClock.Frequency (IVI-compliant)

Syntax SOURce9:FREQuency[:CW|:FIXed] <Num.>

SOURce9:FREQuency[:CW|:FIXed]? <Num.>| <MIN | MAX>

Description

This command may be used to configure the internal clock source frequency. You can also use any of the forms listed below:

- SOURce9:FREQuency
- SOURce9:FREQuency:CW
- SOURce9:FREQuency:FIXed

There is no difference between any of these forms.

The response returns the current internal clock source frequency.

SOURce9:OUTPut[:STATe][?]

IVI-COM Equivalent

IAgilentN490xPGClockIn.Mode (IVI-compliant)

Syntax

SOURce9:OUTPut[:STATe] EXT | INT

SOURce9:OUTPut[:STATe]?

Description

This command can be used to switch the pattern generator's clock generator input from internal to external mode. It is provided for compatibility reasons and is identical with "SENSe6:MODE " on page 150. The latter should be preferred.

SOURce9:SSCLocking[:STATe][?]

IVI-COM Equivalent

IAgilentN490xPGSpreadSpectrumEnable

Syntax

SOURce9:SSCLocking[:STATe] ON | OFF | 1 | 0

SOUR9:SSCL?

Description

Turns the spread spectrum clock modulation on or off.

The query returns the present setting (0 | 1).

SOURce9:SSCLocking:DEViation[?]

IVI-COM Equivalent

IAgilentN490xPGSpreadSpectrum.ModulationRange

Syntax

SOURce9:SSCLocking:DEViation <NR3>

SOUR9:SSCL:DEV? MIN | MAX

Description
Specifies the maximum deviation from the currently set clock rate towards lower frequencies.

The query returns the present setting or the applicable min/max values.

SOURce9:SSCLocking:FREQuency[?]

IVI-COM Equivalent
IAgilentN490xPGSpreadSpectrum.ModulationFrequency

Syntax
SOURce9:SSCLocking:FREQuency <NR3>

SOURce9:SIN:FREQ? MIN | MAX

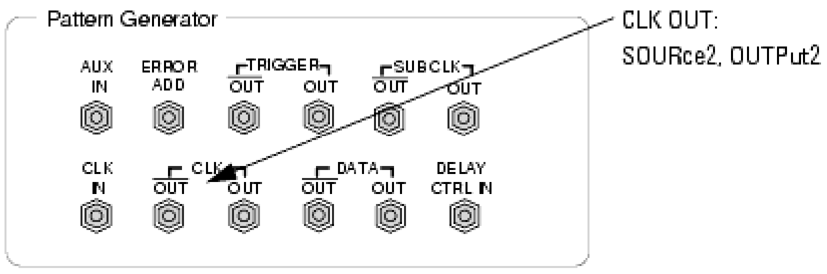
Description
Sets the frequency of the SSC in Hz.

The query returns the present setting or the applicable min/max values.

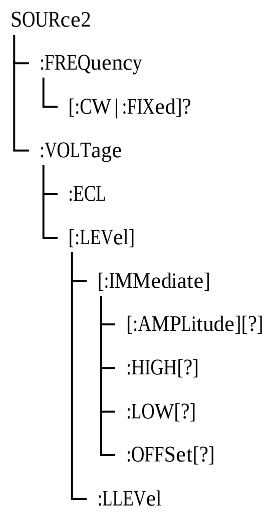
SOURce2 Subsystem

SOURce2 Subsystem - Reference

The SOURce2 Subsystem represents the pattern generator's Clock Out port.



This subsystem has the following SCPI structure:



This subsystem has the following commands:

Table 31

Name	Description under
:FREQuency[:CW|FIXed]?	"SOURce2:FREQuency[:CW|:FIXed]? " on page 137
:VOLTage:ECL	"SOURce2:VOLTage:ECL " on page 137
:VOLTage[:LEVEL][:IMMediate][:AMPLitude][?]	"SOURce2:VOLTage[:LEVel][:IMMediate][:AMPLitude][?] " on page 137
:VOLTage[:LEVEL][:IMMediate]:HIGH[?]	"SOURce2:VOLTage[:LEVel][:IMMediate]:HIGH[?] " on page 138
:VOLTage[:LEVEL][:IMMediate]:LOW[?]	"SOURce2:VOLTage[:LEVel][:IMMediate]:LOW[?] " on page 138
:VOLTage[:LEVEL][:IMMediate]:OFFSet[?]	"SOURce2:VOLTage[:LEVel][:IMMediate]:OFFSet[?] " on page 138
:VOLTage[:LEVEL]:LLEVel	"SOURce2:VOLTage:LLEVel[?] " on page 138

SOURce2:FREQuency[:CW|:FIXed]?

IVI-COM Equivalent	IAgilentN490xPGClockIn.GetFrequency (IVI-compliant)
Syntax	SOURce2:FREQuency[:CW|:FIXed]? [MIN|MAX]
Description	This query returns the bit rate of the measured frequency from internal or external clock.

NOTE: This query is superseded by SENSe6:FREQuency [:CW|:FIXed]?

SOURce2:VOLTage:ECL

IVI-COM Equivalent	IAgilentN490xPGClock.LogicLevel (not IVI-compliant)
Syntax	SOURce2:VOLTage:ECL
Description	Sets the output AMPLitude and HIGH values to those used for the ECL family. There is no query form for this command. This command is provided for backwards compatibility only and is superseded by SOURce2:VOLTage:LLEVel (see "SOURce2:VOLTage:LLEVel[?] " on page 138).

SOURce2:VOLTage[:LEVel][:IMMediate][:AMPLitude][?]

IVI-COM Equivalent	IAgilentN490xPGClockVoltage.VAmplitude (IVI-compliant)
Syntax	SOURce2:VOLTage [:LEVel][:IMMediate][:AMPLitude] <Num.> SOURce2:VOLTage [:LEVel][:IMMediate][:AMPLitude]?
Description	The command sets the peak to peak value of the Clock Out signal in units of Volts. The query returns the peak to peak value of the Clock signal in units of Volts.

SOURce2:VOLTage[:LEVel][:IMMediate]:HIGH[?]

IVI-COM Equivalent IAgilentN490xPGClockVoltage.VHigh (IVI-compliant)

Syntax SOURce2:VOLTage[:LEVel][:IMMediate]:HIGH <Num.>

SOURce2:VOLTage[:LEVel][:IMMediate]:HIGH?

Description The command sets the DC high output level of the pattern generator's Clock Out port in Volts. The query returns the DC high output level of the pattern generator's Clock Out port in Volts.

SOURce2:VOLTage[:LEVel][:IMMediate]:LOW[?]

IVI-COM Equivalent IAgilentN490xPGClockVoltage.VLow (IVI-compliant)

Syntax SOURce2:VOLTage[:LEVel][:IMMediate]:LOW <Num.>

SOURce2:VOLTage[:LEVel][:IMMediate]:LOW?

The command sets the DC low output level of the pattern generator's Clock Out port in Volts. The query returns the DC low output level of the pattern generator's Clock Out port in Volts.

SOURce2:VOLTage[:LEVel][:IMMediate]:OFFSet[?]

IVI-COM Equivalent IAgilentN490xPGClockVoltage.VOffset (IVI-compliant)

Syntax SOURce2:VOLTage[:LEVel][:IMMediate]:OFFSet <Num.>

SOURce2:VOLTage[:LEVel][:IMMediate]:OFFSet?

Description The command sets the offset value of the pattern generator's Clock Out port in Volts. The query returns the offset value of the pattern generator's Clock Out port in Volts.

SOURce2:VOLTage:LLEVel[?]

IVI-COM Equivalent IAgilentN490xPGClockVoltage.LogicLevel (not IVI-compliant)

Syntax SOURce2:VOLTage:LLEVel <Family>

SOURce2:VOLTage:LLEVel?

Input Parameters	**<Family>**: ECL | LVPECL | SCFL | LVDS | CML | CUSTom

NOTE Selecting CUSTom has no effect.

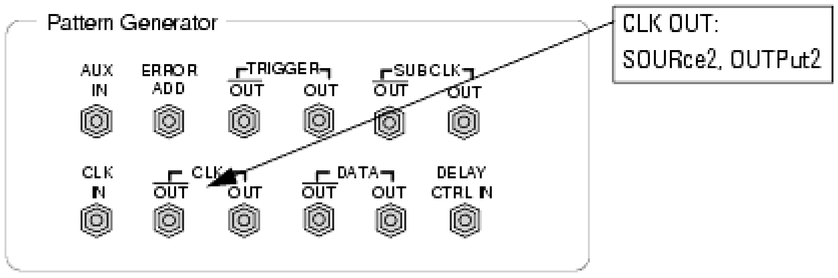
Return Range	ECL | LVPECL | SCFL | LVDS | CML | CUSTom
Description	The command sets the output level appropriate for the specified logic family. The query returns the currently used logic family.

NOTE If any of the voltage parameters have been modified, CUSTom will be returned by the query, even if the parameter has been set back to the default.

OUTPut2 Subsystem

OUTPut2 Subsystem - Reference

The OUTPut2 Subsystem represents the pattern generator's Clock Out port.



This subsystem has the following SCPI structure:

```
OUTput2
 ├ :CENTer
 ├ :COUPling[?]
 ├ [:STATe][?]
 └ :TERMination[?]
```

This subsystem has the following commands:

Table 32

Name	Description under
:CENTer	“OUTPut2:CENTer ” on page 140
:COUPling[?]	“OUTPut2:COUPling[?] ” on page 140
[:STATe][?]	“OUTPut2[:STATe][?] ” on page 141
:TERMination[?]	“OUTPut2:TERMination[?] ” on page 141

OUTPut2:CENTer

IVI-COM Equivalent IAgilentN490xPGGlobal.OutputsDisconnect (not IVI-compliant)

Syntax OUTPut2:CENTer DISConnect | CONNect

Description The DISConnect command sets the voltage at the pattern generator's Clock Out and Data Out ports to 0 V, the CONNect command re-enables the output (to the normal data pattern).

This command is identical with “OUTPut[1]:CENTer ” on page 128.

OUTPut2:COUPling[?]

IVI-COM Equivalent IAgilentN490xPGClockVoltageTerminationEnabled

Syntax OUTPut2:COUPling AC | DC

OUTPut2:COUPling?

Description

The command enables or disables the source of the termination voltage:

- DC: Enables the termination voltage
- AC: Disables the termination voltage

The query returns the current state.

OUTPut2[:STATe][?]

IVI-COM Equivalent

IAgilentN490xPGClock2Enabled

Syntax

OUTPut2[:STATe] 0 | 1 | OFF | ON

OUTPut2[:STATe]?

Description

This command is kept for compatibility reasons. Due to the lack of relays, it is not possible to disable the Serial BERT 's output. This command has no effect. The query will always return "1" = ON.

To force the Data Out and Clock Out ports to 0 V, use the command “OUTPut[1]:CENTer ” on page 128.

OUTPut2:TERMination[?]

IVI-COM Equivalent

IAgilentN490xPGClockVoltageVTermination

Syntax

OUTPut2:TERMination 0 | -2 | 1.3

OUTPut2:TERMination?

Description

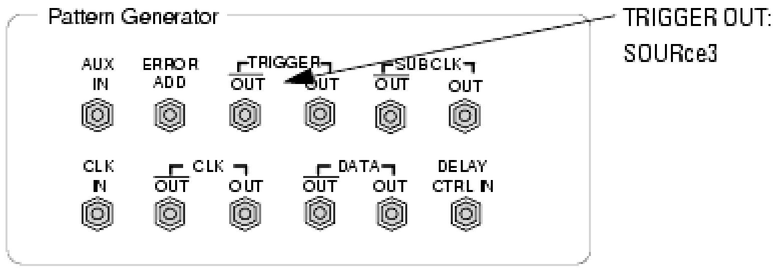
This command sets the data termination level of the pattern generator's Clock Out port. The response form returns the data termination level.

This command is only valid if the coupling is set to DC (see “OUTPut2:COUPling[?] ” on page 140).

SOURce3 Subsystem

SOURce3 Subsystem - Reference

The SOURce3 Subsystem represents the pattern generator's Trigger Out port.



This subsystem has the following SCPI structure:

```
SOURce3
└ :TRIGger
   ├ [:MODE][?]
   ├ :DCDRatio
   ├ :CTDRatio?
   ├ :APATtern<n>[?]
   ├ :MDENsity<n>[?]
   ├ :ZSUBstitut<n>[?]
   ├ :PRBN<n>[?]
   ├ :PRBS<n>[?]
   └ :UPATtern<n>
```

This subsystem has the following commands:

Table 33

Name	Description under
:TRIGger[:MODE][?]	"SOURce3:TRIGger[:MODE][?] " on page 143

Table 33

Name	Description under
:TRIGger:DCDRatio	"SOURce3:TRIGger:DCDRatio " on page 144
:TRIGger:CTDRatio	"SOURce3:TRIGger:CTDRatio? " on page 144
:TRIGger:APATtern<n>[?]	"SOURce3:TRIGger:APATtern<n>[?] " on page 144
:TRIGger:MDENsity<n>[?]	"SOURce3:TRIGger:MDENsity<n>[?] " on page 145
:TRIGger:ZSUBstitut<n>[?]	"SOURce3:TRIGger:ZSUBstitut<n>[?] " on page 146
:TRIGger:PRBN<n>[?]	"SOURce3:TRIGger:PRBN<n>[?] " on page 146
:TRIGger:PRBS<n>[?]	"SOURce3:TRIGger:PRBS<n>[?] " on page 146
:TRIGger:UPATtern<n>	"SOURce3:TRIGger:UPATtern<n> " on page 147

NOTE

See "How the Serial BERT Sends Triggers" on page 60 for details about trigger signals are generated.

SOURce3:TRIGger[:MODE][?]

IVI-COM Equivalent IAgilentN490xPGTrigger.Mode (IVI-compliant)

Syntax SOURce3:TRIGger[:MODE] DCLock | PATTern | SEQuence

SOURce3:TRIGger[:MODE]?

Return Range DCL | PATT | SEQ

Description This command sets the pattern generator's Trigger Out to one of three modes:

Parameters **DCLock**: Sets the Trigger Out to divided clock mode. The divider is set with the command SOURce3:TRIGger:DCDRatio.

PATTern: Sets the Trigger Out to pattern trigger mode. The subsequent commands apply.

SEQuence: Sets the Trigger Out to sequence trigger mode. If a user-defined sequence is used, only DCL or SEQ can be chosen.

In sequence trigger mode, a trigger pulse can be generated whenever a block is started or restarted.

Whether that happens or not, depends on the block characteristics. Trigger On/Off can be specified individually for each block. For details see "[SOURce[1]]:PATTern:SEQuence:DATA[?] " on page 106 and "SequenceExpression for User-Defined Sequences" on page 107.

SOURce3:TRIGger:DCDRatio

IVI-COM Equivalent IAgilentN490xPGTrigger.DivisionRate (IVI-compliant)

Syntax SOURce3:TRIGger:DCDRatio <NR1>

Description The command sets the trigger subratio. CTDRatio? is the equivalent query. Range for the divider: 2 ... 128.

SOURce3:TRIGger:CTDRatio?

IVI-COM Equivalent IAgilentN490xPGTrigger.DivisionRate (IVI-compliant)

Syntax SOURce3:TRIGger:CTDRatio

Description This query returns the trigger subratio. DCDRatio is the equivalent command.

SOURce3:TRIGger:APATtern<n>[?]

IVI-COM Equivalent IAgilentN490xPGTrigger.Patterntype (not IVI-compliant)

Syntax SOURce3:TRIGger:APATtern<n> ABCHange | SOPattern

SOURce3:TRIGger:APATtern<n>?

NOTE This command is for alternate patterns only.

Return Range ABCH | SOP

Description This command defines when a trigger should be sent from the pattern generator's Trigger Out port:

ABChange: The trigger is sent when the pattern being sent changes (from pattern A to pattern B or vice versa).

SOPattern: The pattern generator Trigger Out is synchronized to the start of a pattern.

The query returns the current state of the alternate pattern trigger mode.

NOTE See "How the Serial BERT Uses Alternate Patterns" on page 57 for additional information on how to work with alternate patterns.

SOURce3:TRIGger:MDENsity<n>[?]

IVI-COM Equivalent IAgilentN490xPGPosition.Bit (not IVI-compliant)

Syntax SOURce3:TRIGger:MDENsity<n> <Num.>

SOURce3:TRIGger:MDENsity<n>?

Description This command selects the bit position within the PRBS at which the trigger pulse is to be output for MDEN patterns. The number <n> must be in the range: 7, 10, 11, 13, 15, 23. The parameter <Num> must be in the range 0 through pattern length - 1.

The query returns the bit position within the pattern at which the trigger pulse is to be output.

SOURce3:TRIGger:ZSUBstitut<n>[?]

IVI-COM Equivalent IAgilentN490xPGPosition.Bit (not IVI-compliant)

Syntax SOURce3:TRIGger:ZSUBstitut<n> <Num.>

SOURce3:TRIGger:ZSUBstitut<n>?

Description This command selects the bit position within the zero substituted 2^n PRBS at which the trigger pulse is to be output for ZSUB patterns. The number <n> must be in the range: 7, 10, 11, 13, 15, 23. The parameter <Num> must be in the range 0 through pattern length - 1.

The query returns the bit position within the pattern at which the trigger pulse is to be output.

SOURce3:TRIGger:PRBN<n>[?]

IVI-COM Equivalent IAgilentN490xPGPosition.Bit (not IVI-compliant)

Syntax SOURce3:TRIGger:PRBN<n> <Num.>

SOURce3:TRIGger:PRBN<n>?

Description This command selects the bit position within the PRBS at which the trigger pulse is to be output for PRBN patterns. The number <n> must be in the range: 7, 10, 11, 13, 15, 23. The parameter <Num> must be in the range 0 through pattern length - 1.

The query returns the bit position within the pattern at which the trigger pulse is to be output.

SOURce3:TRIGger:PRBS<n>[?]

IVI-COM Equivalent IAgilentN490xPGPosition.SetPattern (not IVI-compliant)

Syntax SOURce3:TRIGger:PRBS<n> <0 | 1 | OFF | ON>{,<0 | 1 | OFF | ON>}

SOURce3:TRIGger:PRBS<n>?

Description

This command sets the pattern, the occurrence of which causes a trigger pulse to be output for PRBS patterns. In other words, when the defined pattern occurs, a trigger pulse is generated.

The number <n> must be in the range: 7, 10, 11, 15, 23, 31. The number of parameters depends on the pattern length, and is the minimum that can define a unique place in the overall pattern, for example a pattern of length 2^{n-1}, the number of parameters is n. The parameter values are either 1 or 0. An *all-ones* pattern is not allowed.

To generate a trigger pulse for a PRBS7 pattern on occurrence of 1010101, the following command would be sent:

```
SOUR3:TRIG:PRBS7 1,0,1,0,1,0,1
```

The query returns the state of the N-bit trigger pattern function for the pattern generator's Trigger Out.

SOURce3:TRIGger:UPATtern<n>

IVI-COM Equivalent

IAgilentN490xPGPosition.Bit (not IVI-compliant)

Syntax

SOURce3:TRIGger:UPATtern<n> <Num.>

SOURce3:TRIGger:UPATtern<n>?

Description

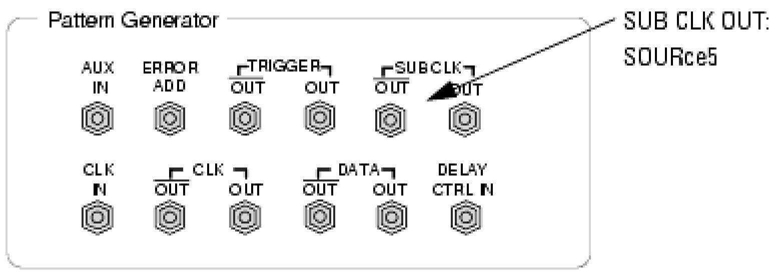
The command selects a bit position within the user pattern at which the trigger pulse is to be output for user patterns. The parameter must be in the range of 0 through pattern length - 1.

The response returns the current bit position within the user pattern at which the trigger pulse is generated.

SOURce5 Subsystem

SOURce5 Subsystem - Reference

The SOURce5 Subsystem represents the pattern generator's Subrate Clock Out port SUB CLK.



This subsystem has just one command:

SOURce5:DIVider[?]

IVI-COM Equivalent IAgilentN490xPGClock2.SubRateDivider

Syntax SOURce5:DIVider <NR1>

SOURce5:DIVider?

Description This command is used to set the clockrate divider. The range is 2 to 128.

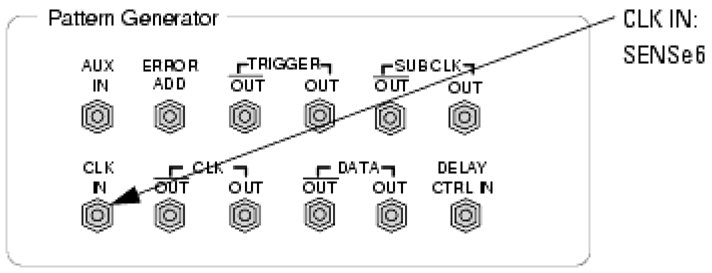
The query returns the current divider setting.

NOTE The subrate clock has a fixed amplitude and no fixed phase correlation with the actual clock.

SENSe6 Subsystem

SENSe6 Subsystem - Reference

The SENSe6 Subsystem represents the pattern generator's Clock In ports.



The Serial BERT has a 10 MHz Reference Input at the rear.

This subsystem has the following SCPI structure:

```
SENSe6
├ :FREQuency
│  └ [:CW | :FIXed]
└ :MODe
```

This subsystem has the following commands:

Table 34

Name	Description under
:FREQuency[:CW|:FIXed][?]	"SENSe6:FREQuency[:CW|:FIXed]?" on page 149
:MODE	"SENSe6:MODE " on page 150

SENSe6:FREQuency[:CW|:FIXed]?

IVI-COM Equivalent IAgilentN490xPGClockIn.GetFrequency (IVI-compliant)

Syntax SENSe6:FREQuency [:CW | :FIXed]?

Description The *query* returns the frequency of the signal at the pattern generator's Clock In port. You may use the following forms of this query:

- SENSe6:FREQ?
- SENSe6:FREQ:CW?
- SENSe6:FREQ:FIXed?

There is no difference between any of these forms.

It is recommended that you switch off the pattern generator's clock output before you change the mode. See "OUTPut[1]:CENTer " on page 128.

NOTE

This query supersedes the following 716xxB command:

- SOURce2:FREQuency[:CW | :FIXed]? <Num.>

SENSe6:MODE

IVI-COM Equivalent IAgilentN490xPGClockIn.Mode (IVI-compliant)

Syntax SENSe6:MODE <INT | EXT | EXTMAN | REF>

Description This command sets the pattern generator's clock source. The mode can be one of the following:

- INTernal

  Selects the Serial BERT's internal clock oscillator.

- EXTernal

  Selects the clock signal at the CLK IN port. The frequency is automatically measured and used.

- EXTMANual

  In this mode, you can measure the frequency of the external source clock (see "SENSe6:FREQuency[:CW|:FIXed]?" on page 149) and set the clock frequency explicitly (see "SOURce9:FREQuency[:CW|FIXed][?] " on page 133).

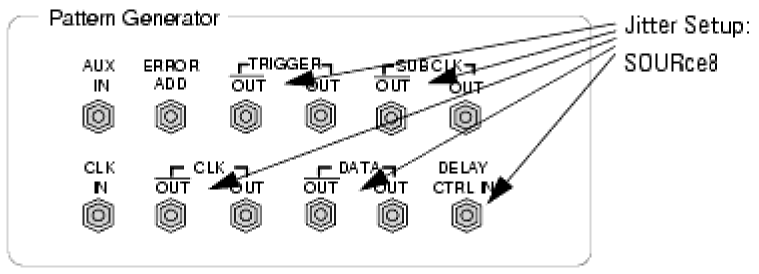
- REFerence (10 MHz Reference clock)

  This is the clock signal at the 10 MHz Ref In port. A frequency of 10 MHz is required.

SOURce8 Subsystem

SOURce8 Subsystem - Reference

The SOURce8 subsystem controls the pattern generator's Jitter Setup. This chapter refers only to Agilent N4903 Serial BERT instruments on which the calibrated and integrated jitter injection option J10 is installed.



This subsystem has the following SCPI structure:

```
:SOURce8[:JITTer]
 ├ :CONF[:TYPE]
 ├ :EXT[:STATe]
 ├ :Global[:STATe]
 ├ :RANDom
 │  └ ...
 ├ :BUNCorrelated
 │  └ ...
 ├ :PERiodic
 │  └ ...
 └ :SINusoidal
    └ ...
```

This subsystem has the following commands and subnodes:

Table 35

Name	Description under
Commands	
:CONFigure	"SOURce8[:JITTer]:CONF[:TYPE][?] " on page 152
:EXTernal	"SOURce8[:JITTer]:EXTernal[:STATe][?] " on page 153
GLOBal	"SOURce8[:JITTer]:GLOBal[:STATE][?] " on page 153
Subnodes	
:RANDom	"SOURce8[:JITTer]:RANDom Subnode" on page 154
:BUNCorrelated	"SOURce8[:JITTer]:BUNCorrelated Subnode" on page 156
:PERiodic	"SOURce8[:JITTer]:PERiodic Subnode" on page 160
:SINusoidal	"SOURce8[:JITTer]:SINusoidal Subnode" on page 162

SOURce8[:JITTer]:CONF[:TYPE][?]

IVI-COM Equivalent IAgilentN490xJitterDistributionMode

Syntax SOURce8:CONFigure[:TYPE] DL1 | DL2

SOURce8:CONF?

Description Selects one of the two lines.

- DL1 provides a delay of up to 200 ps.
- DL2 provides a delay of up to 500 ps. This delay line can be used for data rates up to 3.5 GBit/s.

That means, the command SOURce8:CONF DL2 returns an error if the data rate is above 3.5 GBit/s.

The query returns the present setting (DL1 | DL2).

SOURce8[:JITTer]:EXTernal[:STATe][?]

IVI-COM Equivalent

IAgilentN490xJExternal.Enable

Syntax

SOURce8:EXTernal[:STATe] ON | OFF | 1 | 0

SOURce8:EXT?

Description

Enables/disables the Delay Ctrl input when option J10 is installed. For bandwidth limitations refer to the technical specifications.

Disabling the input terminates the connector to 50 Ω.

The query returns the present setting (0 | 1).

SOURce8[:JITTer]:GLOBal[:STATE][?]

IVI-COM Equivalent

IAgilentN490xJitterEnable

Syntax

SOURce8:GLOBal[:STATE] ON | OFF | 1 | 0

SOURce8:GLOB?

Description

Enables or disables the jitter generation. Refers to random, bounded uncorrelated, periodic, sinusoidal, and external jitter. Has no effect on spread spectrum clocking.

The query returns the present setting (0 | 1).

SOURce8[:JITTer]:RANDom Subnode

Random jitter is generated by an internal noise generator. Pure random jitter has a Gaussian distribution.

This subnode has the following SCPI structure:

```
:SOURce8[:JITTer]
└ :RANDom
  ├ [:STATe] [?]
  ├ :FILTer
  │ ├ [:LPASs][:STATe][?]
  │ └ :HPASs[:STATe][?]
  ├ :LEVel[?]
  └ :CFACtor?
```

This subnode has the following commands:

Table 36

Name	Description under
[:STATe][?]	“SOURce8[:JITTer]:RANDom[:STATe][?] ” on page 154
:FILTer[:LPASs][?]	“SOURce8[:JITTer]:RANDom:FILTer[:LPASs][:STATe][?] ” on page 155
:FILTter:HPASs[?]	“SOURce8[:JITTer]:RANDom:FILTer:HPASs[:STATe][?] ” on page 155
:LEVel[?]	“SOURce8[:JITTer]:RANDom:LEVel[?] ” on page 155
:CFACtor?]	“SOURce8[:JITTer]:RANDom:CFACtor? ” on page 156

SOURce8[:JITTer]:RANDom[:STATe][?]

IVI-COM Equivalent IAgilentN490xJRandom.Enable

Syntax SOURce8:RANDom[:STATE] ON | OFF | 1 | 0

SOURce8:RAND?

Description The command turns the generation of random jitter on or off.

The query returns the present setting (0 | 1).

SOURce8[:JITTer]:RANDom:FILTer[:LPASs][:STATe][?]

IVI-COM Equivalent IAgilentN490xJRandom.LowPassFilterEnable

Syntax SOURce8:RANDom:FILTer ON | OFF | 1 | 0

SOURce8:RANDom:FILT?

Description Turns the 500 MHz low-pass filter for random jitter on or off.

The query returns the present setting (0 | 1).

SOURce8[:JITTer]:RANDom:FILTer:HPASs[:STATe][?]

IVI-COM Equivalent IAgilentN490xJRandom.HighPassFilterEnable

Syntax SOURce8:RANDom:FILTer:HPAS ON | OFF | 1 | 0

SOURce8:RANDom:FILT:HPAS?

Description Turns the 10 MHz high-pass filter for random jitter on or off.

The query returns the present setting (0 | 1).

SOURce8[:JITTer]:RANDom:LEVel[?]

IVI-COM Equivalent IAgilentN490xJRandom.Amplitude, GetMaxAmplitude(), GetMinAmplitude()

Syntax SOURce8:RANDom:LEVel <NR3>

SOURce8:RANDom:LEVel? MIN | MAX

Description The command sets the root mean square (rms) RJ level in unit intervals (UI).

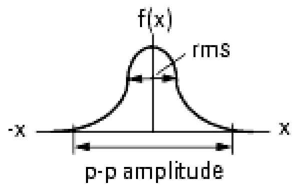
The query returns the present setting or the applicable min/max values in UI.

SOURce8[:JITTer]:RANDom:CFACtor?

Syntax SOURce8:RANDom:CFACtor?

Description This query returns the crest factor of the generated random jitter distribution. The present implementation of random jitter generates a fixed crest factor of 14.

The crest factor is defined as the peak-to-peak value divided by the rms value.



Crest factor = peak-to-peak value / rms value.

SOURce8[:JITTer]:BUNCorrelated Subnode

Bounded uncorrelated jitter is generated from a pseudo random binary sequence (PRBS).

This subnode has the following SCPI structure:

```
:SOURce8[:JITTer]
└ :BUNCorrelated
   ├ [:STATe][?]
   ├ :PRESet[?]
   ├ :FILTer
   │  └ [:LPASs]:SELect[?]
   ├ :PRBSequence
   │  ├ [:SELect][?]
   │  └ :DRATe[?]
   └ :LEVel[?]
```

This subnode has the following commands:

Table 37

Name	Description under
[:STATe][?]	"SOURce8[:JITTer]:BUNCorrelated [:STATe][?] " on page 157
:Preset[?]	"SOURce8[:JITTer]:BUNCorrelated [:STATe][?] " on page 157
:FILTer[:LPASs]:SELect[?]	"SOURce8 [:JITTer]:BUNCorrelated:FILTer:SELect [?] " on page 158
:PRBSequence[:SELect][?]	"SOURce8 [:JITTer]:BUNCorrelated:PRBSequence [?] " on page 158
:PRBSequence:DRATe[?]	"SOURce8 [:JITTer]:BUNCorrelated:PRBSequ:DRATe[?] " on page 159
:LEVel[?]	"SOURce8 [:JITTer]:BUNCorrelated:LEVel[?] " on page 159

SOURce8[:JITTer]:BUNCorrelated[:STATe][?]

IVI-COM Equivalent IAgilentN490xJBoundedUncorrelated.Enable

Syntax SOURce8:BUNCorrelated[:STATE] ON | OFF | 1 | 0

SOURce8:BUNC?

Description Turns the generation of bounded uncorrelated jitter on or off.

The query returns the present setting (0 | 1).

SOURce8[:JITTer]:BUNCorrelated:PRESet[?]

IVI-COM Equivalent	IAgilentN490xJBundedUncorrelated.Standard
Syntax	SOURce8:BUNCorrelated PRESet CUSTom | CEI6G | CEI11G| BGA SOURce8:BUNC:PRES?
Description	Can be used to set the PRBS data rate, polynomial and low-pass filter to predefined values: CUSTom: The individual parameter values apply. CEI6G: Meant for CEI 6 Gbit/s tests: PRBS data rate is 1.1 Gbit/s, the PRBS polynomial is 2 ^{9}-1, the low-pass filter is 100 MHz. CEI11G: Meant for CEI 11 Gbit/s tests: PRBS data rate is 2 Gbit/s, the PRBS polynomial is 2 ^{11}-1, the low-pass filter is 200 MHz. BGA: Preset values for Gaussian distribution: PRBS data rate is 2 Gbit/s, the PRBS polynomial is 2 ^{31}-1, the low-pass filter is 100 MHz. The query returns the present setting.

SOURce8[:JITTer]:BUNCorrelated:FILTer:SELect[?]

IVI-COM Equivalent	IAgilentN490xJBoundedUncorrelatedLowPassFilterCutOffFrequeny
Syntax	SOURce8:BUNCorrelated:FILTer[:LPASs]:SELect LP20 | LP50 | LP100 | LP200 SOURce8:BUNC:FILT:SEL?
Description	Selects the low-pass filter for bounded uncorrelated jitter (20, 50, 100, or 200 MHz). The query returns the present setting.

SOURce8[:JITTer]:BUNCorrelated:PRBSequence[?]

IVI-COM Equivalent	IAgilentN490xJBoundedUncorrelated.PRBS
Syntax	SOURce8:BUNCorrelated:PRBSequence[:SELect] PRBS7 | PRBS8 | PRBS9 | PRBS10 | PRBS11 | PRBS15 | PRBS23 | PRBS31

SOURce8:BUNC:PRBS?

Description: Selects the polynomial 2^{7}-1 to 2^{31}-1 of the PRBS to be generated.

The query returns the present setting.

SOURce8[:JITTer]:BUNCorrelated:PRBSequ:DRATe[?]

IVI-COM Equivalent: IAgilentN490xJBoundedUncorrelated.DataRate

Syntax: SOURce8:BUNCorrelated:PRBSequence:DRATe <NR3>

SOURce8:BUNC:PRBS:DRAT? MIN | MAX

Description: Sets the data rate of the internal PRBS source for bounded uncorrelated jitter in Hz.

The query returns the present setting or the applicable min/max values.

SOURce8[:JITTer]:BUNCorrelated:LEVel[?]

IVI-COM Equivalent: IAgilentN490xJBoundedUncorrelated.Amplitude

Syntax: SOURce8:BUNCorrelated:LEVel <NR3>

SOURce8:BUNCorrelated:LEVel? MIN | MAX

Description: The command sets the bounded uncorrelated jitter peak-to-peak level in unit intervals (UI).

The query returns the present setting or the applicable min/max values in UI.

SOURce8[:JITTer]:PERiodic Subnode

Periodic jitter is generated by a built-in function generator. Sine, square, and triangle waveforms are available.

This subnode has the following SCPI structure:

```
:SOURce8[:JITTer]
└ :PERiodic
   ├ [:STATe][?]
   ├ :FREQuency[?]
   ├ :FUNCtion[:SELect][?]
   └ :LEVel[?]
```

This subnode has the following commands:

Table 38

Name	Description under
[:STATe][?]	"SOURce8[:JITTer]:PERiodic[:STATe][?] " on page 160
:FREQuency[?]	"SOURce8 [:JITTer]:PERiodic:FREQuency[?] " on page 161
:FUNCtion[:SELect][?]	"SOURce8[:JITTer]:PERiodic:FUNCtion [:SELect][?] " on page 161
:LEVel[?]	"SOURce8[:JITTer]:PERiodic:LEVel[?] " on page 161

SOURce8[:JITTer]:PERiodic[:STATe][?]

IVI-COM Equivalent IAgilentN490xJPeriodic.Enable

Syntax SOURce8:PERiodic[:STATE] ON | OFF | 1 | 0

SOURce8:PER?

Description Turns the generation of periodic jitter on or off.

The query returns the present setting (0 | 1).

SOURce8[:JITTer]:PERiodic:FREQuency[?]

IVI-COM Equivalent	IAgilentN490xJPeriodic.ModulationFrequency
Syntax	SOURce8:PERiodic:FREQuency <NR3> SOURce8:PER:FREQ? MIN | MAX
Description	Sets the frequency of the periodic jitter in Hz. Note that the maximum frequency depends on the selected waveform (see "SOURce8[:JITTer]:PERiodic:FUNCtion[:SELect][?]" on page 161). The query returns the present setting or the applicable min/max values.

SOURce8[:JITTer]:PERiodic:FUNCtion[:SELect][?]

IVI-COM Equivalent	IAgilentN490xJPeriodic.WaveForm
Syntax	SOURce8:PERiodic:FUNCtion[:SELect] SINusoid | SQUare | TRIangle SOURce8:PER:FUNC?
Description	Selects the characteristics of the periodic jitter. The query returns the present setting (SIN | SQU | TRI).

SOURce8[:JITTer]:PERiodic:LEVel[?]

IVI-COM Equivalent	IAgilentN490xJPeriodic.Amplitude
Syntax	SOURce8:PERiodic:LEVel <NR3> SOURce8:PER:LEVel? MIN | MAX
Description	The command sets the periodic jitter peak-to-peak level in unit intervals (UI).

The query returns the present setting or the applicable min/max values in UI.

SOURce8[:JITTer]:SINusoidal Subnode

Sinusoidal jitter is generated by modulating the clock that is used for generating the data pattern.

Sinusoidal jitter is not available if the spread spectrum clock is enabled.

This subnode has the following SCPI structure:

```
:SOURce8[:JITTer]
 └ :SINusoidal
    ├ [:STATe][?]
    ├ :CLOCk[?]
    ├ :FREQuency[?]
    └ :LEVel[?]
```

This subnode has the following commands:

Table 39

Name	Description under
[:STATe][?]	"SOURce8[:JITTer]:SINusoidal[:STATe][?] " on page 162
:CLOCk[?]	"SOURce8[:JITTer]:SINusoidal:CLOCk[?]" on page 163
:FREQuency[?]	"SOURce8 [:JITTer]:SINusoidal:FREQuency[?] " on page 163
:LEVel[?]	"SOURce8[:JITTer]:SINusoidal:LEVel[?] " on page 163

SOURce8[:JITTer]:SINusoidal[:STATe][?]

IVI-COM Equivalent IAgilentN490xJSinoidal.Enable

Syntax SOURce8:SINusoidal[:STATE] ON | OFF | 1 | 0

SOUR8:SIN?

Description Turns the generation of sinusoidal jitter on or off.

The query returns the present setting (0 | 1).

SOURce8[:JITTer]:SINusoidal:CLOCk[?]

IVI-COM Equivalent IAgilentN490xJSinusoidalModulatedClock

Syntax SOURce8:SINusoidal:CLOCk PURE | MODulated

SOUR8:SIN:CLOC?

Description Selects the property of the clock that is provided by the pattern generator's CLK OUT port:

- PURE: The clock remains unmodulated.
- MODulated: The clock at CLK OUT is modulated.

The query returns the present setting (PURE | MOD).

SOURce8[:JITTer]:SINusoidal:FREQuency[?]

IVI-COM Equivalent IAgilentN490xJSinoidal.ModulationFrequency

Syntax SOURce8:SINusoidal:FREQuency <NR3>

SOURce8:SIN:FREQ? MIN | MAX

Description Sets the frequency of the SINusoidal jitter in Hz.

The query returns the present setting or the applicable min/max values.

SOURce8[:JITTer]:SINusoidal:LEVel[?]

IVI-COM Equivalent IAgilentN490xJSinoidal.Amplitude

Syntax SOURce8:SINusoidal:LEVel <NR3>

SOURce8:SIN:LEVel? MIN | MAX

Description

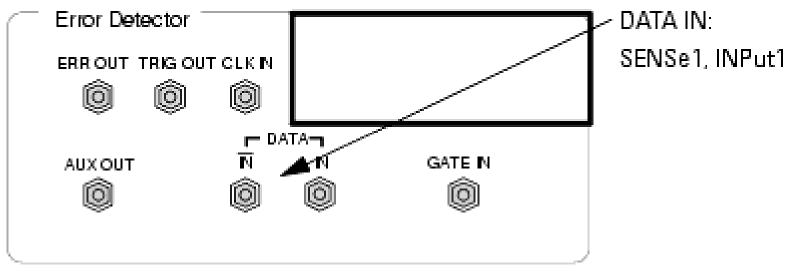
The command sets the SINusoidal jitter peak-to-peak level in unit intervals (UI). Note that the maximum level depends on the jitter frequency.

The query returns the present setting or the applicable min/max values in UI.

INPut[1] Subsystem

INPut[1] Subsystem - Reference

The INPut[1] subsystem represents the error detector's Data In port.



This subsystem has the following commands:

Table 40

Name	Description under
:COUPling[?]	"INPut[1]:COUPling[?] " on page 165
:DELay[?]	"INPut[1]:DELay[?] " on page 165
:POLarity[?]	"INPut[1]:POLarity[?] " on page 166
:TERMination[?]	"INPut[1]:TERMination[?] " on page 166

Table 40

Name	Description under
:STATe[?]	"INPut[1]:STATe[?] " on page 166
:CMODe[?]	"INPut[1]:CMODe[?] " on page 167

INPut[1]:COUPling[?]

IVI-COM Equivalent IAgilentN490xEDDataIn.TerminationEnabled (IVI-compliant)

Syntax INPut[1]:COUPling AC | DC

INPut[1]:COUPling?

Description The command enables or disables the source of the termination voltage:

- DC: Enables the termination voltage
- AC: Disables the termination voltage

The query returns the current state.

NOTE Non-differential operation of the error detector's Data In port requires a termination voltage.

INPut[1]:DELay[?]

IVI-COM Equivalent IAgilentN490xEDDataIn.Delay (IVI-compliant)

Syntax INPut[1]:DELay <Num.>

INPut[1]:DELay?

Description This command sets the delay of the active edge of the clock output relative to the error detector's Data In port. The units are seconds. The value is rounded to the nearest one picosecond. The response returns the current data to clock delay value.

This command has restrictions for frequencies under 620 Mbits/s. See the Serial BERT User Guide (or online Help) for details.

INPut[1]:POLarity[?]

IVI-COM Equivalent IAgilentN490xEDDataIn.Polarity (IVI-compliant)

Syntax INPut[1]:POLarity NORMal|INVerted

INPut[1]:POLarity?

Description The command sets the polarity of the error detector's Data In port.

The query returns the current polarity of the error detector's Data In port.

INPut[1]:TERMination[?]

IVI-COM Equivalent IAgilentN490xEDDataIn.VTermination (IVI-compliant)

Syntax INPut[1]:TERMination <NR3>

INPut[1]:TERMination?

Description This command sets the data termination level of the error detector's Data In port. The response form returns the data termination level.

This command is only valid if the coupling is set to DC (see "INPut [1]:COUPling[?] " on page 165).

If input termination and 0/1 threshold level are to be set up, the input termination should be set up first.

INPut[1]:STATe[?]

IVI-COM Equivalent IAgilentN490xEDDataIn.Enabled (IVI-compliant)

Syntax INPut[1]:STATe ON | OFF | 0 | 1

Description Enables or disables normal data input.

NOTE It is not possible to disable the Serial BERT's input. This command has no effect.

INPut[1]:CMODe[?]

IVI-COM Equivalent IAgilentN490xEDDataIn.Mode (IVI-compliant)

Syntax INPut[1]:CMODe DIFFerential | NORMal | COMPlement

INPut[1]:CMODe?

Description Defines the **C**ompare **MODE**; this defines which input ports (DATA or $\overline{\text{DATA}}$) are active.

The following options are available:

- NORMal

  Only the inputs at the DATA are measured.

- COMPlement

  Only the inputs at the $\overline{\text{DATA}}$ are measured.

- DIFFerential

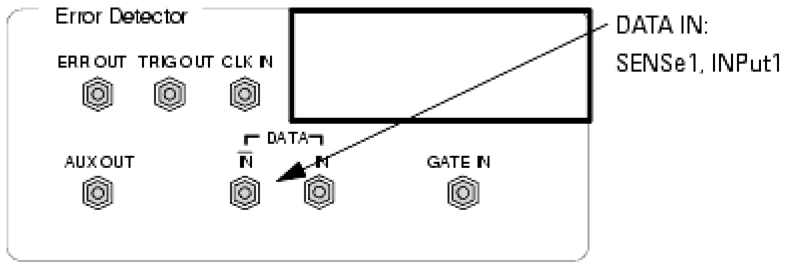
  The differential between DATA and $\overline{\text{DATA}}$ is measured.

NOTE Depending on the options of your Serial BERT, DIFFerential inputs may not be supported at your instrument. See the online Help or the User's Guide for a description of the available options.

SENSe[1] Subsystem

SENSe[1] Subsystem - Reference

The SENSe[1] subsystem represents the error detector's Data In port.



This subsystem has the following SCPI structure:

```
SENSe[1]
├─ :EYE
│   └ ...
├─ :ELOCation[?]
│   └ ...
├─ :BLOCk
│   └ ...
├─ :GATE
│   └ ...
├─ :LOGGing[?]
│   └ :FILename[?]
├─ :PATTern
│   └ ...
├─ :SYNChronizat[?]
│   └ :THReshold[?]
├─ :VOLTage
│   └ ...
├─ :FREQuency
│   └ [:CW | :FIXED][?]
└─ :AUXout
    └ MODe[?]
```

This subsystem has the following subnodes and commands:

Table 41

Name	Description under
Commands	

Table 41

Name	Description under
:LOGGing[?]	“SENSe[1]:LOGGing[?] ” on page 169
:LOGGing:FILename[?]	“SENSe[1]:LOGGing:FILename[?] ” on page 170
:SYNChronization[?]	“SENSe[1]:SYNChronizat[?] ” on page 170
:SYNChronization:THReshold[?]	“SENSe[1]:SYNChronization:THReshold [?] ” on page 171
:FREQuency[:CW|:FIXed][?]	“SENSe[1]:FREQuency[:CW|:FIXed][?] ” on page 171
:AUXout:MODE[1][?]	“SENSe[1]:AUXout:MODE[?] ” on page 172
Subnodes	
:BLOCk	“SENSe[1]:BLOCk Subnode” on page 173
:ELOCation	“SENSe[1]:ELOCation Subnode” on page 175
:EYE	“SENSe[1]:EYE Subnode” on page 179
:GATE	“SENSe[1]:GATE Subnode” on page 185
:PATTern	“SENSe[1]:PATTern Subnode” on page 189
:VOLTage	“SENSe[1]:VOLTage Subnode” on page 206

SENSe[1]:LOGGing[?]

IVI-COM Equivalent IAgilentN490xEDLogging.Enabled (IVI-compliant)

Syntax SENSe[1]:LOGGing 0 | 1 | OFF | ON

SENSe[1]:LOGGing?

Description This command allows you to save accumulated results in a file for later analysis. The query form returns whether or not the accumulated results will be saved in a log file.

NOTE This command is not precisely the same as 716xxB command. The ONCE parameter is no longer supported.

SENSe[1]:LOGGing:FILename[?]

IVI-COM Equivalent IAgilentN490xEDLogging.Filename (IVI-compliant)

Syntax SENSe[1]:LOGGing:FILename <Filename>

SENSe[1]:LOGGing:FILename?

Description This command specifies the file to which logged data is sent. The query returns the filename to which the logged data is sent.

You have to explicitly enable logging with SENSe[1]:LOGGing.

SENSe[1]:SYNChronizat[?]

IVI-COM Equivalent IAgilentN490xEDSynchronisation.AutoEnabled (IVI-compliant)

Syntax SENSe[1]:SYNChronizat ONCE | 0 | 1 | OFF | ON

SENSe[1]:SYNChronizat?

Description These commands configure the settings that control synchronization of the reference pattern to the incoming pattern.

- SENSe[1]:SYNChronizat ON enables automatic resynchronization.
- SENSe[1]:SYNChronizat OFF disables automatic resynchronization.
- SENSe[1]:SYNChronizat ONCE initiates a resynchronization attempt.

The query returns the current selection of the pattern synchronization.

SENSe[1]:SYNChronization:THReshold[?]

IVI-COM Equivalent IAgilentN490xEDSynchronisation.Threshold (IVI-compliant)

Syntax SENSe[1]:SYNChronization:THReshold

SENSe[1]:SYNChronization:THReshold?

Description SENSe[1]:SYNChronizat:THReshold <Num.>

SENSe[1]:SYNChronizat:THReshold?

This command sets the threshold level of error ratio at which synchronization is successful.

NOTE The valid values are 1E-01 thru 1E-08 in decade steps.

The query returns the threshold level of error ratio at which synchronization is set.

SENSe[1]:FREQuency[:CW|:FIXed][?]

IVI-COM Equivalent IAgilentN490xEDCDR.Frequency (not IVI-compliant)

Syntax SENSe[1]:FREQuency[:CW|:FIXed] <Num>

SENSe[1]:FREQuency[:CW|:FIXed]? <Num> | <MIN | MAX>

Description This command sets the clock frequency for clock data recovery. You can use any of the forms listed below:

- SENSe[1]:FREQuency?
- SENSe[1]:FREQuency:CW?
- SENSe[1]:FREQuency:FIXed?

These forms have the same effect.

The query returns the clock frequency for clock data recovery.

NOTE

Depending on the options of your Serial BERT, clock data recovery may not be supported at your instrument. See the online Help or the User's Guide for a description of the available options.

SENSe[1]:AUXout:MODE[?]

IVI-COM Equivalent IAgilentN490xEDAuxOut.Mode (not IVI-compliant)

Syntax SENSe[1]:AUXout:MODE CLOCk | DATA

SENSe[1]:AUXout:MODE?

Description This command sets the mode for the error detector's Aux Out port. The following settings are available:

- CLOCk

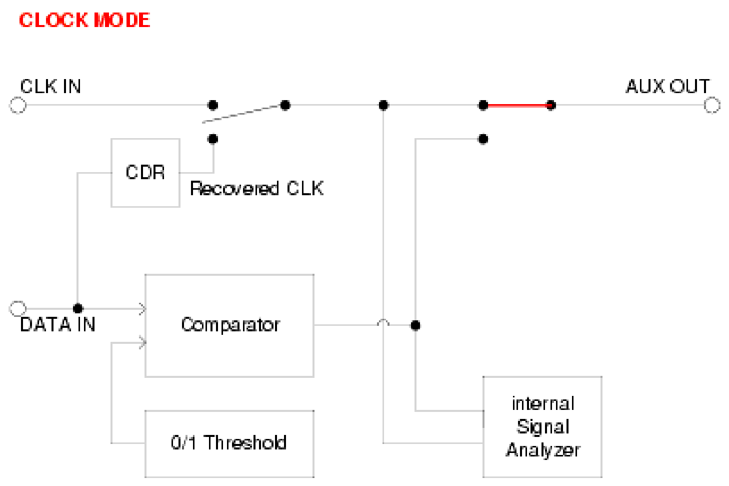
  The clock input is switched directly to the Aux Out port.

- DATA

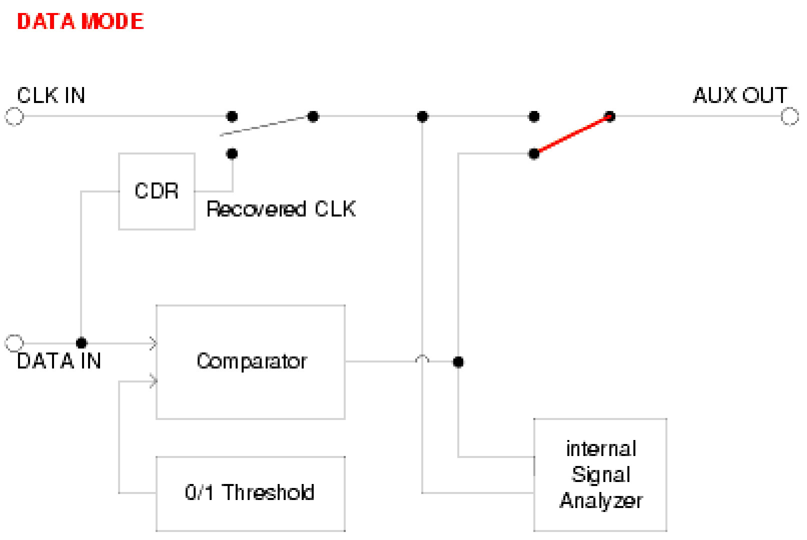
  The data input is switched via a comparator to the Aux Out port.

The comparator is controlled by the 0/1 threshold. You can use an oscilloscope to determine if the 0/1 threshold is correctly set. If the 0/1 threshold is set below or above the data eye, the output at Aux Out will be constant high or low, respectively.

The following figure shows how the clock signal is directed to Aux Out in CLOCk mode:



The following figure shows the circuit in DATA mode:



SENSe[1]:BLOCk Subnode

This subnode has the following SCPI structure:

```
SENSe[1]
└ :BLOCk[?]
   ├ BSTart[?]
   └ BLENgth[?]
```

This subnode has the following commands:

Table 42

Name	Description under
:BLOCk[?]	"SENSe[1]:BLOCk[?]" on page 174
:BLOCk:BSTart[?]	"SENSe[1]:BLOCk:BSTart[?]" on page 174
:BLOCk:BLENgth[?]	"SENSe[1]:BLOCk:BLENgth[?]" on page 175

SENSe[1]:BLOCk[?]

IVI-COM Equivalent: IAgilentN490xEDErrorLocation.Mode (not IVI-compliant)

Syntax: SENSe[1]:BLOCk ON | OFF | 0 | 1 | BEADdress | WPATtern | BLOCk

SENSe[1]:BLOCk?

Description: This command configures the error location feature of the instrument. It is only available for user-defined patterns (selected with PATTern:SELect UPATtern or PATTern:SELect FILename). Only the errors within the defined location are counted by the instrument.

Parameters: The command has the following options:

- WPATtern

  When this is selected, all errors that occur through the entire pattern are counted. OFF and 0 (NULL) have the same effect.

- BEADdress

  Only errors that occur at the specified bit address are counted. Use :ELOC:BEAD to specify the bit address.

- BLOCk

  Errors that occur within the specified bit address range are counted. The bit address range is defined with the :BLOC:BST and BLOC:BLEN commands. ON and 1 have the same effect.

Return Values: The query returns the following:

- 0 (equivalent to WPATtern)
- 1 (equivalent to BLOCk)
- BEAD

SENSe[1]:BLOCk:BSTart[?]

IVI-COM Equivalent: IAgilentN490xEDErrorLocation.BlockStart (not IVI-compliant)

Syntax: SENSe[1]:BLOCk:BSTart <numeric value>

SENSe[1]:BLOCk:BSTart?

Description

Sets the starting bit address of a bit address range for error location. The query returns the current value. This command only has an effect if the BLOCk option is set with the SENSe[1]:BLOCk command.

This value must be in the range: 0 ... pattern length - 1. Values out of range are set to the maximum value silently.

SENSe[1]:BLOCk:BLENgth[?]

IVI-COM Equivalent

IAgilentN490xEDErrorLocation.BlockLength (not IVI-compliant)

Syntax

SENSe[1]:BLOCk:BLENgth <numeric value>

SENSe[1]:BLOCk:BLENgth?

Description

Sets the length of a bit address range for error location. The query returns the current value. This command only has an effect if the BLOCk option is set with the SENSe[1]:BLOCk command.

This value must be in the range: 1 ... (pattern length - one). Zero is set to 1 silently. Values beyond the maximum value are set to the maximum value silently.

NOTE

The length is interpreted *round cycled*.

SENSe[1]:ELOCation Subnode

This subnode has the following SCPI structure:

```
SENSe[1]
└ :ELOCation[?]
   ├ :BEADdress[?]
   ├ :VERBose?
   └ :ECOunt?
```

This subnode has the following commands:

Table 43

Name	Description under
:ELOCation[?]	“SENSe[1]:ELOCation[?]” on page 176
:ELOCation:BEADdress	“SENSe[1]:ELOCation:BEADdress[?]” on page 177
:ELOCation:VERBose?	“SENSe[1]:ELOCation:VERBose?” on page 177
:ELOCation:ECOunt?	“SENSe[1]:ELOCation:ECOunt?” on page 178

See the Serial BERT User's Guide and “Using Error Location Capture - Procedures” on page 48 for additional information.

SENSe[1]:ELOCation[?]

IVI-COM Equivalent IAgilentN490xEDErrorLocation.CaptureErrors (not IVI-compliant)

Syntax SENSe[1]:ELOCation ONCE|OFF

SENSe[1]:ELOCation?

Description ONCE initiates the error location capture measurement. OFF stops a running error location measurement.

When an errored bit has been found, this command sets the location of the errored bit as the value that can be queried with SENSe [1]:ELOCation:BEADress?

Note that for an aborted measurement, no result files are generated.

The query returns the Error Location Capture *command* status. If 1 is returned, Error Location Capture has been triggered (but is not necessarily running). If 0 is returned, Error Location Capture has been either aborted (and may be still running) or successfully finished.

See the Serial BERT User's Guide and “Using Error Location Capture - Procedures” on page 48 for additional information.

This is an overlapped command.

SENSe[1]:ELOCation:BEADdress[?]

IVI-COM Equivalent IAgilentN490xEDErrorLocation.BitAddress (not IVI-compliant)

Syntax SENSe[1]:ELOCation:BEADdress <numeric value>

SENSe[1]:ELOCation:BEADdress?

Description This command only has effect if the BER location mode is set up to locate errors on a single bit in the pattern. The command sets the position of the bit that is to be monitored for errors.

The query returns the current value.

See the Serial BERT User's Guide and "SENSe[1]:BLOCk[?]" on page 174.

If the query is used with Error Location Capture, the return value depends on the Error Location Capture status:

- If an error has been captured, the bit position of the errored bit is returned.
- If no error has been captured since the last start of error location capture, the last set BEADdress is returned.

See the Serial BERT User's Guide and "Using Error Location Capture - Procedures" on page 48 for additional information.

SENSe[1]:ELOCation:VERBose?

IVI-COM Equivalent IAgilentN490xEDErrorLocation.ReadState (not IVI-compliant)

Syntax SENSe[1]:ELOCation:VERBose?

Description This query returns the current state of the error location capture measurement. The following responses are possible:

- `ELOC__NOT_STARTED_YET`

  Status: Stopped; ELOC not started since last instrument power-up.

- `ELOC__RUNNING`

  Status: Running; ELOC is running, searching for an errored bit.

- `ELOC__ERROR_DETECTED`

Status: Running; ELOC has detected an errored bit. The data is being evaluated.

- `ELOC__SUCCESS`

  Status: Stopped; Last ELOC run was ended successfully: an errored bit was found in the bit stream and its bit position was calculated. The bit position is returned with ELOC:BEAD?.

- `ELOC__FAILED`

  Status: Stopped; last ELOC run was cancelled either due to a user error or internal error.

- `ELOC__ABORTED`

  Status: Stopped; last ELOC run was cancelled by the user (either from the user interface or remotely).

NOTE

Please note that the responses each have two underscores after ELOC.

See the Serial BERT User's Guide and "Using Error Location Capture - Procedures" on page 48 for additional information.

SENSe[1]:ELOCation:ECOunt?

IVI-COM Equivalent IAgilentN490xEDErrorLocation.ReadCount (not IVI-compliant)

Syntax SENSe[1]:ELOCation:ECOunt?

Description Returns the number of bit-errors detected within the captured pattern. Is only set when an ELOC run was successful.

See the Serial BERT User's Guide and "Using Error Location Capture - Procedures" on page 48 for additional information.

SENSe[1]:EYE Subnode

This subnode has the following SCPI structure:

```
SENSe[1]
└ :EYE
   ├ :ACENter[?]
   ├ :ALIGN
   │  └ :AUTO[?]
   │     └ :MESSage?
   ├ :HEIGht?
   ├ :QUICk
   │  ├ :ALIGN
   │  │  └ :AUTO?
   │  ├ :ACENter[?]
   │  └ :TCENter[?]
   ├ :TCENter[?]
   ├ :THReshold[?]
   └ :WIDTh?
```

This subnode has the following commands:

Table 44

Name	Description under
:ACENter[?]	“SENSe[1]:EYE:ACENter[?] ” on page 180
:ALIGN:AUTO[?]	“SENSe[1]:EYE:ALIGN:AUTO[?] ” on page 181
:ALIGN:AUTO:MESSage?	“SENSe [1]:EYE:ALIGN:AUTO:MESSage?” on page 182
:HEIGht?	“SENSe[1]:EYE:HEIGht? ” on page 182
:QUICk:ALIGn:AUTO?	“SENSe[1]:EYE:QUICk:ALIGn:AUTO? ” on page 182
:QUICk:ACENter[?]	“SENSe[1]:EYE:QUICk:ACENter[?] ” on page 182

Table 44

Name	Description under
:QUICk:TCENter[?]	"SENSe[1]:EYE:QUICk:TCENter[?] " on page 183
:TCENter[?]	"SENSe[1]:EYE:TCENter[?] " on page 183
:THReshold[?]	"SENSe[1]:EYE:THReshold[?] " on page 184
:WIDTh?	"SENSe[1]:EYE:WIDTh? " on page 184

SENSe[1]:EYE:ACENter[?]

IVI-COM Equivalent IAgilentN490xEDSampling.ZeroOneThresholdCenter (IVI-compliant)

Syntax SENSe[1]:EYE:ACENter

SENSe[1]:EYE:ACENter?

Description This command initiates a search for the 0/1 threshold voltage midway between the two 0/1 threshold voltages with a measured BER just in excess of the BER configured by the EYE:THReshold command. If successful, the command leaves the 0/1 threshold at this value, and the center of the eye can be found by querying the 0/1 threshold value.

If unsuccessful, EYE:HEIGht? returns 9.91E+37 (Not-A-Number, NAN). The command :ACENter OFF aborts a previously started search. When this command is in execution, SENSe [1]:VOLTage:ZOTHreshold:AUTO is set to OFF. The query returns the current value of the 0/1 threshold voltage.

This command is blocked for frequencies under 620 Mbits/s. See for details. See the Serial BERT User Guide (or online Help) for details.

NOTE This command temporarily disables auto align.

NOTE The command :ACENter is an overlapped command. For more information, see "Overlapped and Sequential Commands " on page 74the .

SENSe[1]:EYE:ALIGN:AUTO[?]

IVI-COM Equivalent IAgilentN490xEDSampling.AutoAlign (IVI-compliant)

Syntax SENSe[1]:EYE:ALIGN:AUTO ONCE | 1 | ON

SENSe[1]:EYE:ALIGN:AUTO?

Description This command is only available for instrument setups that include the following:

- BER Threshold ≤ 1.0E-2
- PRBS patterns (2^n -1)
- Memory based patterns

This command starts autoalign.

This command is blocked for frequencies under 620 Mbits/s. See for details. See the Serial BERT User Guide (or online Help) for details.

Return Parameters The query returns one of the following:

- CS_AUTO_ALIGN_INPROGRESS

  Auto Alignment in progress
- CS_CLOCKTODATA_ALIGN_INPROGRESS

  Clock to Data Alignment in progress
- CS_01THRESHOLD_INPROGRESS

  Threshold Alignment in progress
- CS_ABORTED

  Alignment interrupted by user command (*RST or :SENS1:EYE:ALIG:AUTO 0)
- CS_FAILED

  Alignment failed, the reason can be requested by "SENSe [1]:EYE:ALIGN:AUTO:MESSage?" on page 182.
- CS_SUCCESSFUL

  Alignment completed successfully, the eye parameters can be requested.

SENSe[1]:EYE:ALIGN:AUTO:MESSage?

IVI-COM Equivalent IIviDriverUtility.ErrorQuery (IVI-compliant)

Syntax SENSe[1]:EYE:ALIGN:AUTO:MESSage?

Description This query returns any message generated by the last autoalign. The message may be generated by autoalign, threshold center, or datadelay center functions. The message is returned as an unquoted string.

SENSe[1]:EYE:HEIGht?

IVI-COM Equivalent IAgilentN490xEDSampling.ReadEyeHeight (IVI-compliant)

Syntax SENSe[1]:EYE:HEIGht?

Description This is a query command that searches for the value of data amplitude that puts the 0/1 threshold level midway between the upper and lower bounds at which the error ratio exceeds the threshold value set by the :EYE:THReshold command.

If the result is not available or the search was unsuccessful, then the number 9.91E+37 (Not-A-Number, NAN) will be returned.

SENSe[1]:EYE:QUICk:ALIGn:AUTO?

Syntax SENSe[1]:EYE:QUICk:ALIGN:AUTO ONCE | 0 | 1 | OFF | ON

SENSe[1]:EYE:QUICk:ALIGN:AUTO? <Results>

Description This command calls "SENSe[1]:EYE:ALIGN:AUTO[?] " on page 181. It has no functionality of its own.

SENSe[1]:EYE:QUICk:ACENter[?]

Syntax SENSe[1]:EYE:QUICk:ACENter ONCE | 0 | 1 | OFF | ON

SENSe[1]:EYE:QUICk:ACENter?

Description

This command calls "SENSe[1]:EYE:ACENter[?]" on page 180. It has no functionality of its own.

SENSe[1]:EYE:QUICk:TCENter[?]

Syntax

SENSe[1]:EYE:QUICk:TCENter ONCE | 0 | 1 | OFF | ON

SENSe[1]:EYE:QUICk:TCENter?

Description

This command calls "SENSe[1]:EYE:TCENter[?]" on page 183. It has no functionality of its own.

SENSe[1]:EYE:TCENter[?]

IVI-COM Equivalent

IAgilentN490xEDSamplingClockDataAlignCenter

IVI-COM Equivalent

IAgilentN490xEDSampling.ClockDataAlignCenter (IVI-compliant)

Syntax

SENSe[1]:EYE:TCENter ONCE | 0 | 1 | OFF | ON

SENSe[1]:EYE:TCENter?

Description

This command initiates a search for the value of data/clock delay that puts the active clock edge in the center of the data eye, midway between the two relative delay points with a measured BER just in excess of the BER configured by the EYE:THReshold command. If successful, the command leaves the data/clock delay at this value and the center of the eye can be found by querying the data delay value. If unsuccessful, EYE:WIDth? will return 9.91E+37 (Not-A-Number, NAN). The command :TCENter OFF aborts a previously started search.

NOTE

The clock/data align feature (used to center the sampling point in the data input eye) uses information derived from the input clock frequency.

For the clock/data align feature to work properly, the input frequency must be stable during the measurement. The frequencies at the start and end of the measurement are compared, and if they differ by more than 10%, the measurement fails.

When a source clocking the instrument changes frequency, it will take time for the instrument to sense the change and adjust its configuration. Refer to the sections dealing with clock stabilization to ensure that the instrument's configuration has stabilized following any change of frequency prior to performing a clock to data alignment.

There is no need to alter the sync-mode before or after a clock to data alignment procedure, as AUTO sync-mode is automatically configured for the duration of the procedure.

NOTE

The command :TCENter is an overlapped command. See the *Serial BERT Programming Guide*.

SENSe[1]:EYE:THReshold[?]

IVI-COM Equivalent IAgilentN490xEDSynchronisation.Threshold (IVI-compliant)

Syntax SENSe[1]:EYE:THReshold <Num.>

SENSe[1]:EYE:THReshold?

Description The command sets the BER threshold to be used in the determination of the edges of the eye.

The query returns the current BER threshold value.

SENSe[1]:EYE:WIDTh?

IVI-COM Equivalent IAgilentN490xEDSampling.ReadEyeWidth (IVI-compliant)

Syntax SENSe[1]:EYE:WIDTh?

Description This is a query that interrogates the eye width found by the most recent search for the value of data/clock delay that put the active edge in the center of the data eye.

If the result is not available or the search was unsuccessful, then 9.91E+37 (Not-A-Number, NAN) will be returned.

SENSe[1]:GATE Subnode

This subnode has the following SCPI structure:

```
SENSe[1]
└ :GATE
   ├ :BURSt[?]
   ├ :MANNer[?]
   ├ :MODE[?]
   ├ :PERiod
   │  ├ :BITS[?]
   │  ├ :ERRors[?]
   │  └ [:TIME][?]
   └ [:STATe][?]
```

This subnode has the following commands:

Table 45

Name	Description under
:BURSt[?]	“SENSe[1]:GATE:BURSt[?] ” on page 186
:MANNer[?]	“SENSe[1]:GATE:MANNer[?] ” on page 186
:MODE[?]	“SENSe[1]:GATE:MODE[?] ” on page 187
:PERiod:BITS[?]	“SENSe[1]:GATE:PERiod:BITS[?] ” on page 187
:PERiod:ERRors[?]	“SENSe[1]:GATE:PERiod:ERRors[?] ” on page 188
:PERiod[:TIMe][?]	“SENSe[1]:GATE:PERiod[:TIMe][?] ” on page 188

Table 45

Name	Description under
[:STATe][?]	"SENSe[1]:GATE[:STATe][?] " on page 188

SENSe[1]:GATE:BURSt[?]

IVI-COM Equivalent IAgilentN490xEDAccumulation.BurstGatingEnabled (not IVI-compliant)

Syntax SENSe[1]:GATE:BURSt 0|1|OFF|ON

Description The command turns the burst sync mode OFF or ON.

The query returns the current setting.

SENSe[1]:GATE:MANNer[?]

IVI-COM Equivalent IAgilentN490xEDAccumulation.PeriodMode (not IVI-compliant)

Syntax SENSe[1]:GATE:MANNer <Manner>

SENSe[1]:GATE:MANNer?

Return Range TIME | ERR | BITS

Description This command sets the manner in which the accumulation period is controlled. <Manner> can be one of the following:

- TIME

  The error detector performs SINGle and REPetitive accumulation periods that are controlled by elapsed time.

- ERRors

  The error detector performs SINGle and REPetitive accumulation periods that are controlled by the accumulation of bit errors.

- BITS

  The error detector performs SINGle and REPetitive accumulation periods that are controlled by the accumulation of clock bits.

The query returns the current manner of accumulation.

SENSe[1]:GATE:MODE[?]

IVI-COM Equivalent IAgilentN490xEDAccumulation.ActivationMode (IVI-compliant)

Syntax SENSe[1]:GATE:MODE <Mode>

SENSe[1]:GATE:MODE?

Return Range MAN | SING | REP

Description The command sets the accumulation period mode **<Mode>**, which can be either:

- MANual

  The accumulation is started manually (by sending GATE:STATe ON).

- SINGle

  Errors are accumulated over one accumulation period.

- REPetitive

  The accumulation loops, according to the setting of :GATe:MANNer.

Executing this command invalidates all past results. The query returns the current accumulation mode.

SENSe[1]:GATE:PERiod:BITS[?]

IVI-COM Equivalent IAgilentN490xEDAccumulation.Bits (IVI-compliant)

Syntax SENSe[1]:GATE:PERiod:BITS <Num.>

SENSe[1]:GATE:PERiod:BITS?

Description When GATE:MANNer is set to BITS, the duration of the accumulation period is set in clock bits (or periods). **<Num.>** can be a value between 1E7 and 1E15 in decade steps.

Executing this command invalidates all past results.

The query returns the number of bits to which the gate period is set.

SENSe[1]:GATE:PERiod:ERRors[?]

IVI-COM Equivalent IAgilentN490xEDAccumulation.Errors (IVI-compliant)

Syntax SENSe[1]:GATE:PERiod:ERRors <Num.>

SENSe[1]:GATE:PERiod:ERRors?

Description When GATE:MANNer is set to ERRors, the command sets the duration of the accumulation period in bit errors. Values of 10, 100 and 1000 are permitted.

Executing this command invalidates all past results.

The query returns the number of errors to which the gate period is set.

SENSe[1]:GATE:PERiod[:TIMe][?]

IVI-COM Equivalent IAgilentN490xEDAccumulation.Time (IVI-compliant)

Syntax SENSe[1]:GATE:PERiod[:TIME] <Num.>

SENSe[1]:GATE:PERiod[:TIME]?

Description When GATE:MANNer is set to TIME, the duration of the accumulation period is set in seconds. Neither a value less than 1 second nor greater than 8639999 seconds (99 days, 23 hours, 59 minutes and 59 seconds) is permitted.

The command causes all past results to be labelled as invalid.

The query returns the time to which the gate period is set.

SENSe[1]:GATE[:STATe][?]

IVI-COM Equivalent IAgilentN490xEDAccumulationGetState

Syntax SENSe[1]:GATE[:STATe] 0 | 1 | OFF | ON

SENSe[1]:GATE[:STATe]?

This command turns accumulation on or off.

NOTE

Previous commands that have altered the configuration of the instrument might not have settled. In order to ensure that the GATE ON command is not executed until conditions have settled, it is strongly recommended that the frequency be allowed to stabilize prior to the GATE ON command, and then be followed by a synchronization search. For more information, see "Allowing Serial BERT to Settle - Concepts" on page 22.

NOTE

When GATE:MODE SINGle is executed, the GATE[:STATe]ON command is an overlapped command. For more information, see "Overlapped and Sequential Commands " on page 74 the .

The query returns the current state of the accumulation gating.

SENSe[1]:PATTern Subnode

This subnode has the following SCPI structure:

```
SENSe[1]
└ :PATTern
  ├ :FORMat
  │ └ [:DATA][?]
  ├ :MDENsity
  │ └ [:DENSity][?]
  ├ [:SELect][?]
  ├ :TRACk[?]
  ├ :CAPTure
  ├ :UPATtern<n>
  │ └ . . .
  ├ :UFILe
  │ └ . . .
  └ :ZSUBstitut
    └ [:ZRUN][?]
```

This subnode has the following subnodes and commands:

Table 46

Name	Description under
Commands	
:FORMat[:DATA][?]	"SENSe[1]:PATTern:FORMat[:DATA][?] " on page 190
:MDENsity[:DENSity][?]	"SENSe[1]:BLOCk:BLENgth[?]" on page 175
[:SELect][?]	"SENSe[1]:PATTern[:SELect][?] " on page 191
:TRACk[?]	"SENSe[1]:PATTern:TRACk[?] " on page 193
:CAPTure	"SENSe[1]:PATTern:CAPTure" on page 194
:ZSUBstitut[:ZRUN][?]	"SENSe[1]:PATTern:ZSUBstitut[:ZRUN][?]" on page 194
Subnodes	
:UPATtern<n>	"SENSe[1]:PATTern:UPATtern Subnode" on page 195
:UFILe	"SENSe[1]:PATTern:UFILe Subnode" on page 201

SENSe[1]:PATTern:FORMat[:DATA][?]

IVI-COM Equivalent IAgilentN490xEDPatternfileGetData

Syntax SENSe[1]:PATTern:FORMat[:DATA] PACKed, <NR1>

SENSe[1]:PATTern:FORMat[:DATA]?

Input parameters **<PACKed>**: Permits the packing of bits within a byte to be set.

<NR1>: Can be 1, 4, or 8.

Return Range 1 | 4 | 8

Description The command controls the format of data transfer for the :PATTern:UPATtern<n>:DATA, :PATTern:UPATtern<n>:IDATa, :PATTern:UFILe:DATA and :PATTern:UFILe:IDATa commands. The following values are possible:

- 1

  The data is sent as a string of 1s and 0s.
- 4

  The data is sent as a string of hex characters.
- 8

  The data is sent as a string of full ASCII characters.

The query returns the current value of the data pack.

See "Working with User Patterns in SCPI" on page 65 for descriptions on how to use the data packing.

SENSe[1]:PATTern:MDENsity[:DENSity][?]

IVI-COM Equivalent IAgilentN490xEDDataIn.MarkDensity (not IVI-compliant)

Syntax SENSe[1]:PATTern:MDENsity[:DENSity] <NR2>

SENSe[1]:PATTern:MDENsity[:DENSity]?

Input Parameters **<NR2>**: 0.125, 0.25, 0.5, 0.75, 0.875

Description The command sets the ratio of high bits to the total number of bits in the pattern. The ratio may be varied in eighths, from one to seven (eighths), but excluding three and five.

The query returns the mark density in eighths.

SENSe[1]:PATTern[:SELect][?]

IVI-COM Equivalent IAgilentN490xEDDataIn.SelectData (IVI-compliant)

Syntax SENSe[1]:PATTern[:SELect] <Source>

SENSe[1]:PATTern[:SELect]?

Description

This command defines the type of pattern that is expected. The parameter is retained for backwards compatibility and may be one of the following:

PRBS<n>	<n> = 7, 10, 11, 15, 23, 31
PRBN<n>	<n> = 7, 10,11,13, 15, 23
ZSUBstitut<n>	<n> = 7, 10,11,13, 15, 23
UPATtern<n>	<n> = 1 through 12
MDENsity<n>	<n> = 7, 10,11,13, 15, 23
FILename,	<string>
BRM	Bit Recovery Mode

ZSUBstitut: **Z**ero **SUB**stitution; used for defining PRBN patterns in which a block of bits is replaced by a block of zeros. The length of the block is defined by "SENSe[1]:PATTern:ZSUBstitut[:ZRUN][?]" on page 194.

MDENsity: **M**ark **DEN**sity; used for defining a PRBN pattern in which the user can set the mark density. The mark density is set with "SENSe[1]:BLOCk:BLENgth[?]" on page 175.

UPATtern<n>: **U**ser **PAT**tern; used to define the contents of a pattern store. For the Serial BERT, <n> can be 1 - 12.

FILename: A parameter that allows the remote user to load a user pattern from the instrument's disk drive. This is the preferred mechanism for loading user patterns in the Serial BERT.

BRM: A parameter that puts the error detector into Bit Recovery Mode. In BRM, the error detector does not expect any specific data. This mode can be used if the incoming data is completely unknown. To set the initial sampling point, use "SENSe[1]:EYE:ALIGN:AUTO [?] " on page 181.

BRM cannot be set if the error detector is in Burst mode.

For measuring BER in BRM, the error detector uses an additional, invisible sampling point. The relative BER that can be measured in BRM is not necessarily the exact BER-the error counter results may differ from precise measurements. Only if the signal in the middle of the eye is free of errors, the relative BER matches the exact BER.

The error detector stays in BRM until a pattern is selected.

NOTE

Compared with usual operation using expected patterns, Bit Recovery Mode imposes some restrictions, for example:

- No error location capture possible
- No burst synchronization possible
- No Auto Synchronization possible
- No Synchronization Now possible

For details please refer to the User Guide.

The query form returns the pattern's types in short form.

NOTE

If a user-defined pattern is selected and the [:SELECT]? query is used, the response is UPAT. The particular value of <n> or the name of the file specified in the command form is not returned.

To get the path of a user pattern file, use the UFILe:NAME? command.

SENSe[1]:PATTern:TRACk[?]

IVI-COM Equivalent IAgilentN490xEDDataIn.TrackingEnabled (not IVI-compliant)

Syntax SENSe[1]:PATTern:TRACk 0 | 1 | OFF | ON

SENSe[1]:PATTern:TRACk?

Description This command enables and disables the error detector pattern tracking. When pattern tracking is enabled, the following commands sent to either the pattern generator or error detector are sent to both:

- SOURce[1] | SENSe[1] :PATTern:SELect[?]
- SOURce[1] | SENSe[1] :PATTern:FORMat[?]
- SOURce[1] | SENSe[1] :PATTern:MDENsity[:DENSity][?]
- SOURce[1] | SENSe[1] :PATTern:ZSUBstitut[:ZRUN][?]

The query returns the current state of the pattern track setting.

When pattern tracking is disabled, both instruments continue to use the current settings. Enabling pattern tracking causes the error detector to take over the settings of the pattern generator.

NOTE

Pattern tracking cannot be used in combination with user-defined sequences which generate up to four patterns and loops. Maintaining user-defined sequences is a feature of the pattern generator (see also “[SOURce[1]]:PATTern:SEQuence:DATA [?] ” on page 106). Pattern tracking is automatically disabled when a user-defined sequence is enabled.

SENSe[1]:PATTern:CAPTure

IVI-COM Equivalent

Syntax SENSe[1]:PATTern:CAPTure filename|Length|Description

Description File name:File where the captured data is stored as a standard pattern file.

Length:The captured pattern length in bits.

Description:a description for the captured pattern.

SENSe[1]:PATTern:ZSUBstitut[:ZRUN][?]

IVI-COM Equivalent IAgilentN490xEDDataIn.ZeroSub (not IVI-compliant)

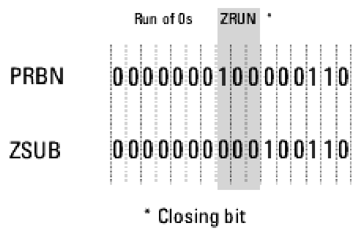
Syntax [SENSe[1]]:PATTern:ZSUBstitut[:ZRUN] MINimum | MAXimum | <numeric value>

[SENSe[1]]:PATTern:ZSUBstitut[:ZRUN]?

Return Range <NR3>

Description ZSUB patterns are PRBN patterns, where a number of bits are replaced by zeroes. The zero substitution starts after the longest runs of zeroes in the pattern (for example, for PRBN 2^7, after the run of 7 zeroes). This command allows you to define the length of the run of zeroes. For example, to produce 10 zeroes in a PRBN 2^7 pattern, three additional bits after the run of 7 zeroes must be replaced by zeroes. The bit after the run of zeroes (the closing bit) is set to 1.

The following figure shows an example, where a run of 10 zeroes is inserted into a PRBN 2^7 pattern.



This command is only active when a ZSUB pattern has been selected (see “SENSe[1]:PATTern[:SELect][?] ” on page 191).

Range The minimum value is the PRBN value - 1. The maximum value is length of the pattern - 1. So, for a PRBN 2^7 pattern, the minimum value is 6, and the maximum value is 127 (2^7 - 1).

SENSe[1]:PATTern:UPATtern Subnode

This subnode has the following SCPI structure:

```
SENSe[1]
└ :PATTern
   └ :UPATtern<n>
      ├ :DATA[?]
      ├ :IDATa[?]
      ├ [:LENGth][?]
      ├ :LABel[?]
      └ :USE[?]
```

This subnode has the following commands:

Table 48

Name	Description under
:DATA[?]	“SENSe [1]:PATTern:UPATtern<n>:DATA[?] ” on page 196
:IDATa[?]	“SENSe [1]:PATTern:UPATtern<n>:IDATa[?] ” on page 198
[:LENGth][?]	“SENSe[1]:PATTern:UPATtern<n> [:LENGth][?] ” on page 199

Table 48

Name	Description under
:LABel[?]	"SENSe [1]:PATTern:UPATtern<n>:LABel[?]" on page 199
:USE[?]	"SENSe[1]:PATTern:UPATtern<n>:USE [?] " on page 200

NOTE

For the UPATtern<n> commands, <n> can be in the range 0 - 12. 0 (zero) is used to select the current pattern, 1 - 12 selects one of the user patterns in the memory.

SENSe[1]:PATTern:UPATtern<n>:DATA[?]

IVI-COM Equivalent IAgilentN490xEDPatternfile.SetData (IVI-compliant)

Syntax SENSe[1]:PATTern:UPATtern<n>:DATA [A|B,] <block_data>

SENSe[1]:PATTern:UPATtern<n>:DATA? [A|B]

Return Range The query returns the standard (A) or alternate pattern (B) of the file found under <filename>.

Description This command is used to set the bits in user pattern files. See "Working with User Patterns in SCPI" on page 65 for a detailed description on how to edit user patterns.

The parameters have the following meanings:

Table 49

Parameter	Description
[A|B]	Defines for which pattern the data is to be set (A = standard pattern, B = alternate pattern). If the pattern file describes a standard pattern (:USE = STRaight), this parameter cannot be B.
<filename>	Name of the file being defined. If the file does not exist, it is created.

Table 49

Parameter	Description
<block data>	The data that describes the pattern (see the following for the description).

<block data>

The <block data> parameter contains the actual data for setting the bits of the user pattern. The bits can also be packed using the FORMat[:DATA] command. If the bits are not packed, they are handled as 8-bit data. See "[SOURce[1]]:PATTern:FORMat[:DATA][?]" on page 98.

This command also sets the pattern length to fit the length of the data: If the data block is longer than the pattern, the pattern is extended to fit the data; if the data block is shorter than the pattern, the pattern is truncated to the end of the data.

<block data> starts with a header that indicates the length of the desired resulting data. The length of the <block data> embedded in the header always refers to the length of the data block in bytes.

For example, consider the following header:

#19<data>

#	Start of the header.
1	Number of decimal digits to follow to form the length.
9	Length of the data block (in bytes) that follows.
<data>	The pattern data, packed according the DATA:PACKed command.

- For non-packed data (or 8-bit packed data), the <block data> required to set an 8-bit pattern of alternating 1s and 0s (01010101) would be:

  #11U (Note: "U" is the ASCII representation of binary 01010101.)

- For 4-bit packed data, the <block data> required to set the same pattern would be:

  #1255

- For 1-bit packed data, the <block data> would be as follows:

  #1801010101

SENSe[1]:PATTern:UPATtern<n>:IDATa[?]

IVI-COM Equivalent

IAgilentN490xEDPatternfile.SetDataBlock (IVI-compliant)

Syntax

SENSe[1]:PATTern:UPATtern<n>:IDATa [A|B,] <start_bit>, <length_in_bits>, <block_data>

SENSe[1]:PATTern:UPATtern<n>:IDATa? [A|B,] <start_bit>, <length_in_bits>

Return Range

The query returns the selected bits of the standard (A) or alternate (B) pattern of the file found under <filename>.

Description

This command is used to set specific bits in a user pattern. It is similar to the :DATA command. The :IDATa command is a contraction of the phrase **I**ncremental **DATA** and is used to download part of a user-defined pattern.

The parameters have the following meanings:

Table 51

Parameter	Description
[A|B]	Defines for which pattern the data is to be set (A = standard pattern, B = alternate pattern). If the pattern file describes a standard pattern (:USE = STRaight), this parameter cannot be B.
<filename>	Name of the file being defined. If the file does not exist, it is created.
<start bit>	First bit to be overwritten (starting with 0).
<length_in_bits>	Number of bits to be overwritten.
<block data>	The data that describes the pattern (see "SENSe [1]:PATTern:UPATtern<n>:DATA[?]" on page 196 for the description).

The use of the parameters can be best illustrated by an example. If we have an alternate 16-bit pattern of 0s only, and we want to set the last four bits to 1s, the IDATa command would appear as follows:

- If the data packing is 8:

  SENSe[1]:PATTern:UPAT1:IDATa B, <filename>, 12, 4, #11(&F0) (where (&F0) is replaced by the ASCII representation of the value)

- If the data packing is 4:

  SENSe[1]:PATTern:UPAT1:IDATa B, <filename>, 12, 4, #11F

- If the data packing is 1:

  SENSe[1]:PATTern:UPAT1:IDATa B, <filename>, 12, 4, #141111

The response form returns <block data> at the specified location.

NOTE

See "Working with User Patterns in SCPI" on page 65 for more information on using this command.

SENSe[1]:PATTern:UPATtern<n>[:LENGth][?]

IVI-COM Equivalent IAgilentN490xEDPatternfile.Length (IVI-compliant)

Syntax SENSe[1]:PATTern:UPATtern<n>[:LENGth] <Num.>

SENSe[1]:PATTern:UPATtern<n>[:LENGth]?

Description This command sets the length of a user pattern file. The query returns the length of the user pattern file. If an alternate pattern is selected (:USE APATtern), the LENGth command sets the length of each half of the pattern.

Note that the :DATA command sets the length of the file.

See "Working with User Patterns in SCPI" on page 65 for information on using this command.

SENSe[1]:PATTern:UPATtern<n>:LABel[?]

IVI-COM Equivalent IAgilentN490xEDPatternfile.Description (IVI-compliant)

Syntax SENSe[1]:PATTern:UPATtern<n>:LABel <string>

SENSe[1]:PATTern:UPATtern<n>:LABel?

Description This command sets a description for a user pattern file. The query returns the description. See "Working with User Patterns in SCPI" on page 65 for information on using this command.

SENSe[1]:PATTern:UPATtern<n>:USE[?]

IVI-COM Equivalent IAgilentN490xEDPatternfile.Alternate (IVI-compliant)

Syntax SENSe[1]:PATTern:UPATtern<n>:USE STRaight | APATtern

SENSe[1]:PATTern:UPATtern<n>:USE?

Return Range STR | APAT

Description This command defines whether a user pattern file should be a straight pattern or an alternate pattern:

- STRaight

  The pattern is repeatedly output.

- APATtern

  The pattern is composed of two halves. The output depends on various other commands; see "How the Serial BERT Uses Alternate Patterns" on page 57 for more information.

The default is set to have a length of 128 bits for each half pattern; all bits are set to zero and the trigger is set to occur on the A/B changeover. See "Working with User Patterns in SCPI" on page 65 for information on using this command.

SENSe[1]:PATTern:UFILe Subnode

This subnode has the following SCPI structure:

```
SENSe[1]
└ :PATTern
   └ :UFILe
      ├ [:LENGth][?]
      ├ :LABel[?]
      ├ :USE[?]
      ├ :DATA[?]
      ├ :IDATa[?]
      └ :NAME?
```

This subnode has the following commands:

Table 52

Name	Description under
[:LENGth][?]	"SENSe[1]:PATTern:UFILe[:LENGth][?] " on page 201
:LABel[?]	"SENSe[1]:PATTern:UFILe:LABel[?] " on page 202
:USE[?]	"SENSe[1]:PATTern:UFILe:USE[?] " on page 202
:DATA[?]	"SENSe[1]:PATTern:UFILe:DATA[?] " on page 203
:IDATa[?]	"SENSe[1]:PATTern:UFILe:IDATa[?] " on page 204
:NAMe[?]	"SENSe[1]:PATTern:UFILe:NAMe[?] " on page 206

SENSe[1]:PATTern:UFILe[:LENGth][?]

IVI-COM Equivalent IAgilentN490xLocalPatternfile.Length (IVI-compliant)

Syntax SENSe[1]:PATTern:UFILe[:LENGth] <filename>, <numeric_value>

SENSe[1]:PATTern:UFILe[:LENGth]? <filename>

Description This command sets the length of a user pattern file. The query returns the length of the user pattern file. If an alternate pattern is selected (:USE APATtern), the LENGth command sets the length of each half of the pattern.

Note that the :DATA command automatically sets the length of the file.

See "Working with User Patterns in SCPI" on page 65 for information on using this command.

SENSe[1]:PATTern:UFILe:LABel[?]

IVI-COM Equivalent IAgilentN490xPGPatternfile.Description (IVI-compliant)

Syntax SENSe[1]:PATTern:UFILe:LABel <filename>, <string>

SENSe[1]:PATTern:UFILe:LABel? <filename>

Description This command sets a description for a user pattern file. The query returns the description. See "Working with User Patterns in SCPI" on page 65 for information on using this command.

SENSe[1]:PATTern:UFILe:USE[?]

IVI-COM Equivalent IAgilentN490xPGPatternfile.Alternate (IVI-compliant)

Syntax SENSe[1]:PATTern:UFILe:USE <filename>, STRaight | APATtern

SENSe[1]:PATTern:UFILe:USE? <filename>

Description This command defines whether a user pattern file should be a straight pattern or an alternate pattern:

- STRaight

  The pattern is repeatedly output.

- APATtern

The pattern is composed of two halves. The output depends on various other commands; see "How the Serial BERT Uses Alternate Patterns" on page 57 for more information.

The default is set to have a length of 128 bits for each half pattern; all bits are set to zero and the trigger is set to occur on the A/B changeover.

SENSe[1]:PATTern:UFILe:DATA[?]

IVI-COM Equivalent IAgilentN490xPGPatternfile.SetData (IVI-compliant)

Syntax SENSe[1]:PATTern:UFILe:DATA [A | B,] <filename>, <block_data>

SENSe[1]:PATTern:UFILe:DATA? [A|B,] <filename>

Return Range The query returns the standard (A) or alternate pattern (B) of the file found under <filename>.

Description This command is used to set the bits in user pattern files. See "Working with User Patterns in SCPI" on page 65 for a detailed description on how to edit user patterns.

The parameters have the following meanings:

Table 53

Parameter	Description
[A|B]	Defines for which pattern the data is to be set (A = standard pattern, B = alternate pattern). If the pattern file describes a standard pattern (:USE = STRaight), this parameter cannot be B.
<filename>	Name of the file being defined. If the file does not exist, it is created.
<block data>	The data that describes the pattern (see the following for the description).

<block data> The <block data> parameter contains the actual data for setting the bits of the user pattern. The bits can also be packed using the FORMat[:DATA] command. If the bits are not packed, they are

handled as 8-bit data. See "[SOURce[1]]:PATTern:FORMat[:DATA][?]" on page 98.

This command also sets the pattern length to fit the length of the data: If the data block is longer than the pattern, the pattern is extended to fit the data; if the data block is shorter than the pattern, the pattern is truncated to the end of the data.

<block data> starts with a header that indicates the length of the desired resulting data. The length of the <block data> embedded in the header always refers to the length of the data block in bytes.

For example, consider the following header:

#19<data>

#	Start of the header.
1	Number of decimal digits to follow to form the length.
9	Length of the data block (in bytes) that follows.
<data>	The pattern data, packed according the DATA:PACKed command.

- For non-packed data (or 8-bit packed data), the <block data> required to set an 8-bit pattern of alternating 1s and 0s (01010101) would be:

  #11U (Note that "U" is the ASCII representation of 85)
- For 4-bit packed data, the <block data> required to set the same pattern would be:

  #1255
- For 1-bit packed data, the <block data> would be as follows:

  #1801010101

SENSe[1]:PATTern:UFILe:IDATa[?]

IVI-COM Equivalent IAgilentN490xPGPatternfile.SetDataBlock (IVI-compliant)

Syntax SENSe[1]:PATTern:UFILe:IDATa [A | B,] <filename>, <start_bit>, <length_in_bits>, <block_data>

SENSe[1]:PATTern:UFILe:IDATa? [A|B,] <filename>, <start_bit>, <length_in_bits>

Return Range

The query returns the selected bits of the standard (A) or alternate (B) pattern of the file found under <filename>.

Description

This command is used to set specific bits in a user pattern. It is similar to the :DATA command. The :IDATa command is a contraction of the phrase **I**ncremental **DATA** and is used to download part of a user-defined pattern.

The parameters have the following meanings:

Table 55

Parameter	Description
[A|B]	Defines for which pattern the data is to be set (A = standard pattern, B = alternate pattern). If the pattern file describes a standard pattern (:USE = STRaight), this parameter cannot be B.
<filename>	Name of the file being defined. If the file does not exist, it is created.
<start bit>	First bit to be overwritten (starting with 0).
<length_in_bits>	Number of bits to be overwritten.
<block data>	The data that describes the pattern (see "[SOURce[1]]:PATTern:UFILe:DATA[?] " on page 114 for the description).

The use of the parameters can be best illustrated by an example. If we have an alternate 16-bit pattern of 0s only, and we want to set the last four bits to 1s, the IDATa command would appear as follows:

- If the data packing is 8:

  SOURce1:PATTern:UFILe:IDATa B, <filename>, 12, 4, #11(&F0) (where (&F0) is replaced by the ASCII representation of the value)

- If the data packing is 4:

  SOURce1:PATTern:UFILe:IDATa B, <filename>, 12, 4, #11F

- If the data packing is 1:

  SOURce1:PATTern:UFILe:IDATa B, <filename>, 12, 4, #141111

The response form returns <block data> at the specified location.

NOTE

See “Working with User Patterns in SCPI” on page 65 for more information on using this command.

SENSe[1]:PATTern:UFILe:NAMe[?]

IVI-COM Equivalent: IAgilentN490xEDDataInGetSelectData

Syntax: SENSe[1]:PATTern:UFILe:NAME?

Description: This query returns the file name of the currently used user pattern. It is only valid if SENSe[1]:PATTern:SELect? returns UPAT.

SENSe[1]:VOLTage Subnode

This subnode has the following SCPI structure:

```
SENSe[1]
└ :ZOTHreshold[?]
   ├ :RANGe
   │  ├ [:HIGH][?]
   │  └ :LOW[?]
   └ :AUTO[?]
```

This subnode has the following commands:

Table 56

Name	Description under
:ZOTHreshold[?]	“SENSe[1]:VOLTage:ZOTHreshold[?] ” on page 207
:ZOTHreshold:RANGe[:HIGH][?]	“SENSe [1]:VOLTage:ZOTHreshold:RANGe [:HIGH][?] ” on page 207

Table 56

Name	Description under
:ZOTHreshold:RANGe:LOW[?]	"SENSe [1]:VOLTage:ZOTHreshold:RANGe:LOW [?] " on page 208
:ZOTHreshold:AUTO[?]	"SENSe [1]:VOLTage:ZOTHreshold:AUTO[?] " on page 208

SENSe[1]:VOLTage:ZOTHreshold[?]

IVI-COM Equivalent IAgilentN490xEDSampling.ZeroOneThreshold (IVI-compliant)

Syntax SENSe[1]:VOLTage:ZOTHreshold <Num.>

SENSe[1]:VOLTage:ZOTHreshold?

Description This command sets the level at which the error detector discriminates between a 0 and a 1.

A numeric value parameter sets the level to a given value in Volts. It also sets :ZOTHreshold:AUTO to OFF.

When in :ZOTHreshold:AUTO OFF, the query returns the last user-entered value.

When in :ZOTHreshold:AUTO ON, the query returns the value automatically determined by the hardware.

If you are going to use this command to set the 0/1 threshold, first disable the automatic mode with :ZOTHreshold:AUTO OFF (see "SENSe[1]:VOLTage:ZOTHreshold:AUTO[?] " on page 208).

SENSe[1]:VOLTage:ZOTHreshold:RANGe[:HIGH][?]

IVI-COM Equivalent IAgilentN490xEDSampling.ZeroOneThresholdVHigh (not IVI-compliant)

Syntax SENSe[1]:VOLTage:ZOTHreshold:RANGe[:HIGH] <NR3>

SENSe[1]:VOLTage:ZOTHreshold:RANGe[:HIGH]?

Description — Sets/returns the higher limit of the input range of the zero/one threshold.

SENSe[1]:VOLTage:ZOTHreshold:RANGe:LOW[?]

IVI-COM Equivalent — IAgilentN490xEDSampling.ZeroOneThresholdVLow (not IVI-compliant)

Syntax — SENSe[1]:VOLTage:ZOTHreshold:RANGe:LOW <NR3>

SENSe[1]:VOLTage:ZOTHreshold:RANGe:LOW?

Description — Sets/returns the lower limit of the input range of the zero/one threshold.

SENSe[1]:VOLTage:ZOTHreshold:AUTO[?]

IVI-COM Equivalent — IAgilentN490xEDSampling.ZeroOneThresholdTrack (not IVI-compliant)

Syntax — SENSe[1]:VOLTage:ZOTHreshold:AUTO 0 | 1 | OFF | ON

SENSe[1]:VOLTage:ZOTHreshold:AUTO?

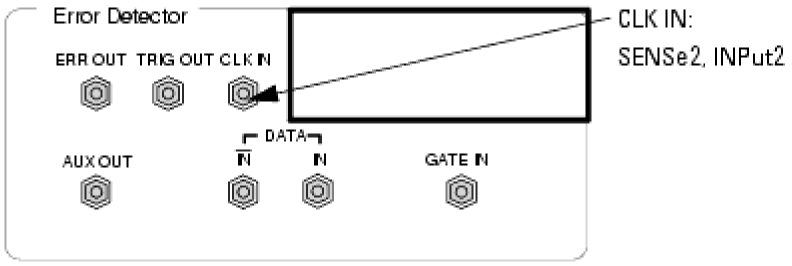
Description — This command enables an automatic mode, in which the 0/1 threshold level is set to the mean of the input signal.

The query returns the current setting of the hardware discrimination circuit.

INPut2 Subsystem

INPut2 Subsystem - Reference

The INPut2 Subsystem represents the error detector's Clock In port.



This subsystem has the following SCPI structure:

```
INPut2
├ :TERMination[?]
└ :COUPling[?]
```

This subsystem has the following commands:

INPut2:TERMination[?]

IVI-COM Equivalent IAgilentN490xEDClockInVTermination

Syntax INPut2:TERMination 0 | -2 | 1.3

INPut2:TERMination?

This command is obsolete. It has no effect.

INPut2:COUPling[?]

IVI-COM Equivalent IAgilentN490xEDClockInTerminationEnabled

Syntax INPut2:COUPling AC | DC

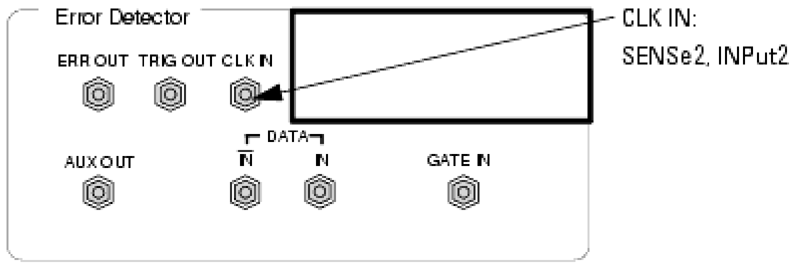
INPut2:COUPling?

This command is obsolete. It has no effect.

SENSe2 Subsystem

SENSe2 Subsystem - Reference

The SENSe2 Subsystem controls the error detector's Clock In port.



This subsystem has the following SCPI structure:

```
SENSe2
├ :FREQuency
│ ├ [:CW | :FIXed]?
│ └ :CDR[?]
│   ├ :FADJust[?]
│   ├ :LBANdwidth[?]
│   ├ :PEAKing[?]
│   │ └ :AVAilable?
│   ├ :SSCLocking[?]
│   │ └ :DEViation[?]
│   ├ :TDENsity[?]
│   │ └ :MEASure?
│   ├ :RANGe?
│   └ :THReshold
│     ├ [:VALue][?]
│     ├ :AUTO[?]
│     └ :MEASure[?]
└ :VOLTage
  └ :EDGE[?]
```

This subsystem has the following commands:

Table 57

Name	Description under
:FREQuency[:CW | FIXed]?	“SENSe2:FREQuency[:CW|:FIXed]? ” on page 212
:FREQuency:CDR[?]	“SENSe2:FREQuency:CDR[?] ” on page 212
:FREQuency:CDR:FADJust[?]	“SENSe2:FREQuency:CDR:FADJust[?]” on page 214
:FREQuency:CDR:LBANdwidth[?]	“SENSe2:FREQuency:CDR:LBANdwidth [?]” on page 214
:FREQuency:CDR:PEAKing[?]	“SENSe2:FREQuency:CDR:PEAKing[?]” on page 214
:FREQuency:CDR:PEAKing:AVAilable[?]	“SENSe2:FREQuency:CDR:PEAKing:AVAilable?” on page 214
:FREQuency:CDR:SSCLocking[?]	“SENSe2:FREQuency:CDR:SSCLocking [?]” on page 215
:FREQuency:CDR:SSCLocking:DEViation [?]	“SENSe2:FREQuency:CDR:SSCLocking: DEViation[?]” on page 215
:FREQuency:CDR:TDENsity[?]	“SENSe2:FREQuency:CDR:TDENsity[?]” on page 216
:FREQuency:CDR:TDENsity:MEASure[?]	“SENSe2:FREQuency:CDR:TDENsity:MEASure?” on page 216
:FREQuency:CDR:RANGe?	“SENSe2:FREQuency:CDR:RANGe? ” on page 215
:FREQuency:CDR:AUTO[?]	“SENSe2:FREQuency:CDR:THReshold:AUTO[?]” on page 216
:FREQuency:CDR:THReshold[:VALue][?]	“SENSe2:FREQuency:CDR:THReshold [:VALue][?]” on page 217
:FREQuency:CDR:THReshold:MEASure [?]	“SENSe2:FREQuency:CDR:THReshold: MEASure[?]” on page 217

Table 57

Name	Description under
:VOLTage:EDGe[?]	“SENSe2:VOLTage:EDGe[?] ” on page 217

NOTE

Depending on the options of your Serial BERT, the commands relating to clock data recovery (CDR) may not be available. See the online Help or the User's Guide for a description of the available options.

SENSe2:FREQuency[:CW|:FIXed]?

IVI-COM Equivalent IAgilentN490xEDClockIn.GetFrequency (IVI-compliant)

Syntax SENSe2:FREQuency[:CW|:FIXed]?

Description This query returns the internal clock source frequency. You can use any of the forms listed below:

- SENSe2:FREQuency?
- SENSe2:FREQuency:CW?
- SENSe2:FREQuency:FIXed?

These forms have the same effect.

SENSe2:FREQuency:CDR[?]

IVI-COM Equivalent IAgilentN490xEDCDR.Enabled (not IVI-compliant)

Syntax SENSe2:FREQuency:CDR ON | OFF

SENSe2:FREQuency:CDR?

Description Enables or disables clock data recovery (CDR) mode.

How Does Clock Data Recovery Work? In CDR mode, the CDR has to recover the clock from the incoming data. To do this, the hardware has to decide whether the voltage at the input connector is a logical '1' or '0' and then recover the clock from the detected transitions.

Because the regular threshold voltage is not only used to determine the optimum sampling for the data, but also to perform measurements such as eye diagram or output level measurements, it is not possible to use it for the clock recovery.

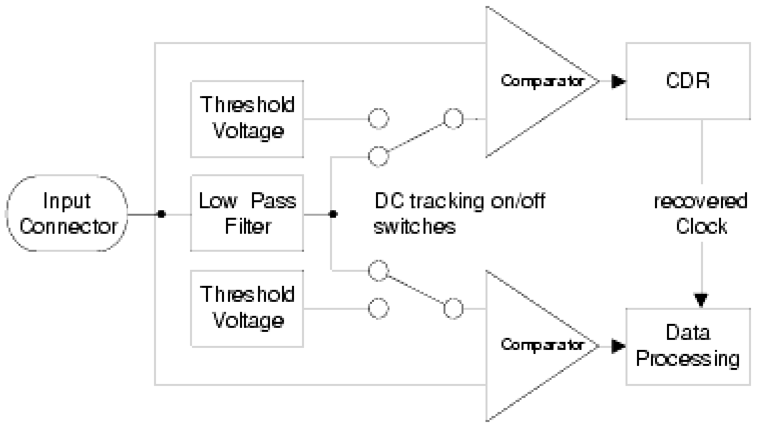
For this reason, the clock recovery circuitry has it's own comparator for the incoming data. This comparator also needs to know the threshold voltage (0/1 decision threshold).

The threshold voltage can be derived from the input signal via a low-pass filter. This will work fine for most applications. But applications that do not provide a continuous data stream at the input (for example, any application using bursts) cannot use this low-pass filter, because the threshold voltage will drift from the correct level when there is no input. In such cases, the threshold can be specified manually. It is then no longer derived from the input signal (see the following figure). The manually set threshold voltage must of course be within the input range.

The difference between the data path and the CDR path is that the comparator of the CDR is always single-ended. Thus, this comparator always needs a threshold voltage that lies between the high and low levels of the incoming signal.

The differential threshold of the data path comparator has no relation to the single-ended threshold of the CDR path comparator. This means that in differential mode, the two thresholds will be different and in single-ended mode (either normal and complement) they will/can be equal (except during measurements).

The following figure shows a simplified block diagram. It does not reflect the different input modes (especially the differential case), but it matches both single-ended cases.



SENSe2:FREQuency:CDR:FADJust[?]

Syntax

```
SENSe2:FREQuency:CDR:FADJust <NR3>
SENSe2:FREQuency:CDR:FADJust?
```

Description

This command allows fine adjustments of the CDR hardware, to improve its jitter response in some circumstances. The allowed values are between -1 and +1. Default is 0.

SENSe2:FREQuency:CDR:LBANdwidth[?]

Syntax

```
SENSe2:FREQuency:CDR:LBANdwidth <NR3>
SENSe2:FREQuency:CDR:LBANdwidth?
```

Description

Sets/gets the loop bandwidth of the CDR. It can be between 100 kHz and 12 MHz, but the allowed values depend on the clock frequency of the CDR. For details, please refer to the Technical Data Sheet.

SENSe2:FREQuency:CDR:PEAKing[?]

Syntax

```
SENSe2:FREQuency:CDR:PEAKing <NR3>
SENSe2:FREQuency:CDR:PEAKing?
```

Description

Sets/gets the peaking of the CDR, in dB. The allowed values depend on both the clock frequency of the CDR and its loop bandwidth, and can be queried with SENS2:FREQ:PEAK:AVA? The peaking value will automatically change either after the change in the CDR frequency or after the change of the loop bandwidth. Peaking should be queried again by the remote program after such a change in the CDR parameters.

SENSe2:FREQuency:CDR:PEAKing:AVAilable?

Syntax

```
SENSe2:FREQuency:CDR:PEAKing:AVAilable?
```

Description

Gets a comma-separated list of allowed peaking values for the current CDR settings. The command SENS2:FREQ:CDR:PEAK will only accept one of the the values returned by this query. These values change either when the CDR frequency or its loop bandwidth is modified. The values are expressed in dB.

SENSe2:FREQuency:CDR:RANGe?

IVI-COM Equivalent IAgilentN490xEDCDR.GetRanges (not IVI-compliant)

Syntax SENSe2:FREQuency:CDR:RANGe?

Description Comma-separated list of CDR (Hertz) ranges: "min, max, min, max, min, max"

For example: 2.45e+009,3.21e+009,4.9e+009,6.42e+009,9.9e+009,1.09e+010

This indicates the following CDR ranges:

- 2.45 - 3.21 GHz
- 4.9 - 6.42 GHz
- 9.9 - 10.9 GHz

SENSe2:FREQuency:CDR:SSCLocking[?]

Syntax SENSe2:FREQuency:CDR:SSCLocking ON|OFF|0|1

SENSe2:FREQuency:CDR:SSCLocking?

Description Either enables or disables the CDR support for spread spectrum clocking. When enabled, the CDR is able to extract a clock that deviates from the nominal CDR frequency value up to the deviation value set via the SCPI command SENS2:FREQ:CDR:SSCL:DEV.

SENSe2:FREQuency:CDR:SSCLocking:DEViation[?]

Syntax SENSe2:FREQuency:CDR:SSCLocking:DEViation <NR3>

SENSe2:FREQuency:CDR:SSCLocking:DEViation?

Description Sets/gets the amount of spread spectrum clocking,in percent, that the CDR can follow. The allowed values are between 0 and 0.5. The interval is defined asymmetrically; for a value of e.g. 0.3, the clock may vary between the nominal CDR frequency, and 99.7% of this value (i.e. a deviation of -0.3%). The clock may not be above the nominal CDR frequency.

SENSe2:FREQuency:CDR:TDENsity[?]

Syntax

SENSe2:FREQuency:CDR:TDENsity <NR3>

SENSe2:FREQuency:CDR:TDENsity?

Description

Sets/gets the transition density of the incoming signal, in percent. The allowed values are between 25 and 100. This value is internally used by the CDR to adjust its gain for locking on the incoming data.

Example: A 1010 pattern has a transition density of 100%, while a 1100 pattern has a transition density of 50%.

SENSe2:FREQuency:CDR:TDENsity:MEASure?

Syntax

SENSe2:FREQuency:CDR:TDENsity:MEASure?

Description

Tries to measure the transition density of the incoming data, and returns the measured value, in percent. This measurement only succeeds if the CDR is locked; otherwise, an error is placed in the error queue. The programmer needs to write the measured value in the hardware via SENS2:FREQ:CDR:TDEN, if desired (this is not done by the measurement).

SENSe2:FREQuency:CDR:THReshold:AUTO[?]

IVI-COM Equivalent

IAgilentN490xEDCDR.ThresholdAutoTracking (not IVI-compliant)

Syntax

SENSe2:FREQuency:CDR:THReshold:AUTO 0|1|OFF|ON

SENSe2:FREQuency:CDR:THReshold:AUTO?

Description

Enables/disables the automatic tracking for the CDR threshold. The query returns the current setting. Default is ON.

The automatic tracking should be disabled for burst applications.

SENSe2:FREQuency:CDR:THReshold[:VALue][?]

IVI-COM Equivalent IAgilentN490xEDCDR.Threshold (not IVI-compliant)

Syntax SENSe2:FREQuency:CDR:THReshold:VALue <NR3>

SENSe2:FREQuency:CDR:THReshold:VALue?

Description Sets/gets the manual threshold for CDR bit transitions.

SENSe2:FREQuency:CDR:THReshold:MEASure[?]

IVI-COM Equivalent IAgilentN490xEDCDR.MeasureAndSetThreshold (not IVI-compliant)

Syntax SENSe2:FREQuency:CDR:THReshold:MEASure ONCe

SENSe2:FREQuency:CDR:THReshold:MEASure?

Description The command measures the DC level at the CDR input and sets the measured value as CDR threshold.

The query returns the current DC level at the CDR input.

SENSe2:VOLTage:EDGe[?]

IVI-COM Equivalent IAgilentN490xEDClockIn.ActiveEdge (IVI-compliant)

Syntax SENSe2:VOLTage:EDGe NEGative | POSitive

SENSe2:VOLTage:EDGe?

Description Sets the active edge of the clock input:

- NEGative

  The falling edge starts the period in which the input data is sampled.

- POSitive

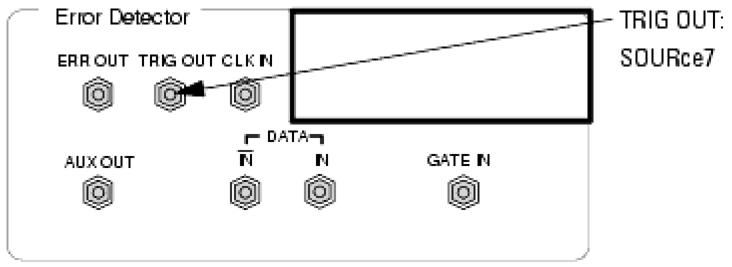
  The rising edge starts the period in which the input data is sampled.

This command has restrictions for frequencies under 620Mbits/s. See for details. See the User Guide (or online Help) for details.

SOURce7 Subsystem

SOURce7 Subsystem - Reference

SOURce7 represents the error detector's Trigger Out port.



This subsystem has the following SCPI structure:

```
SOURce7
└ :TRIGger
   ├ [:MODE][?]
   ├ DCDRatio
   └ CTDRatio?
```

This subsystem has the following commands:

SOURce7:TRIGger[:MODE][?]

IVI-COM Equivalent IAgilentN490xEDTrigger.Mode (IVI-compliant)

Syntax SOURce7:TRIGger[:MODE] DCLock | PATTern

SOURce7:TRIGger[:MODE]?

Description The command configures the error detector's Trigger Out port from the error detector to be either:

- DCLock

  Divided clock mode (a square wave at clock rate/8)

- PATTern

  Pattern mode (a pulse synchronized to repetitions of the pattern)

The query returns current mode for the Trigger Out of the error detector.

SOURce7:TRIGger:DCDRatio

IVI-COM Equivalent	IAgilentN490xEDTrigger.ClockDivisionRate (IVI-compliant)
Syntax	SOURce7:TRIGger:DCDRatio <NR1>
Description	Sets the trigger subratio.

SOURce7:TRIGger:CTDRatio?

IVI-COM Equivalent	IAgilentN490xEDTrigger.ClockDivisionRate (IVI-compliant)
Syntax	SOURce7:TRIGger:CTDratio?
Description	Returns the trigger subratio.

[P]FETCh Subsystem

[P]FETCh Subsystem - Reference

The [P]FETCh Subsystem is used to query the error detector's results. The PFETch subsystem returns the results immediately previously to the current results.

This subsystem has the following SCPI structure:

```
[P]FETCh
 ├─ [:SENSe[1]]
 │   └─ . . .
 └─ :SENSe2
     ├─ :BCOunt?
     └─ :FREQuency
         └─ [:CW|:FIXed]?
```

This subsystem has the following commands:

Table 58

Name	Description under
Commands	
SENSe2:BCOunt?	“[P]FETCh:SENSe2:BCOunt? ” on page 220
SENSe2:FREQ[:CW|:FIXed]?	“[P]FETCh:SENSe2:FREQuency [:CW|:FIXed]? ” on page 221
Subnode	
[SENSe[1]]	“[P]FETCh[:SENSe[1]] Subnode” on page 221

[P]FETCh:SENSe2:BCOunt?

IVI-COM Equivalent IAgilentN490xEDMeasurement.ReadBitCount (IVI-compliant)

Syntax [P]FETCh:SENSe2:BCOunt?

Description This query returns the accumulated bit count since the start of the accumulation period.

[P]FETCh:SENSe2:FREQuency[:CW|:FIXed]?

IVI-COM Equivalent IAgilentN490xEDMeasurement.ReadClockFrequency (IVI-compliant)

Syntax [P]FETCh:SENSe2:FREQuency[:CW|:FIXed]?

Description This query returns the current frequency of the signal on the clock input. This measurement is independent of the accumulation period.

[P]FETCh[:SENSe[1]] Subnode

This subnode has the following SCPI structure:

```
[P]FETCh
└ [:SENSe[1]]
   ├ :BURSt
   │  └ ...
   ├ :ECOunt
   │  └ ...
   ├ :EFINterval
   │  └ ...
   ├ :EINTerval
   │  └ ...
   ├ :G821
   │  └ ...
   ├ :ERATio
   │  └ ...
   ├ :GATe
   │  └ :ELAPsed?
   ├ :OASZero
   │  └ [:TOTal]?
   ├ :ZASone
   │  └ [:TOTal]?
   └ :LOSS
      ├ :POWer?
      └ :SYNChronizat?
```

This subnode has the following commands and subnodes:

Table 59

Name	Description under
Commands	
:GATe:ELAPsed?	"[P]FETCh[:SENSe[1]]:GATe:ELAPsed? " on page 222
:LOSS:POWer?	"[P]FETCh[:SENSe[1]]:LOSS:POWer? " on page 223
:LOSS:SYNChronizat?	"[P]FETCh[:SENSe [1]]:LOSS:SYNChronizat? " on page 223
Subnodes	
:BURSt	"[P]FETCh[:SENSe[1]]:BURSt Subnode" on page 223
:ECOunt	"[P]FETCh[:SENSe[1]]:ECOunt Subnode" on page 226
:EFINterval	"[P]FETCh[:SENSe[1]]:EFINterval Subnode" on page 228
:EINTerval	"[P]FETCh[:SENSe[1]]:EINTerval Subnode" on page 230
:ERATio	"[P]FETCh[:SENSe[1]]:ERATio Subnode" on page 232

[P]FETCh[:SENSe[1]]:GATe:ELAPsed?

IVI-COM Equivalent IAgilentN490xEDAccumulation.FetchElapsed (IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:GATe:ELAPsed?

Description This query returns information about the degree to which the accumulation period has progressed.

If SENSe[1]:GATE:MANNer TIME is selected, then this command returns the elapsed time in the accumulation period in units of seconds.

If SENSe[1]:GATE:MANNer ERRors is selected, this command returns the elapsed errors into the accumulation period.

If SENSe[1]:GATE:MANNer BITS is selected, then this command returns the elapsed clock bits into the accumulation period.

[P]FETCh[:SENSe[1]]:LOSS:POWer?

Syntax [P]FETCh[:SENSe[1]]:LOSS:POWer?

Description This query returns the total number of seconds that power was lost since the start of the accumulation period.

This command is not supported.

[P]FETCh[:SENSe[1]]:LOSS:SYNChronizat?

IVI-COM Equivalent IAgilentN490xEDMeasurement.ReadSecondsSyncLoss (IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:LOSS:SYNChronizat?

Description This query returns of the total number of seconds for which the incoming pattern was not synchronized to the reference pattern during the accumulation period.

[P]FETCh[:SENSe[1]]:BURSt Subnode

This subnode has the following SCPI structure:

```
[P]FETCh
└ [:SENSe[1]]
   └ :BURSt
      ├ :BCOunt?
      ├ :DCYCle?
      ├ :SRATio?
      ├ :TCOunt?
      └ :STATe?
```

This subnode has the following commands:

Table 60

Name	Description under
:BCOunt?	"[P]FETCh[:SENSe[1]]:BURSt:BCOunt? " on page 224
:DCYCle?	"[P]FETCh[:SENSe[1]]:BURSt:DCYCle? " on page 224
:SRATio?	"[P]FETCh[:SENSe[1]]:BURSt:SRATio? " on page 225
:TCOunt?	"[P]FETCh[:SENSe[1]]:BURSt:TCOunt? " on page 225
:STATe?	"[P]FETCh[:SENSe[1]]:BURSt:STATe?" on page 225

[P]FETCh[:SENSe[1]]:BURSt:BCOunt?

IVI-COM Equivalent IAgilentN490xEDBurst.ReadBadCount (not IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:BURSt:BCOunt?

Description This query returns the **B**ad Burst **CO**unt since the start of the accumulation period. If Burst mode is OFF, it returns 9.91E+37 (Not-A-Number, NAN).

For more information on how the Serial BERT handles burst measurements, see the Serial BERT User Guide (or online Help).

[P]FETCh[:SENSe[1]]:BURSt:DCYCle?

IVI-COM Equivalent IAgilentN490xEDBurst.ReadDutyCycle (not IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:BURSt:DCYCle?

Description This query is obsolete. It returns 9.91E+37 (Not-A-Number, NAN).

[P]FETCh[:SENSe[1]]:BURSt:SRATio?

IVI-COM Equivalent	IAgilentN490xEDBurst.ReadSyncRatio (not IVI-compliant)
Syntax	[P]FETCh[:SENSe[1]]:BURSt:SRATio?
Description	This query returns the Burst **S**ynchronization **RAT**io since the start of the accumulation period. If Burst mode is OFF, it returns 9.91E +37 (Not-A-Number, NAN).

[P]FETCh[:SENSe[1]]:BURSt:TCOunt?

IVI-COM Equivalent	IAgilentN490xEDBurst.ReadTotalCount (not IVI-compliant)
Syntax	[P]FETCh[:SENSe[1]]:BURSt:TCOunt?
Description	This query returns the **T**otal Burst **CO**unt since the start of the accumulation period. If Burst mode is OFF, it returns 9.91E+37 (Not-A-Number, NAN).

[P]FETCh[:SENSe[1]]:BURSt:STATe?

IVI-COM Equivalent IAgilentN490xEDBurst.ReadState (not IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:BURSt:STATe?

Description This query returns the burst state. While the accumulation period is running, the burst state always indicates, for example, if the current gate is too long or not. Thus, if the duty cycle or frequency of the gate signal is changed, the burst state also changes.

It may be of interest to know if an errored state occurred during a measurement. For this reason, any errored state is stored even if the warning was only active for one burst out of thousands.

The following states may be returned:

- `BURST_RESULT_STATE__NO_ERROR`

  No burst errors have occurred (any of those listed below).

- `BURST_RESULT_STATE__GATE_SIGNAL_TOO_LONG`

The Gate In signal is too long. This error can occur if there are too many bits within a burst. The limit is 4Gbit. The length of the Gate In signal therefore depends on the bit rate.

At 13 Gb/s, this state occurs roughly after 0.3 s (at slower bit rates, this occurs later). This error has a higher priority than "no unique 48bits".

- `BURST_RESULT_STATE__NO_UNIQUE_48BITS_FOUND`

  For reliable synchronization, a pattern must contain a unique 48-bit pattern (the detect word). If the current pattern does not have a detect word, this error occurs.

  If this status occurs, the synchronization may be incorrect, as could also the measured bit error rate. There are standard patterns that may contain more than one instance of the used detect word. Statistically, every other burst would be correctly synchronized.

  In this case, it is recommended that you redefine the pattern. This error can only occur with memory-based patterns.

- `BURST_RESULT_STATE__UNKNOWN`

  The status is unknown. This can occur if accumulation has not been started, or if Burst Sync mode has not been activated.

NOTE

Please note that the responses each have two underscores after STATE.

[P]FETCh[:SENSe[1]]:ECOunt Subnode

This subnode has the following SCPI structure:

```
[P]FETCh
└ [:SENSe[1]]
  └ :ECOunt
    ├ [:ALL]
    │ └ [:FULL]
    │   ├ [:TOTal]?
    │   └ :DELTa?
    ├ :OASZero
    │ └ [TOTal]?
    └ :ZASone
      └ [:TOTal]?
```

This subnode has the following commands:

Table 61

Name	Description under
:ECOunt[:ALL][:FULL][:TOTal]?	"[P]FETCh[:SENSe[1]]:ECOunt[:ALL][:FULL][:TOTal]? " on page 227
:ECOunt[:ALL][:FULL]:DELTa?	"[P]FETCh[:SENSe[1]]:ECOunt[:ALL][:FULL]:DELTa? " on page 227
:ECOunt:OASZero[:TOTal]?	"[P]FETCh[:SENSe[1]]:ECOunt:OASZero[:TOTal]? " on page 227
:ECOunt:ZASone[:TOTal]?	"[P]FETCh[:SENSe[1]]:ECOunt:ZASone[:TOTal]? " on page 228

[P]FETCh[:SENSe[1]]:ECOunt[:ALL][:FULL][:TOTal]?

IVI-COM Equivalent IAgilentN490xEDErrorCount.Read (IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:ECOunt[:ALL][:FULL][:TOTal]?

Description This query is a contraction of the phrase **Error COUnt**. It is the number of errors received in a time interval.

[P]FETCh[:SENSe[1]]:ECOunt[:ALL][:FULL]:DELTa?

IVI-COM Equivalent IAgilentN490xEDErrorCount.ReadDelta (IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:ECOunt[:ALL][:FULL]:DELTa?

Description The "instantaneous" number of errors, calculated from the counts obtained in the last deci-second. This value is available even when accumulation is turned off.

[P]FETCh[:SENSe[1]]:ECOunt:OASZero[:TOTal]?

IVI-COM Equivalent IAgilentN490xEDErrorCount.ReadOAZ (IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:ECOunt:OASZero[:TOTal]?

Description This is a contraction of the phrase **O**ne received **ASZ**ero. The command returns the number of errored ones (each true data one that is received as a data zero).

[P]FETCh[:SENSe[1]]:ECOunt:ZASone[:TOTal]?

IVI-COM Equivalent IAgilentN490xEDErrorCount.ReadZAO (IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:ECOunt:ZASone[:TOTal]?

Description This is a contraction of the phrase **Z**ero received **AS**one. The command returns the number of errored zeros (each true data zero that is received as a data one).

[P]FETCh[:SENSe[1]]:EFINterval Subnode

This subnode has the following SCPI structure:

```
[P]FETCh
└ [:SENSe[1]]
  └ :EFINterval
    ├ :SEConds?
    ├ :DSEConds?
    ├ :CSEConds?
    └ :MSEConds?
```

This subnode has the following commands:

Table 62

Name	Description under
:SEConds?	“[P]FETCh[:SENSe[1]]:EFINterval:SEConds? ” on page 229
:DSEConds?	“[P]FETCh[:SENSe[1]]:EFINterval:DSEConds? ” on page 229

Table 62

Name	Description under
:CSEConds?	"[P]FETCh[:SENSe [1]]:EFINterval:CSEConds? " on page 229
:MSEConds?	"[P]FETCh[:SENSe [1]]:EFINterval:MSEConds? " on page 230

[P]FETCh[:SENSe[1]]:EFINterval:SEConds?

IVI-COM Equivalent IAgilentN490xEDIntervals.ReadErrorFreeSeconds (IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:EFINterval:SEConds?

Description This query is a contraction of the phrase **Error-Free INterval** and returns a count of the number of seconds during which no error was detected.

[P]FETCh[:SENSe[1]]:EFINterval:DSEConds?

IVI-COM Equivalent IAgilentN490xEDIntervals.ReadErrorFreeDeciSeconds (not IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:EFINterval:DSEConds?

Description This query is a contraction of the phrase **Error-Free INterval** and returns a count of the number of deci-seconds during which no error was detected.

[P]FETCh[:SENSe[1]]:EFINterval:CSEConds?

IVI-COM Equivalent IAgilentN490xEDIntervals.ReadErrorFreeCentiSeconds (not IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:EFINterval:CSEConds?

Description This query is a contraction of the phrase **Error-Free INterval** and returns a count of the number of centiseconds during which no error was detected.

[P]FETCh[:SENSe[1]]:EFINterval:MSEConds?

IVI-COM Equivalent IAgilentN490xEDIntervals.ReadErrorFreeMilliSeconds (not IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:EFINterval:MSEConds?

Description This query is a contraction of the phrase **Error-Free INterval** and returns a count of the number of milliseconds during which no error was detected.

[P]FETCh[:SENSe[1]]:EINTerval Subnode

This subnode has the following SCPI structure:

```
[P]FETCh
└ [:SENSe[1]]
    └ :EINTerval
        ├ :SEConds?
        ├ :DSEConds?
        ├ :CSEConds?
        └ :MSEConds?
```

This subnode has the following commands:

Table 63

Name	Description under
:SEConds?	"[P]FETCh[:SENSe[1]]:EINTerval:SEConds? " on page 231
:DSEConds?	"[P]FETCh[:SENSe[1]]:EINTerval:DSEConds? " on page 231

Table 63

Name	Description under
:CSEConds?	"[P]FETCh[:SENSe [1]]:EINTerval:CSEConds? " on page 231
:MSEConds?	"[P]FETCh[:SENSe [1]]:EINTerval:MSEConds? " on page 232

[P]FETCh[:SENSe[1]]:EINTerval:SEConds?

IVI-COM Equivalent IAgilentN490xEDIntervals.ReadErroredSeconds (IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:EINTerval:SEConds?

Description This query is a contraction of the phrase **E**rrored **INT**erval and returns a count of the number of seconds during which one or more errors were detected.

[P]FETCh[:SENSe[1]]:EINTerval:DSEConds?

IVI-COM Equivalent IAgilentN490xEDIntervals.ReadErroredDeciSeconds (not IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:EINTerval:DSEConds?

Description This query is a contraction of the phrase **E**rrored **INT**erval and returns a count of the number of deci-seconds during which one or more errors were detected.

[P]FETCh[:SENSe[1]]:EINTerval:CSEConds?

IVI-COM Equivalent IAgilentN490xEDIntervals.ReadErroredCentiSeconds (not IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:EINTerval:CSEConds?

Description This query is a contraction of the phrase **E**rrored **INT**erval and returns a count of the number of centiseconds during which one or more errors were detected.

[P]FETCh[:SENSe[1]]:EINTerval:MSEConds?

IVI-COM Equivalent IAgilentN490xEDIntervals.ReadErroredMilliSeconds (not IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:EINTerval:MSEConds?

Description This query is a contraction of the phrase **E**rrored **INT**erval and returns a count of the number of milliseconds during which one or more errors were detected.

[P]FETCh[:SENSe[1]]:ERATio Subnode

This subnode has the following SCPI structure:

```
[P]FETCh
└ [:SENSe[1]]
   └ :ERATio
      ├ [:ALL][:FULL][:TOTal][?]
      ├ [:ALL][:FULL]:DELTa[?]
      ├ :OASZero[:TOTal][?]
      └ :ZASones[:TOTal][?]
```

This subnode has the following commands:

Table 64

Name	Description under
[:ALL][:FULL][:TOTal][?]	"[P]FETCh[:SENSe[1]]:ERATio[:ALL][:FULL][:TOTal]? " on page 233
[:ALL][:FULL]:DELTa[?]	"[P]FETCh[:SENSe[1]]:ERATio[:ALL][:FULL]:DELTa? " on page 233
:OASZero[:TOTal][?]	"[P]FETCh[:SENSe[1]]:ERATio:OASZero[:TOTal]? " on page 233

Table 64

Name	Description under
:ZASone[:TOTal][?]	“[P]FETCh[:SENSe[1]]:ERATio:ZASone [:TOTal]? ” on page 234

[P]FETCh[:SENSe[1]]:ERATio[:ALL][:FULL][:TOTal]?

IVI-COM Equivalent IAgilentN490xEDErrorRatio.Read (IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:ERATio[:ALL][:FULL][:TOTal]?

Description This query is a contraction of the phrase **Error RAT**io. It is the ratio of the number of errors to the number of bits received in the interval specified by gate period.

[P]FETCh[:SENSe[1]]:ERATio[:ALL][:FULL]:DELTa?

IVI-COM Equivalent IAgilentN490xEDErrorRatio.ReadDelta (IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:ERATio[:ALL][:FULL]:DELTa?

Description This query returns the "instantaneous" error ratio calculated from the counts obtained in the last deci-second. This value is available even when accumulation is turned off.

[P]FETCh[:SENSe[1]]:ERATio:OASZero[:TOTal]?

IVI-COM Equivalent IAgilentN490xEDErrorRatio.ReadOAZ (IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:ERATio:OASZero[:TOTal]?

Description This is a contraction of the phrase **O**ne received **AS Z**ero. The query returns the ratio of erred ones (a true data one received a data zero) to number of bits.

[P]FETCh[:SENSe[1]]:ERATio:ZASone[:TOTal]?

IVI-COM Equivalent IAgilentN490xEDErrorRatio.ReadZAO (IVI-compliant)

Syntax [P]FETCh[:SENSe[1]]:ERATio:ZASone[:TOTal]?

Description This is a contraction of the phrase **Z**ero received **AS O**ne. The query returns the ratio of erred zeros (a true data zero received a data one) to number of bits.

[P]FETCh[:SENSe[1]]:G821 Subnode

This subnode has the following SCPI structure:

```
[P]FETCh
└ [:SENSe[1]]
  └ :G821
    ├ :AVAilability?
    ├ :UNAVailability?
    ├ :SESeconds?
    ├ :DMINutes?
    └ :ESEConds?
```

This subnode has the following commands:

Table 65

Name	Description under
:AVAilability?	“[P]FETCh[:SENSe[1]]:G821:AVAilability? ” on page 235
:UNAVailability?	“[P]FETCh[:SENSe[1]]:G821:UNAVailability? ” on page 235
:SESeconds?	“[P]FETCh[:SENSe[1]]:G821:SESeconds? ” on page 235
:DMINutes?	“[P]FETCh[:SENSe[1]]:G821:DMINutes? ” on page 236

Table 65

Name	Description under
:ESEConds?	"[P]FETCh[:SENSe[1]]:G821:ESEConds? " on page 236

NOTE

The following commands return a percentage of seconds that have been classified according to the CCITT's G.821 specification.

[P]FETCh[:SENSe[1]]:G821:AVAilability?

IVI-COM Equivalent IAgilentN490xEDG821ReadPercentAvailability

Syntax [P]FETCh[:SENSe[1]]:G821:AVAilability?

Description Returns the G.821 *availability*.

[P]FETCh[:SENSe[1]]:G821:UNAVailability?

IVI-COM Equivalent IAgilentN490xEDG821ReadPercentUnavailability

Syntax [P]FETCh[:SENSe[1]]:G821:UNAVailability?

Description Returns the G.821 *unavailability*.

[P]FETCh[:SENSe[1]]:G821:SESeconds?

IVI-COM Equivalent IAgilentN490xEDG821ReadPercentSeverelyErroredSeconds

Syntax [P]FETCh[:SENSe[1]]:G821:SESeconds?

Description Returns the G.821 *severely errored seconds*.

[P]FETCh[:SENSe[1]]:G821:DMINutes?

IVI-COM Equivalent	IAgilentN490xEDG821ReadDegradedMinutes
Syntax	[P]FETCh[:SENSe[1]]:G821:DMINutes?
Description	Returns the G.821 *degraded minutes*.

[P]FETCh[:SENSe[1]]:G821:ESEConds?

IVI-COM Equivalent	IAgilentN490xEDG821ReadPercentErroredSeconds
Syntax	[P]FETCh[:SENSe[1]]:G821:ESEConds?
Description	Returns the G.821 *errored seconds*.

PLUGin Subsystem

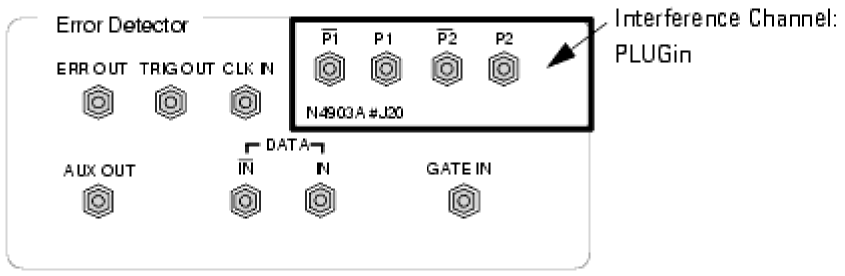
PLUGin Subsystem - Reference

The PLUGin subsystem controls the optional Interference Channel #J20, a hardware plug-in module.

The Interference Channel #J20 can be used to add sinusoidal interference to the generated data and to simulate intersymbol interference.

Sinusoidal interference adds a common-mode, differential, or single-ended sinewave signal to the generated data. It is used to test common mode rejection of a receiver and to emulate vertical eye closure.

Intersymbol interference of various degree is generated by selecting one of nine on-board traces.



This subsystem has the following SCPI structure:

```
:PLUGin[:J20]
 ├ :Global[:STATe]
 ├ :SINTerference
 │  ├ [:STATe][?]
 │  ├ :VOLTage[?]
 │  ├ :LEVel[:IMMediate][:AMPLitude][?]
 │  ├ :FREQency[?]
 │  └ :MODE[?]
 └ :ISINterference[:SELect]
    └ :TRACe[?]
```

This subsystem has the following commands:

Table 66

Name	Description under
Commands	
:GLOBal[:STATE][?]	“PLUGin[:J20]:GLOBal[:STATE][?] ” on page 238
:SINTerference[:STATe][?]	“PLUGin[:J20]:SINTerference[:STATe][?] ” on page 238
:SINTerference:VOLTage[?]	“PLUGin[:J20]:SINTerference:VOLTage[?] ” on page 238
:SINTerference:FREQuency[?]	“PLUGin[:J20]:SINTerference:FREQuency[?] ” on page 239

Table 66

Name	Description under
:SINTerference:MODE[?]	"PLUGin[:J20]:SINTerference:MODE[?] " on page 239
:ISINterference:TRACe[?]	"PLUGin[:J20]:ISINterference:TRACe[?] " on page 239

PLUGin[:J20]:GLOBal[:STATE][?]

IVI-COM Equivalent IAgilentN490xJInterferenceEnable

Syntax PLUGin[:J20]:GLOBal[:STATE] ON | OFF | 1 | 0

PLUGin:GLOB?

Description Enables or disables the plug-in module.

When the module is enabled, the previous (present) setup is restored.

The query returns the present setting (0 | 1).

PLUGin[:J20]:SINTerference[:STATe][?]

IVI-COM Equivalent IAgilentN490xJSinusoidalInterferenceEnable

Syntax PLUGin[:J20]:SINTerference[:STATE] ON | OFF | 1 | 0

PLUGin[:J20]:SINT?

Description Turns the generation of sinusoidal interference on or off.

The query returns the present setting (0 | 1).

PLUGin[:J20]:SINTerference:VOLTage[?]

IVI-COM Equivalent IAgilentN490xJSinusoidalInterferenceAmplitude

Syntax PLUGin[:J20]:SINTerference:VOLTage[:LEVel][:IMMediate][:AMPLitude] <NR3>

PLUGin[:J20]:SINT:VOLTage? MIN | MAX

Description Sets the sinusoidal interference voltage amplitude in V.

The query returns the present setting or the applicable min/max values.

PLUGin[:J20]:SINTerference:FREQuency[?]

IVI-COM Equivalent IAgilentN490xJSinusoidalInterferenceModulationFrequency

Syntax PLUGin[:J20]:SINTerference:FREQuency <NR3>

PLUGin[:J20]:SINT:FREQ? MIN | MAX

Description Sets the frequency of the sinusoidal interference in Hz.

The query returns the present setting or the applicable min/max values.

PLUGin[:J20]:SINTerference:MODE[?]

IVI-COM Equivalent IAgilentN490xJSinusoidalInterferenceMode

Syntax PLUGin[:J20]:SINTerference:MODE DIFFerential | SNORmal | SCOMplement | COMMon

PLUGin[:J20]:SINT:MODE?

Description Sets the sinusoidal interference mode.

The query returns the present setting (DIFF, SNOR, SCOM, COMM).

PLUGin[:J20]:ISINterference:TRACe[?]

IVI-COM Equivalent IAgilentN490xJISIPathLength

Syntax PLUGin[:J20]:ISINterference[:SELect]:TRACe <NR1>

PLUGin[:J20]:ISIN:TRAC? MIN | MAX

Description Selects a trace for the intersymbol interference mode (1 to 9).

The query returns the present setting or the applicable min/max values.

STATus Subsystem

STATus Subsystem - Reference

The STATus Subsystem provides an interface to the instrument's Status Register. For information on how to work with the Status register, see "Reading the Serial BERT's Status - Reference" on page 28.

This subsystem has the following SCPI structure:

```
STATus
 ├─ :CLOSs
 │   └ ...
 ├─ :OPERation
 │   └ ...
 ├─ :PRESet
 └─ :QUEStionable
```

This subsystem has the following commands and subnodes:

Table 67

Name	Description under
Commands	
STATus:PRESet	"STATus:PRESet " on page 241
Subnodes	

Table 67

Name	Description under
STATus:CLOSs	"CLOSs Subnode" on page 242
STATus:OPERation	"STATus:OPERation Subnode" on page 244
STATus:QUEStionable	"STATus:QUEStionable Subnode" on page 247

STATus:PRESet

IVI-COM Equivalent IAgilentN490xStatus.Preset (not IVI-compliant)

Syntax STATus:PRESet

Description The PRESet command is an event that configures the SCPI and device-dependent status data structures, such that the device-dependent events are reported at a higher level through the mandatory part of the status reporting structures.

The PRESet command affects only the enable register and the transition filter registers for the SCPI mandated and device dependent status data structures. PRESet does not affect either the "status byte" or the "standard event status" as defined by IEEE 488.2. PRESet does not clear any of the event registers. The *CLS command is used to clear all event registers in the device status reporting mechanism.

From the device-dependent status data structures, the PRESet command sets the enable register to all one's and the transition filter to recognize both positive and negative transitions. For the SCPI mandatory status data structures, the PRESet command sets the transition filter registers to recognize only positive transitions and sets the enable register to zero.

CLOSs Subnode

This subnode refers to the clock loss status register. It has the following SCPI structure:

```
STATus
└ :CLOSs
   ├ :CONDition?
   ├ :ENABle[?]
   ├ [:EVENt]?
   ├ :NTRansition[?]
   └ :PTRansition[?]
```

This subnode has the following commands:

Table 68

Name	Description under
:CONDition	“STATus:CLOSs:CONDition ” on page 242
:ENABle[?]	“STATus:CLOSs:ENABle[?] ” on page 243
[:EVENt]?	“STATus:CLOSs[:EVENt]? ” on page 243
:NTRansition[?]	“STATus:CLOSs:NTRansition[?] ” on page 243
:PTRansition[?]	“STATus:CLOSs:PTRansition[?] ” on page 244

STATus:CLOSs:CONDition

IVI-COM Equivalent: IAgilentN490xStatus.Register (not IVI-compliant)

Syntax: STATus:CLOSs:CONDition?

Description: This query returns the contents of the condition register in the Clock Loss Status Register. See “Clock Loss Register” on page 31 for the layout of the Clock Loss register.

STATus:CLOSs:ENABle[?]

IVI-COM Equivalent IAgilentN490xStatus.Register (not IVI-compliant)

Syntax STATus:CLOSs:ENABle <Num.>

STATus:CLOSs:ENABle?

Description The command sets the enable mask in the Clock Loss Register, which allows true conditions in the event register to be reported in the summary bit. The query returns the weighted value of the bits that are set in the enable register. See "Clock Loss Register" on page 31 for the layout of the Clock Loss register.

STATus:CLOSs[:EVENt]?

IVI-COM Equivalent IAgilentN490xStatus.Register (not IVI-compliant)

Syntax STATus:CLOSs[:EVENt]?

Description The bits in this register indicate pattern generator and error detector clock loss. This query returns whether the pattern generator or error detector has experienced the clock loss. See "Clock Loss Register" on page 31 for the layout of the Clock Loss register.

STATus:CLOSs:NTRansition[?]

IVI-COM Equivalent IAgilentN490xStatus.Register (not IVI-compliant)

Syntax STATus:CLOSs:NTRansition <Num.>

STATus:CLOSs:NTRansition?

Description This command sets the negative transition register state in the Clock Loss Register. When a bit in this mask is set to "1", negative (logic 1 changing to logic 0) transitions of this bit are allowed to pass. The query returns the weighted value of the bits that are set to pass negative transitions in the transition filter. See "Clock Loss Register" on page 31 for the layout of the Clock Loss register.

STATus:CLOSs:PTRansition[?]

IVI-COM Equivalent IAgilentN490xStatus.Register (not IVI-compliant)

Syntax STATus:CLOSs:PTRansition <Num.>

STATus:CLOSs:PTRansition?

Description This command sets the positive transition register state in the Clock Loss Register. When a bit in this mask is set to "1", positive transitions (logic 0 changing to logic 1) of this bit are allowed to pass. This is the default setting of the instrument. The query returns the weighted value of the bits that are set to pass positive transitions in the transition filter. See "Clock Loss Register" on page 31 for the layout of the Clock Loss register.

STATus:OPERation Subnode

This subnode has the following SCPI structure:

```
STATus
└ :OPERation
   ├ :CONDition?
   ├ :ENABle[?]
   ├ [:EVENt]?
   ├ :NTRansition[?]
   └ :PTRansition[?]
```

This subnode has the following commands:

Table 69

Name	Description under
:CONDition?	"STATus:OPERation:CONDition? " on page 245
:ENABle[?]	"STATus:OPERation:ENABle[?] " on page 245
[:EVENt]?	"STATus:OPERation[:EVENt]? " on page 245

Table 69

Name	Description under
:NTRansition[?]	“STATus:OPERation:NTRansition[?]” on page 246
:PTRansition[?]	“STATus:OPERation:PTRansition[?]” on page 246

STATus:OPERation:CONDition?

IVI-COM Equivalent IAgilentN490xStatus.Register (not IVI-compliant)

Syntax STATus:OPERation:CONDition?

Description This query only returns the contents of the condition register in the Operation Status Register. See “Operation Status Register” on page 34 for the layout of the Operation Status register.

STATus:OPERation:ENABle[?]

IVI-COM Equivalent IAgilentN490xStatus.Register (not IVI-compliant)

Syntax STATus:OPERation:ENABle

STATus:OPERation:ENABle?

Description The command sets the enable mask in the Operation Status Register, which allows true conditions in the event register to be reported in the summary bit. The query returns the weighted value of the bits that are set in the enable register. See “Operation Status Register” on page 34 for the layout of the Operation Status register.

STATus:OPERation[:EVENt]?

IVI-COM Equivalent IAgilentN490xStatus.Register (not IVI-compliant)

Syntax STATus:OPERation[:EVENt]?

Description

This query returns the contents of the Operation Status event register. See “Operation Status Register ” on page 34 for the layout of the Operation Status register.

Note that reading the event register clears it.

STATus:OPERation:NTRansition[?]

IVI-COM Equivalent

IAgilentN490xStatus.Register (not IVI-compliant)

Syntax

STATus:OPERation:NTRansition

STATus:OPERation:NTRansition?

Description

This command sets the transition filter state in the Operation Status Register. When this mask is set to "1", negative (logic 1 changing to logic 0) transitions are allowed to pass. The query returns the weighted value of the bits that are set to pass negative transitions in the transition filter. See “Operation Status Register ” on page 34 for the layout of the Operation Status register.

STATus:OPERation:PTRansition[?]

IVI-COM Equivalent

IAgilentN490xStatus.Register (not IVI-compliant)

Syntax

STATus:OPERation:PTRansition

STATus:OPERation:PTRansition?

Description

This command sets the transition filter state in the Operation Status Register. When this mask is set to "1", positive transitions (logic 0 changing to logic 1) are allowed to pass. This is the default setting of the instrument. The query returns the weighted value of the bits that are set to pass positive transitions in the transition filter. See “Operation Status Register ” on page 34 for the layout of the Operation Status register.

STATus:QUEStionable Subnode

This subnode has the following SCPI structure:

```
STATus
└ :QUEStionable
   ├ :CONDition?
   ├ :ENABle[?]
   ├ [:EVENt]?
   ├ :NTRansition[?]
   └ :PTRansition[?]
```

This subnode has the following commands:

Table 70

Name	Description under
:CONDition?	"STATus:QUEStionable:CONDition? " on page 247
:ENABle[?]	"STATus:QUEStionable:ENABle[?] " on page 248
[:EVENt]?	"STATus:QUEStionable[:EVENt]? " on page 248
:NTRansition[?]	"STATus:QUEStionable:NTRansition[?] " on page 248
:PTRansition[?]	"STATus:QUEStionable:PTRansition[?] " on page 249

STATus:QUEStionable:CONDition?

IVI-COM Equivalent IAgilentN490xStatus.Register (not IVI-compliant)

Syntax STATus:QUEStionable:CONDition?

Description This query returns the contents of the condition register in the Questionable Status Register. See "Questionable Status Register " on page 32 for the layout of the Questionable Status register.

STATus:QUEStionable:ENABle[?]

IVI-COM Equivalent IAgilentN490xStatus.Register (not IVI-compliant)

Syntax STATus:QUEStionable:ENABle

STATus:QUEStionable:ENABle?

Description The command form sets the enable mask in the Questionable Status Register, which allows true conditions in the event register to be reported in the summary bit. The query form returns the weighted value of the bits that are set in the enable register. See "Questionable Status Register " on page 32 for the layout of the Questionable Status register.

STATus:QUEStionable[:EVENt]?

IVI-COM Equivalent IAgilentN490xStatus.Register (not IVI-compliant)

Syntax STATus:QUEStionable[:EVENt]

STATus:QUEStionable:ENABle?

Description This query form returns the contents of the Questionable Status event register. See "Questionable Status Register " on page 32 for the layout of the Questionable Status register.

STATus:QUEStionable:NTRansition[?]

IVI-COM Equivalent IAgilentN490xStatus.Register (not IVI-compliant)

Syntax STATus:QUEStionable:NTRansition

STATus:QUEStionable:NTRansition?

Description This command sets the transition filter state in the Questionable Status Register. When this mask is set to "1", negative (logic 1 changing to logic 0) transitions are allowed to pass. The query form returns the weighted value of the bits that are set to pass negative transitions in the transition filter. See "Questionable Status Register " on page 32 for the layout of the Questionable Status register.

STATus:QUEStionable:PTRansition[?]

IVI-COM Equivalent IAgilentN490xStatus.Register (not IVI-compliant)

Syntax STATus:QUEStionable:PTRansition

STATus:QUEStionable:PTRansition?

Description This command sets the transition filter state in the Questionable Status Register. When this mask is set to "1", positive transitions (logic 0 changing to logic 1) are allowed to pass. This is the default setting of the instrument. The query returns the weighted value of the bits that are set to pass positive transitions in the transition filter. See "Questionable Status Register " on page 32 for the layout of the Questionable Status register.

SYSTem Subsystem

SYSTem Subsystem - Reference

The SYSTem subsystem represents general system functions.

The subsystem has the following SCPI structure:

```
SYSTem
├─ :BEEPer
│   ├─ :MODE[?]
│   ├─ :STATe
│   ├─ :THReshold[?]
│   └─ :VOLume[?]
├─ :ERRor
│   └─ [:NEXT]?
├─ :GPIB[?]
├─ :HELP
│   └─ :HEADers?
├─ :LICense
│   ├─ :AVAilable?
│   ├─ :DEPendency?
│   ├─ :HOST?
│   ├─ :INFO?
│   ├─ :INSTall
│   └─ :SERial?
└─ :VERSion?
```

This subsystem has the following commands:

Table 71

Name	Description under
:BEEPer:MODE[?]	“SYSTem:BEEPer:MODE[?] ” on page 251
:BEEPer:STATe	“SYSTem:BEEPer:STATe[?] ” on page 251
:BEEPer:THReshold[?]	“SYSTem:BEEPer:THReshold[?] ” on page 252
:BEEPer:VOLume[?]	“SYSTem:BEEPer:VOLume[?] ” on page 252
:ERRor[:NEXT]?	“SYSTem:ERRor[:NEXT]? ” on page 252
:GPIB[?]	“SYSTem:GPIB[?]” on page 253
:HELP:HEADers?	

Table 71

Name	Description under
:LICense:AVAilable?	"SYSTem:LICense:AVAilable? " on page 253
:LICense:DEPendency?	"SYSTem:LICense:DEPendency? " on page 253
:LICense:HOST?	"SYSTem:LICense:HOST? " on page 254
:LICense:INFO?	"SYSTem:LICense:INFO? " on page 254
:LICense:INSTall	"SYSTem:LICense:INSTall " on page 254
:VERSion?	"SYSTem:VERSion? " on page 255

The LICense commands refer to the specific N4903 licensed options.

SYSTem:BEEPer:MODE[?]

IVI-COM Equivalent IAgilentN490xAudio.Mode (not IVI-compliant)

Syntax SYSTem:BEEPer:MODE BERalarm | TONes

SYSTem:BEEPer:MODE?

Description The command form sets the instrument's audible beeper to trigger on either a specific BER level (BERalarm) or on any occurrence of errors (TONes).

The response returns the current mode setting of the instrument's audible beeper.

SYSTem:BEEPer:STATe[?]

IVI-COM Equivalent IAgilentN490xAudio.Enabled (not IVI-compliant)

Syntax SYSTem:BEEPer:STATe 0 | 1 | OFF | ON

SYSTem:BEEPer:STATe?

Description The command turns on and off the instrument's audible beeper. The response returns whether or not the instrument's audible beeper is turned on.

SYSTem:BEEPer:THReshold[?]

IVI-COM Equivalent IAgilentN490xAudio.Threshold (not IVI-compliant)

Syntax SYSTem:BEEPer:THReshold <Numeric value>

SYSTem:BEEPer:THREshold?

Description The command sets the BER threshold value at which the instrument's audible beeper will produce sounds.

The response returns the current setting of the BER threshold at which the instrument's audible beeper will produce sounds.

SYSTem:BEEPer:VOLume[?]

IVI-COM Equivalent IAgilentN490xAudio.Volume (not IVI-compliant)

Syntax SYSTem:BEEPer:VOLume <Numeric value>

SYSTem:BEEPer:VOLume?

Description The command controls the volume of the instrument's audible beeper. The response returns the current volume of the instrument's audible beeper.

SYSTem:ERRor[:NEXT]?

IVI-COM Equivalent IIviDriverOperation.GetNextInterchangeWarning (IVI-compliant)

Syntax SYSTem:ERRor[:NEXT]?

Description This query pulls the next error from the error queue, and returns the error number and a string describing the error. The error queue has a depth of 20.

SYSTem:GPIB[?]

IVI-COM Equivalent IAgilentN490xUtilities.GPIBAddress (not IVI-compliant)

Syntax SYSTem:GPIB <Numeric value>

SYSTem:GPIB?

Description Sets or returns the instrument's GPIB address.

SYSTem:HELP:HEADers?

Syntax SYSTem:HELP:HEADers?

Description This query returns the complete list of instrument commands. Not all of the commands are implemented, however. For more information, refer to the specific command groups in this guide.

SYSTem:LICense:AVAilable?

IVI-COM Equivalent IAgilentN490xLicenseGetAvailableLicenses

Syntax LICense:AVAilable?

Description This query returns a quoted string, a comma-separated list of available licenses, for example "A01, J12". There is no differentiation between SW or HW options or if the option is already installed or not.

The query is similar to *OPT? but less verbose.

SYSTem:LICense:DEPendency?

IVI-COM Equivalent IAgilentN490xLicenseGetDependencies

Syntax LICense:DEPendency? <"OptionNumber">

Description This query returns a quoted string, a comma-separated list of license identifiers required before the specified option/license can be installed, e.g. "J10, A01, N4903A".

SYSTem:LICense:HOST?

IVI-COM Equivalent IAgilentN490xLicenseGetInstrumentID

Syntax LICense:HOST?

Description Licensed options are bound to a certain host. This query returns the host ID as a quoted string, either MAC address or CPU ID.

SYSTem:LICense:INFO?

IVI-COM Equivalent IAgilentN490xLicenseOptionInfo

Syntax LICense:INFO? <"OptionNumber">

Description This query returns a quoted string, separated by commas, that describes the specified option, for example: "J12, Jitter Compliance Suite, 1.0, Yes, SW"

Explanation:

- OptionNumber: e.g. J12
- OptionDescription: e.g. Jitter Compliance Suite
- Version Number: e.g. 1.0
- Installation Status: Yes or No
- Option is hardware or software related: HW, SW

SYSTem:LICense:INSTall

This command is used to activate a licensed option on an instrument.

To activate a licensed option, you need a license file that contains the license code. This file needs to be installed.

The license file is readable. You can reference the license file or submit its contents. Therefore, this command has three options.

IVI-COM Equivalent	IAgilentN490xLicenseInstallFile
Syntax for accessing the file	LICense:INSTall:FILE <"FeatureName">, <"LicenseFilePath">
Description	This command requires that the license file is accessible (e.g. has been copied to the instrument). <"FeatureName"> is the name of the license, given as a quoted string (like "Bit Recovery Mode" or "SSC Generation". <"LicenseFilePath"> specifies the directory where the license file is stored. The command opens the license file, reads its contents, and installs (stores) the license.
IVI-COM Equivalent	IAgilentN490xLicenseInstallKey
Syntax for submitting the key only	LICense:INSTall:CODE <"FeatureName">, <"60DigitLicenseKey">
Description	This command requires that you provide the FeatureName and the LicenseKey contained in the license file in quoted strings. It installs (stores) the license.
Syntax for submitting the file contents	LICense:INSTall:CODE <"FeatureName">, <"Contents of License File">
Description	This command requires that you supply the FeatureName and the complete contents of the license file as two quoted strings. It installs (stores) the license. OptionNumber and LicenseKey are automatically separated.

SYSTem:VERSion?

Syntax	SYSTem:VERSion?
Description	This query returns the version of the SCPI programming language, which supports the GPIB commands.

MEASure Subsystem

MEASure Subsystem - Reference

The MEASure Subsystem represents the instrument's advanced analysis features that can be controlled over SCPI.

NOTE

Note that for these commands, the asterisks denote the "handle" of a measurement. To obtain the handle, the measurement object must first be created.

The MEASure Subsystem is first of all meant for programming the Fast Eye Mask measurement, the Jitter Tolerance Characterization measurement, and the Eye Diagram measurement.

For an example on how to program the Fast Eye Mask measurement with SCPI, see "Running the Fast Eye Mask - Procedures" on page 37.

For an example on how to program the Jitter Tolerance Characterization measurement with SCPI, see "Running JTol Characterization - Procedures" on page 42.

For an example on how to program the Eye Diagram measurement with SCPI, see "Running the Eye Diagram - Procedures" on page 40.

The subsystem has the following SCPI structure:

```
MEASure
├─ :SYSTem:ERRor?
├─ :GENeric(*)
│  └ ...
├─ :FEMask(*)
│  └ ...
├─ :JTOL
│  ├─ :CHAR(*)
│  │  └ ...
│  └─ :COMP(*)
│     └ ...
└─ :EMask(*)
   └ ...
```

This subsystem has the following commands and subnodes:

Table 72

Name	Description under
Commands	
:SYSTem:ERRor?	"MEAS:SYSTem:ERRor?" on page 258
Subnodes	
:GENeric	"MEASure:GENeric(*) Subnode" on page 258
:FEMask	"MEASure:FEMask(*) Subnode" on page 262
:JTOLerance:CHARacterization	"MEASure:JTOL:CHAR(*) Subnode" on page 269
:JTOLerance:COMPliance	"MEASure:JTOL:COMP(*) Subnode" on page 281
:EMASk	"MEASure:EMASk(*) Subnode" on page 292

MEAS:SYSTem:ERRor?

IVI-COM Equivalent IAgilentN490xEDDataIn.GetMeasurementError()

Syntax MEASure:SYSTem:ERR?

Description This query pulls the next error from the error queue. It returns the error and removes the error from the queue.

MEASure:GENeric(*) Subnode

This subnode has the following SCPI structure:

```
MEASure
└ :GENeric(*)
   ├ :ABORt
   ├ :CLOSe
   ├ :GO
   ├ :OPC?
   ├ :PROGress?
   ├ :PASSed?
   ├ :STATe?
   └ :DATA
      └ :AVAilable?
```

This subnode has the following commands:

Table 73

Name	Description under
:ABORt	“MEAS:GENeric(*):ABORt” on page 259
:CLOSe	“MEAS:GENeric(*):CLOSe” on page 259
:GO	“MEAS:GENeric(*):GO” on page 259
:OPC?	“MEAS:GENeric(*):OPC?” on page 260
:PROGress?	“MEAS:GENeric(*):PROGress?” on page 260

Table 73

Name	Description under
:PASSed?	"MEAS:GENeric(*):PASSed?" on page 260
:STATe?	"MEAS:GENeric(*):STATe?" on page 260
:DATA:AVAilable?	"MEAS:GENeric(*):DATA:AVAilable?" on page 261

MEAS:GENeric(*):ABORt

IVI-COM Equivalent IAgilentN490xEDFastEye.Abort()

Syntax MEAS:GENeric(*):ABORt

Description Aborts a running measurement. All parameters stored in the measurement object itself are still valid.

MEAS:GENeric(*):CLOSe

IVI-COM Equivalent IAgilentN490xEDFastEye.Close()

Syntax MEAS:GENeric(*):CLOSe

Description The measurement object is closed. The measurement object is removed from the internal measurement manager, the handle becomes invalid.

MEAS:GENeric(*):GO

IVI-COM Equivalent IAgilentN490xEDFastEye.Go()

Syntax MEAS:GENeric(*):GO

Description The measurement is started with the previously defined parameters.

MEAS:GENeric(*):OPC?

IVI-COM Equivalent IAgilentN490xEDFastEye.GetRunState()

Syntax MEAS:GENeric(*):OPC? [BLOCk | NBLock]

Description This query returns 1 when all pending measurement operations have been finished.

In NBLock (No blocking) mode, it returns "0" if the measurement is not finished. In BLOCk mode (no argument or with BLOCk as argument), the call waits until the measurement has finished and then returns "1".

MEAS:GENeric(*):PROGress?

IVI-COM Equivalent IAgilentN490xEDFastEye.Progress()

Syntax MEAS:GENeric(*):PROGress?

Description This query returns the progress of the measurement from 0.0 (not started) to 1.0 (completed).

MEAS:GENeric(*):PASSed?

IVI-COM Equivalent IAgilentN490xEDFastEye.Passed()

Syntax MEAS:GENeric(*):PASSed?

Description This query returns whether or not the measurement passed (0 = failed, 1 = passed).

MEAS:GENeric(*):STATe?

IVI-COM Equivalent IAgilentN490xEDFastEye.State()

Syntax MEAS:GENeric(*):STATe?

Description

Returns the measurement state:

- PROGram: Measurement not yet started, parameters can be changed.
- RUNNing: Measurement running, no parameters can be changed.
- FINished: Measurement completed. Similar to PROGram state, results can be queried.
- ABORted: Measurement was aborted.
- ERRor: The measurement has stopped due to an error. Use the query MEAS:SYSTem:ERRor? to get a description of the error.

MEAS:GENeric(*):DATA:AVAilable?

IVI-COM Equivalent

IAgilentN490xEDFastEye.GetDataCount()

Syntax

MEAS:GENeric(*):DATA:AVAilable?

Description

This query returns the number of measured points in integer format (NR1). The number is continually updated until the measurement has finished.

MEASure:FEMask(*) Subnode

This subnode has the following SCPI structure:

```
MEASure
└ :FEMask(*)
   ├ :CREate?
   └ :PARameter
      ├ :MCBits[?]
      ├ :MERRor
      │  ├ [:VALue][?]
      │  └ :MODE[?]
      ├ :PFCRiteria
      │  └ [:VALue][?]
      ├ :TRESolution
      │  └ :TYPE[?]
      ├ :THReshold
      │  └ [:INPut]
      │     └ :TYPE[?]
      └ :POINt
         ├ [:VALue][?]
         └ :NUMber[?]
```

This subnode has the following commands:

Table 74

Name	Description under
:CREate?	“MEAS:FEMask:CREate?” on page 263
:PARameter:MCBits[?]	“MEAS:FEMask(*):PARameter:MCBits [?]” on page 264
:PARameter:MERR[:VALue][?]	“MEAS:FEMask(*):PARameter:MERRor [:VALue][?]” on page 264
:PARameter:MERR:MODE[?]	“MEAS:FEMask (*):PARameter:MERRor:MODE[?]” on page 265
:PARameter:PFCRiteria[:VALue][?]	“MEAS:FEMask (*):PARameter:PFCRiteria[:VALue][?]” on page 265

Table 74

Name	Description under
:PARameter:TRESolution:TYPE[?]	"MEAS:FEMask (*):PARameter:TRESolution:TYPE[?]" on page 265
:PARameter:THReshold[:INPut]:TYPE[?]	"MEAS:FEMask (*):PARameter:THReshold[:INPut]:TYPE [?]" on page 265
:PARameter:POINt[:VALue][?]	"MEAS:FEMask(*):PARameter:POINt [:VALue][?]" on page 266
:PARameter:POINt:NUMBer[?]	"MEAS:FEMask (*):PARameter:POINt:NUMBer[?]" on page 267
:FETC:DATA[:VALUE]?	"MEAS:FEMask(*):FETC:DATA [:VALUE]? " on page 267
:FETC:DATA:RELative?	"MEAS:FEMask (*):FETC:DATA:RELative?" on page 268
:POINt:PASSed?	"MEAS:FEMask(*):POINt:PASSed? " on page 268

MEAS:FEMask:CREate?

IVI-COM Equivalent IAgilentN490xED2.CreateMeasurement()

Syntax MEAS:FEMask:CREate?

Description Creates a Fast Eye Mask measurement object. The return value is the handle used to identify this object by any calls to this object. If, for example, 2 is returned, typical calls to this object would be:

- :MEAS:GEN2:GO
- :MEAS:FEM2:PASSed?

MEAS:FEMask(*):PARameter:MCBits[?]

IVI-COM Equivalent IAgilentN490xEDFastEye.ComparedBits

Syntax MEAS:FEMask(*):PARameter:MCBits <num>

MEAS:FEMask(*):PARameter:MCBits?

Description After this number of compared bits, the measurement stops for the current sample point and moves to the next one.

The default is 1 million bits. That means, you can measure a bit error rate as low as 10 ^-6 (one error per million).

A smaller number reduces the duration of the whole Fast Eye Mask measurement. A larger number increases the precision of the measured bit error rates.

Note that the measurement moves to the next point when either the number of errors or the number of measured bits is reached.

The query returns the setting, not the number of bits actually measured.

MEAS:FEMask(*):PARameter:MERRor[:VALue][?]

IVI-COM Equivalent IAgilentN490xEDFastEye.ComparedErrors

Syntax MEAS:FEMask(*):PARameter:MERRor[:VALue]<num>

MEAS:FEMask(*):PARameter:MERRor[:VALue]?

Description After this number of errors, the measurement stops for the current sample point and moves to the next one. This allows you to speed up the measurement. You can switch off this option if only the number of compared bits is important.

This parameter is only evaluated when error mode is enabled (see following command).

Note that the measurement moves to the next point when either the number of errors or the number of measured bits is reached.

The query returns the setting, not the number of bits actually measured.

MEAS:FEMask(*):PARameter:MERRor:MODE[?]

IVI-COM Equivalent	IAgilentN490xEDFastEye.CheckComparedErrors
Syntax	MEAS:FEMask(*):PARameter:MERRor:MODE DISable | ENABle MEAS:FEMask(*):PARameter:MERRor:MODE?
Description	Enables/disables the evaluation of errored bits as criteria for moving to the next measurement point. The query returns the current state.

MEAS:FEMask(*):PARameter:PFCRiteria[:VALue][?]

IVI-COM Equivalent	IAgilentN490xEDFastEyeBERThreshold
Syntax	MEAS:FEMask(*):PARameter:PFCRiteria[:VALue] <num> MEAS:FEMask(*):PARameter:PFCRiteria[:VALue]?
Description	Specifies the maximum allowed bit error rate for determining whether a single measurement point has passed or failed. The query returns the current setting.

MEAS:FEMask(*):PARameter:TRESolution:TYPE[?]

IVI-COM Equivalent	IAgilentN490xEDFastEye.TimingType
Syntax	MEAS:FEMask(*):PARameter:TRESolution:TYPE TIME | UINTerval MEAS:FEMask(*):PARameter:TRESolution:TYPE?
Description	Specifies whether the time parameters are to be specified/returned in seconds (TIME) or unit intervals (UINT). For details on the unit intervals, see the online Help.

MEAS:FEMask(*):PARameter:THReshold[:INPut]:TYPE[?]

IVI-COM Equivalent	IAgilentN490xEDFastEye.ThresholdType

Syntax
MEAS:FEMask(*):PARameter:THReshold[:INPut]:TYPE ABSolute | PCT | OFFSet

MEAS:FEMask(*):PARameter:THReshold[:INPut]:TYPE?

Description
Determines how the threshold for the data point is reported:

- ABSolute: Absolute value for the threshold
- OFFSet: Threshold is reported relative to the analyzer threshold.
- PCT: Threshold is reported in percentage:
  - 0 % = Low level of the analyzer threshold.
  - 100 % = High level of the analyzer threshold

The threshold is returned with the following commands:

- "MEAS:FEMask(*):FETC:DATA[:VALUE]? " on page 267
- "MEAS:FEMask(*):FETC:DATA:RELative?" on page 268

MEAS:FEMask(*):PARameter:POINt[:VALue][?]

IVI-COM Equivalent
IAgilentN490xEDFastEye.SetMaskPoint(), GetMaskPoint()

Syntax
MEAS:FEMask(*):PARameter:POINt[:VALue] nIndex, dDelay, dThreshold

MEAS:FEMask(*):PARameter:POINt[:VALue]? nIndex

Description
Specifies the threshold and timing value for the specified point. The query returns the current values. The following parameters are used:

- nIndex

  Index of the data point to be specified/queried, 1 - 32. The number of points is specified/returned with :NUMBer.
- dDelay

  Delay (data type: NRf) for the data point reported according to the TRESolution.
- dThreshold

  Threshold (data type: Double) for the data point reported according to the :TYPE.

MEAS:FEMask(*):PARameter:POINt:NUMBer[?]

IVI-COM Equivalent	IAgilentN490xEDFastEye.MaskPointCount
Syntax	MEAS:FEMask(*):PARameter:POINt:NUMBer MEAS:FEMask(*):PARameter:POINt:NUMBer?
Description	Specifies/returns the number of data points to be measured. The maximum number of data points is 32.

MEAS:FEMask(*):FETC:DATA[:VALUE]?

IVI-COM Equivalent IAgilentN490xEDFastEye.GetResults()

Syntax MEAS:FEMask(*):FETC:DATA[:VALUE]? nItems

Description Returns a comma-separated list of measured values for nItems data points. If fewer points are available, only the values for available points are returned.

The list has the following parameters:

- First value returned is number of data points actually returned.
- Second value is a flag that indicates whether still more points are available.
- For each returned measurement point, 7 values are returned:
  - dDelay: Delay (data type: NR3).
  - dThreshold: Value for the bit error rate threshold (data type: NR3).
  - dComparedBits: Number of compared bits (data type: NR3).
  - dErroneousBits: Number of errors (data type: NR3).
  - dErroneousZeros: Number of errors from zeros (data type: NR3).
  - dErroneousOnes: Number of errors from ones (data type: NR3).
  - bExtrapolatedFlag: Flag 0 or 1 indicating if the data was calculated or extrapolated by the firmware server.

MEAS:FEMask(*):FETC:DATA:RELative?

IVI-COM Equivalent IAgilentN490xEDFastEye.GetRelativeResults()

Syntax MEAS:FEMask(*):FETC:DATA:RELative? nItems

Description Returns a comma-separated list of measured values for nItems data points. If fewer points are available, only the values for available points are returned.

The list has the following parameters:

- First value returned is number of data points actually returned.
- Second value is a flag that indicates whether still more points are available.
- For each returned measurement point, 10 values are returned:
  - dDelay: Delay (data type: NR3).
  - dThreshold: Value for the bit error rate threshold (data type: NR3).
  - dComparedBits: Number of compared bits (data type: NR3).
  - dErroneousBits: Number of errors (data type: NR3).
  - dErroneousZeros: Number of errors from zeros (data type: NR3).
  - dErroneousOnes: Number of errors from ones (data type: NR3).
  - bExtrapolatedFlag: Flag 0 or 1 indicating if the data was calculated or extrapolated by the firmware server.
  - Delay (data type: NR3) as set by "MEAS:FEMask (*):PARameter:POINt[:VALue][?]" on page 266
  - Threshold (data type: NR3) as set by "MEAS:FEMask (*):PARameter:POINt[:VALue][?]" on page 266
  - ThresholdType (data type: NR3) as set by "MEAS:FEMask (*):PARameter:TRESolution:TYPE[?]" on page 265

MEAS:FEMask(*):POINt:PASSed?

IVI-COM Equivalent IAgilentN490xEDFastEyeGetMaskPoint

Syntax MEAS:FEMask(*):POINt:PASSed? nIndex

Description Returns whether the specified measurement point has passed.

MEASure:JTOL:CHAR(*) Subnode

This subnode has the following SCPI structure:

```
MEASure
└ :JTOL:CHARacterization(*)
  ├ :CREate?
  ├ :PARameter
  │ ├ :FLISt[?]
  │ │ └ FILL
  │ ├ :BTESt[?]
  │ │ ├ :TBER[?]
  │ │ ├ :CLEVel[?]
  │ │ ├ :NBITs[?]
  │ │ └ :NERRors[?]
  │ ├ :SEARch[?]
  │ │ ├ :ULINear:STEP[?]
  │ │ ├ :DLINear:STEP[?]
  │ │ ├ :ULOGarithmic
  │ │ │ ├ :STARt[?]
  │ │ │ └ :RATIo[?]
  │ │ ├ :DLOGarithmic
  │ │ │ ├ :STOP[?]
  │ │ │ └ :RATIo[?]
  │ │ ├ :BINary:ACCuracy[?]
  │ │ └ :EBINary:ACCuracy
  │ │   ├ :COARse[?]
  │ │   └ :FINE[?]
  │ └ :RTIMe[?]
  └ :FETCh:DATA
    ├ :AVAilable?
    ├ :POINts?
    └ :CAPability?
```

This subnode has the following commands:

Table 75

Name	Description under
:CREate?	“MEAS:JTOLerance:CHARaracterization:CREate?” on page 272
:PARameter:FLISt[?]	“MEAS:JTOL:CHAR(*):PARameter:FLISt[?]” on page 272
:PARameter:FLISt:FILL	“MEAS:JTOL:CHAR(*):PARameter:FLISt:FILL” on page 272
:PARameter:BTESt[?]	“MEAS:JTOL:CHAR(*):PARameter:BTESt[?]” on page 273
:PARameter:BTESt:TBER[?]	“MEAS:JTOL:CHAR(*):PARameter:BTESt:TBER[?]” on page 273
:PARameter:BTESt:CLEVel[?]	“MEAS:JTOL:CHAR(*):PARameter:BTESt:CLEVel[?]” on page 273
:PARameter:BTESt:NBITs[?]	“MEAS:JTOL:CHAR(*):PARameter:BTESt:NBITs[?]” on page 274
:PARameter:BTESt:NERRors[?]	“MEAS:JTOL:CHAR(*):PARameter:BTESt:NERRors[?]” on page 274
:PARameter:SEARch[?]	“MEAS:JTOL:CHAR(*):PARameter:SEARch[?]” on page 274
:PARameter:SEARch:ULINear:STEP[?]	“MEAS:JTOL:CHAR(*):PAR:SEARch:ULINear:STEP[?]” on page 275
:PARameter:SEARch:DLINear:STEP[?]	“MEAS:JTOL:CHAR(*):PAR:SEARch:DLINear:STEP[?]” on page 275
:PARameter:SEARch:ULOG:STARt[?]	“MEAS:JTOL:CHAR(*):PAR:SEARch:ULOG:STARt[?]” on page 276

Table 75

Name	Description under
:PARameter:SEARch:ULOG:RATIo[?]	“MEAS:JTOL:CHAR (*):PAR:SEARch:ULOG:RATIo[?]” on page 276
:PARameter:SEARch:DLOG:STOP[?]	“MEAS:JTOL:CHAR (*):PAR:SEARch:DLOG:STOP[?]” on page 276
:PARameter:SEARch:DLOG:RATIo[?]	“MEAS:JTOL:CHAR (*):PAR:SEARch:DLOG:RATIo[?]” on page 277
:PARameter:SEARch:BINary:ACC[?]	“MEAS:JTOL:CHAR (*):PAR:SEARch:BINary:ACC[?]” on page 277
:PARameter:SEARch:EBIN:ACC:COARse [?]	“MEAS:JTOL:CHAR (*):PAR:SEARch:EBIN:ACC:COARse[?]” on page 277
:PARameter:SEARch:EBIN:ACC:FINE[?]	“MEAS:JTOL:CHAR (*):PAR:SEARch:EBIN:ACC:[FINE][?]” on page 278
:PARameter:RTIMe[?]	“MEAS:JTOL:CHAR (*):PARameter:RTIMe[?]” on page 278
:FETC:DATA:AVAilable?	“MEAS:JTOL:CHAR (*):FETC:DATA:AVAilable?” on page 279
:FETC:DATA:POINts?	“MEAS:JTOL:CHAR (*):FETC:DATA:POINts?” on page 279
:FETC:DATA:CAPability?	“MEAS:JTOL:CHAR (*):FETC:DATA:CAPability?” on page 280

MEAS:JTOLerance:CHARaracterization:CREate?

IVI-COM Equivalent IAgilentN490xJitterCreateMeasurement

Syntax MEAS:JTOLerance:CHARacterization:CREate?

Description Creates a Jitter Tolerance Characterization measurement object. The return value is the handle used to identify this object by any calls to this object. If, for example, 7 is returned, typical calls to this object would be:

- :MEAS:JTOL:CHAR7:PAR:BTES:TBER 1.0e-10
- :MEAS:GEN7:GO

MEAS:JTOL:CHAR(*):PARameter:FLISt[?]

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurement.FrequencyList

Syntax MEAS:JTOLerance:CHAR(*):PARameter:FLISt (<NR3>, <NR3>, ...)

MEAS:JTOL:CHAR(*):PAR:FLIS?

Description The command specifies a list of jitter frequencies for the measurement, for example 2.5e4, 5.1e6, 4.755e5, and so on. The unit is Hz. The values must be separated by commas.

The query returns the present setting, a comma-separated list.

MEAS:JTOL:CHAR(*):PARameter:FLISt:FILL

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurement.GenerateFrequencyList ()

Syntax MEAS:JTOLerance:CHAR(*):PARameter:FLISt:FILL <NR3>, <NR3>, <NR1>

Description This command specifies the start frequency, the stop frequency, and the number of steps to be used for a Jitter Tolerance Characterization measurement. The frequency unit is Hz.

The individual frequencies are automatically calculated in logarithmically equidistant steps. They are returned by the MEAS:JTOL:CHAR(*):PAR:FLIS? query.

MEAS:JTOL:CHAR(*):PARameter:BTESt[?]

IVI-COM Equivalent	IAgilentN490xJCharacterizationMeasurementBERTestMode
Syntax	MEAS:JTOLerance:CHAR(*):PARameter:BTESt CLEVel | ABSolute MEAS:JTOL:CHAR(*):PAR:BTESt?
Description	The command sets the verification method to either confidence level (CLEVel) or numbers of received bits/errors (ABSolute). In both cases, additional parameters must be specified. The query returns the present setting, CLEV or ABS.

MEAS:JTOL:CHAR(*):PARameter:BTESt:TBER[?]

IVI-COM Equivalent	IAgilentN490xJCharacterizationMeasurement.TargetBER
Syntax	MEAS:JTOLerance:CHAR(*):PARameter:BTESt:TBER <NR3> MEAS:JTOL:CHAR(*):PAR:BTESt:TBER?
Description	This command specifies the target bit error ratio, for example 1.0e-12. The query returns the present value.

MEAS:JTOL:CHAR(*):PARameter:BTESt:CLEVel[?]

IVI-COM Equivalent	IAgilentN490xJCharacterizationMeasurement.TargetBERConfidence
Syntax	MEAS:JTOLerance:CHAR(*):PARameter:BTESt:CLEVel <NR3> MEAS:JTOL:CHAR(*):PAR:BTESt:CLEV?
Description	This command is needed if the verification method is set to CLEVel. It sets the confidence level, for example 0.97. The setting takes effect when the confidence level CLEVel is enabled by means of the MEAS:JTOL:CHAR(*):PAR:BTESt command.

The query returns the present setting.

MEAS:JTOL:CHAR(*):PARameter:BTESt:NBITs[?]

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurementNumberOfBits

Syntax MEAS:JTOLerance:CHAR(*):PARameter:BTESt:NBITs <NR3>

MEAS:JTOL:CHAR(*):PAR:BTESt:NBITs?

Description This command is needed if the verification method is set to ABSolute. It sets the number of bits to be captured and compared, for example 5e9. As soon as this number is exceeded, the measurement proceeds to the next amplitude. The setting takes effect when the mode ABSolute is enabled by means of the MEAS:JTOL:CHAR(*):PAR:BTESt command.

The query returns the present number of bits.

MEAS:JTOL:CHAR(*):PARameter:BTESt:NERRors[?]

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurementNumberOfErrors

Syntax MEAS:JTOLerance:CHAR(*):PARameter:BTESt:NERRors <NR3>

MEAS:JTOL:CHAR(*):PAR:BTESt:NERRors?

Description This command is needed if the verification method is set to ABSolute. It sets the number of errored bits. As soon as this number is exceeded, the measurement proceeds to the next amplitude. The setting takes effect when the mode ABSolute is enabled by means of the MEAS:JTOL:CHAR(*):PAR:BTESt command.

The query returns the present number.

MEAS:JTOL:CHAR(*):PARameter:SEARch[?]

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurement.SearchAlgorithm

Syntax MEAS:JTOLerance:CHAR(*):PARameter:SEARch ULINear | ULOGarithmic |DLINear | DLOGarithmic | BINary | EBINary

MEAS:JTOL:CHAR(*):PAR:SEARch?

Description The command sets the search method to one of the following:

- upwards linear (ULIN)
- upwards logarithmic (ULOG)
- downwards linear (DLIN)
- downwards logarithmic (DLOG)
- binary (BIN)
- extended binary (EBIN)

The query returns the present setting.

MEAS:JTOL:CHAR(*):PAR:SEARch:ULINear:STEP[?]

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurementGetSearchParameter

Syntax MEAS:JTOLerance:CHAR(*):PARameter:SEARch:ULINear:STEP <NR3>

MEAS:JTOL:CHAR(*):PAR:SEARch:ULIN:STEP?

Description The command sets the jitter amplitude step size for an upwards linear search. The unit is UI.

The query returns the present step size.

MEAS:JTOL:CHAR(*):PAR:SEARch:DLINear:STEP[?]

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurement.SetSearchParameter/ GetSearchParameter

Syntax MEAS:JTOLerance:CHAR(*):PARameter:SEARch:DLINear:STEP <NR3>

MEAS:JTOL:CHAR(*):PAR:SEARch:DLIN:STEP?

Description The command sets the jitter amplitude step size for a downwards linear search. The unit is UI.

The query returns the present step size.

MEAS:JTOL:CHAR(*):PAR:SEARch:ULOG:STARt[?]

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurement.SetSearchParameter/ GetSearchParameter

Syntax MEAS:JTOLerance:CHAR(*):PARameter:SEARch:ULOG:STARt <NR3>

MEAS:JTOL:CHAR(*):PAR:SEARch:ULOG:STARt?

Description The command sets the jitter amplitude start value for an upwards logarithmic search. The unit is UI.

The query returns the present value.

MEAS:JTOL:CHAR(*):PAR:SEARch:ULOG:RATIo[?]

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurement.SetSearchParameter/ GetSearchParameter

Syntax MEAS:JTOLerance:CHAR(*):PARameter:SEARch:ULOG:RATIo <NR3>

MEAS:JTOL:CHAR(*):PAR:SEARch:ULOG:RATIo?

Description The command sets the ratio between two successive jitter amplitudes for an upwards logarithmic search. The ratio must be greater than 1.

For example, if the start value is 0.1 UI and the ratio is 2, the jitter amplitude from one measured point to the next will be 0.1 UI, 0.2 UI, 0.4 UI, 0.8 UI, 1.6 UI, and so on.

The query returns the present ratio.

MEAS:JTOL:CHAR(*):PAR:SEARch:DLOG:STOP[?]

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurement.SetSearchParameter/ GetSearchParameter

Syntax MEAS:JTOLerance:CHAR(*):PARameter:SEARch:DLOG:STOP <NR3>

MEAS:JTOL:CHAR(*):PAR:SEARch:DLOG:STOP?

Description The command sets the jitter amplitude stop value for a downwards logarithmic search. The unit is UI.

The query returns the present value.

MEAS:JTOL:CHAR(*):PAR:SEARch:DLOG:RATIo[?]

IVI-COM Equivalent	IAgilentN490xJCharacterizationMeasurement.SetSearchParameter/ GetSearchParameter
Syntax	MEAS:JTOLerance:CHAR(*):PARameter:SEARch:DLOG:RATIo <NR3> MEAS:JTOL:CHAR(*):PAR:SEARch:DLOG:RATIo?
Description	The command sets the ratio between two successive jitter amplitudes for a downwards logarithmic search. The ratio must be smaller than 1. For example, if the start value is 1000 UI and the ratio is 0.5, the jitter amplitude from one measured point to the next will be 1000 UI, 500UI, 250 UI, 125 UI, 62.5 UI, and so on. The query returns the present ratio.

MEAS:JTOL:CHAR(*):PAR:SEARch:BINary:ACC[?]

IVI-COM Equivalent	IAgilentN490xJCharacterizationMeasurement.SetSearchParameter/ GetSearchParameter
Syntax	MEAS:JTOLerance:CHAR(*):PARameter:SEARch:BINary:ACCuracy <NR3> MEAS:JTOL:CHAR(*):PAR:SEARch:BIN:ACC?
Description	The command sets the accuracy of a binary search. This is the minimum amplitude step size for locating the pass/fail transition. The unit is UI. The query returns the present value.

MEAS:JTOL:CHAR(*):PAR:SEARch:EBIN:ACC:COARse[?]

IVI-COM Equivalent	IAgilentN490xJCharacterizationMeasurement.SetSearchParameter/ GetSearchParameter

Syntax MEAS:JTOLerance:CHAR (*):PARameter:SEARch:EBINary:ACCuracy :COARse <NR3>

MEAS:JTOL:CHAR(*):PAR:SEARch:EBIN:ACC:COAR?

Description The command sets the coarse accuracy of an extended binary search. This is the amplitude step size that is used for locating the pass/fail transition coarsely. The unit is UI.

The query returns the present value.

MEAS:JTOL:CHAR(*):PAR:SEARch:EBIN:ACC:[FINE][?]

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurement.SetSearchParameter/ GetSearchParameter

Syntax MEAS:JTOLerance:CHAR(*):PARameter:SEARch:EBINary:ACCuracy [:FINE] <NR3>

MEAS:JTOL:CHAR(*):PAR:SEARch:EBIN:ACC?

Description The command sets the fine accuracy of an extended binary search. This is the amplitude step size that is used after the pass/fail transition has been coarsely located. The unit is UI.

The query returns the present value.

MEAS:JTOL:CHAR(*):PARameter:RTIMe[?]

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurement.DUTRelaxTime

Syntax MEAS:JTOLerance:CHAR(*):PARameter:RTIMe <NR3>

MEAS:JTOL:CHAR(*):PAR:RTIMe?

Description The command sets the DUT relax time. This is the time that takes effect after the measured BER is higher than the target BER. The next BER measurement starts after this time has elapsed. The unit is seconds.

The query returns the present value in seconds.

MEAS:JTOL:CHAR(*):FETC:DATA:AVAilable?

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurement.GetDataCount()

Syntax MEAS:JTOLerance:CHAR(*):FETC:DATA:AVAilable?

Description This query returns the number of measured points in integer format (NR1). The number is continually updated until the measurement has finished.

MEAS:JTOL:CHAR(*):FETC:DATA:POINts?

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurement.GetDataPoints()

Syntax MEAS:JTOLerance:CHAR(*):FETC:DATA:POINts? <NR1>,<NR1>

Description This query returns the results of measured points.

The first parameter specifies the number of points (must be between 1 and the number of measured points, as returned by MEAS:JTOL:CHAR(*):FETC:DATA:AVAilable?), the second parameter specifies the start point (0 is the highest point at the start frequency).

For example, if the number of measured points is 119, you can request

- all points: MEAS:JTOL:CHAR(*):FETC:DATA:POIN? 119,0
- a single point: MEAS:JTOL:CHAR(*):FETC:DATA:POIN? 1,42
- the first 12 points: MEAS:JTOL:CHAR(*):FETC:DATA:POIN? 12,0
- 40 points in the middle: MEAS:JTOL:CHAR(*):FETC:DATA:POIN? 40,40

The results for each point are returned as a comma-separated list with the following contents:

- Frequency in Hz (NR3)
- Jitter amplitude in UI (NR3)
- the measured BER (NR3)
- the number of received bits (NR3)
- the number of errors detected (NR3)
- Passed/failed flag (NR1); 0 = failed, 1 = passed

MEAS:JTOL:CHAR(*):FETC:DATA:CAPability?

IVI-COM Equivalent IAgilentN490xJCharacterizationMeasurement.GetCapbability()

Syntax MEAS:JTOLerance:CHAR(*):FETC:DATA:CAPability? DUT | INST

Description This query returns a comma-separated list that contains one value for each measured frequency.

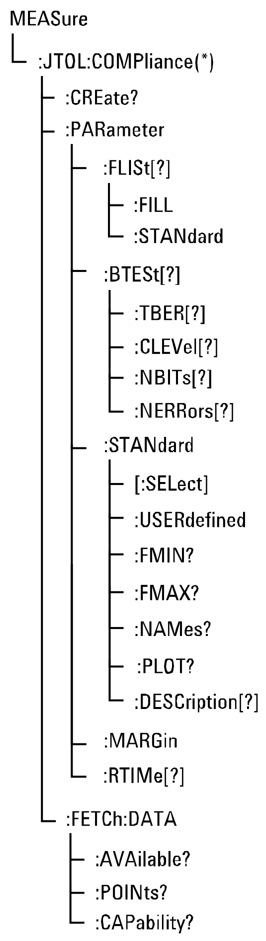
The values depend on the query parameter:

- INSTrument: The maximum jitter amplitude the Serial BERT could generate.
- DUT: The maximum jitter amplitude the device under test could stand without exceeding the target BER.

The unit is UI.

MEASure:JTOL:COMP(*) Subnode

This subnode has the following SCPI structure:



This subnode has the following commands:

Table 76

Name	Description under
:CREate?	"MEAS:JTOLerance:COMPliance:CREate?" on page 283
:PARameter:FLISt[?]	"MEAS:JTOL:COMP (*):PARameter:FLISt[?]" on page 284
:PARameter:FLISt:FILL	"MEAS:JTOL:COMP (*):PARameter:FLISt:FILL" on page 284

Table 76

Name	Description under
:PARameter:FLISt:STANdard	“MEAS:JTOL:COMP (*):PARameter:FLISt:STANdard” on page 284
:PARameter:BTESt[?]	“MEAS:JTOL:COMP (*):PARameter:BTESt[?]” on page 285
:PARameter:BTESt:TBER[?]	“MEAS:JTOL:COMP (*):PARameter:BTESt:TBER[?]” on page 285
:PARameter:BTESt:CLEVel[?]	“MEAS:JTOL:COMP (*):PARameter:BTESt:CLEVel[?]” on page 285
:PARameter:BTESt:NBITs[?]	“MEAS:JTOL:COMP (*):PARameter:BTESt:NBITs[?]” on page 286
:PARameter:BTESt:NERRors[?]	“MEAS:JTOL:COMP (*):PARameter:BTESt:NERRors[?]” on page 286
:PARameter:STANdard[:SELect]	“MEAS:JTOL:COMP (*):PARameter:STANdard[:SELect][?]” on page 286
:PARameter:STANdard:USERdefined	“MEAS:JTOL:COMP (*):PAR:STANdard:USERdefined[?]” on page 287
:PARameter:STANdard:FMIN?	“MEAS:JTOL:COMP (*):PAR:STANdard:FMIN?” on page 288
:PARameter:STANdard:FMAX?	“MEAS:JTOL:COMP (*):PAR:STANdard:FMAX?” on page 288
:PARameter:STANdard:NAMes?	“MEAS:JTOL:COMP (*):PAR:STANdard:NAMes?” on page 288

Table 76

Name	Description under
:PARameter:STANdard:PLOT?	"MEAS:JTOL:COMP (*):PAR:STANdard:PLOT?" on page 289
:PARameter:STANdard:DESCription[?]	"MEAS:JTOL:COMP (*):PAR:STANdard:DESCription[?]" on page 289
:PARameter:MARGin	"MEAS:JTOL:COMP(*):PAR:MARGin" on page 290
:PARameter:RTIMe[?]	"MEAS:JTOL:COMP(*):PAR:RTIMe[?]" on page 290
:FETC:DATA:AVAilable?	"MEAS:JTOL:COMP (*):FETC:DATA:AVAilable?" on page 290
:FETC:DATA:POINts?	"MEAS:JTOL:COMP (*):FETC:DATA:POINts?" on page 290
:FETC:DATA:CAPability?	"MEAS:JTOL:COMP (*):FETC:DATA:CAPability?" on page 291

MEAS:JTOLerance:COMPliance:CREate?

IVI-COM Equivalent IAgilentN490xJitter.CreateMeasurement()

Syntax MEAS:JTOLerance:COMPliance:CREate?

Description Creates a Jitter Tolerance Compliance measurement object. The return value is the handle used to identify this object by any calls to this object. If, for example, 8 is returned, typical calls to this object would be:

- :MEAS:JTOL:COMP8:PAR:BTES:TBER 1.0e-10
- :MEAS:GEN8:GO

MEAS:JTOL:COMP(*):PARameter:FLISt[?]

IVI-COM Equivalent: IAgilentN490xJComplianceMeasurement.FrequencyList

Syntax:

MEAS:JTOLerance:COMP(*):PARameter:FLISt (<NR3>, <NR3>, ...)

MEAS:JTOL:COMP(*):PAR:FLIS?

Description:

The command specifies a list of jitter frequencies for the measurement, for example 2.5e4, 5.1e6, 4.755e5, and so on. The unit is Hz. The values must be separated by commas.

The query returns the present setting, a comma-separated list.

MEAS:JTOL:COMP(*):PARameter:FLISt:FILL

IVI-COM Equivalent: IAgilentN490xJComplianceMeasurement.GenerateFrequencyList()

Syntax: MEAS:JTOLerance:COMP(*):PARameter:FLISt:FILL <NR3>, <NR3>, <NR1>

Description:

This command specifies the start frequency, the stop frequency, and the number of steps to be used for a Jitter Tolerance Compliance measurement. The frequency unit is Hz.

The individual frequencies are automatically calculated in logarithmically equidistant steps. They are returned by the MEAS:JTOL:COMP(*):PAR:FLIS? query.

MEAS:JTOL:COMP(*):PARameter:FLISt:STANdard

IVI-COM Equivalent: IAgilentN490xJComplianceMeasurementPresetFrequencyList

Syntax: MEAS:JTOLerance:COMP(*):PARameter:FLISt:STANdard

Description:

Creates a list of test frequencies for a Jitter Tolerance Compliance measurement. The list contains test frequencies that are recommended for the selected standard: The start frequency of the standard, the edge frequencies, and the stop frequency. The minimum number of test frequencies depends on the selected standard. If the number of steps specified with the

MEAS:JTOL:COMP(*):PAR:FLISt:FILL command is larger than the minimum number of steps, additional frequencies are interpolated.

The individual frequencies are returned by the MEAS:JTOL:COMP (*):PAR:FLIS? query.

MEAS:JTOL:COMP(*):PARameter:BTESt[?]

IVI-COM Equivalent	IAgilentN490xJComplianceMeasurementBERTestMode
Syntax	MEAS:JTOLerance:COMP(*):PARameter:BTESt CLEVel | ABSolute MEAS:JTOL:COMP(*):PAR:BTESt?
Description	The command sets the verification method to either confidence level (CLEVel) or numbers of received bits/errors (ABSolute). In both cases, additional parameters must be specified. The query returns the present setting, CLEV or ABS.

MEAS:JTOL:COMP(*):PARameter:BTESt:TBER[?]

IVI-COM Equivalent	IAgilentN490xJComplianceMeasurementTargerBER
Syntax	MEAS:JTOLerance:COMP(*):PARameter:BTESt:TBER <NR3> MEAS:JTOL:COMP(*):PAR:BTESt:TBER?
Description	This command specifies the target bit error ratio, for example 1.0e-12. The query returns the present value.

MEAS:JTOL:COMP(*):PARameter:BTESt:CLEVel[?]

IVI-COM Equivalent	IAgilentN490xJComplianceMeasurementTargetBERConfidence
Syntax	MEAS:JTOLerance:COMP(*):PARameter:BTESt:CLEVel <NR3> MEAS:JTOL:COMP(*):PAR:BTESt:CLEV?
Description	This command is needed if the verification method is set to CLEVel. It sets the confidence level, for example 0.97. The setting takes

effect when the confidence level CLEVel is enabled by means of the MEAS:JTOL:COMP(*):PAR:BTESt command.

The query returns the present setting.

MEAS:JTOL:COMP(*):PARameter:BTESt:NBITs[?]

IVI-COM Equivalent: IAgilentN490xJComplianceMeasurementNumberOfBits

Syntax: MEAS:JTOLerance:COMP(*):PARameter:BTESt:NBITs <NR3>

MEAS:JTOL:COMP(*):PAR:BTESt:NBITs?

Description: This command is needed if the verification method is set to ABSolute. It sets the number of bits to be captured and compared, for example 5e9. As soon as this number is exceeded, the measurement proceeds to the next frequency. The setting takes effect when the mode ABSolute is enabled by means of the MEAS:JTOL:COMP(*):PAR:BTESt command.

The query returns the present number of bits.

MEAS:JTOL:COMP(*):PARameter:BTESt:NERRors[?]

IVI-COM Equivalent: IAgilentN490xJComplianceMeasurementNumberOfErrors

Syntax: MEAS:JTOLerance:COMP(*):PARameter:BTESt:NERRors <NR3>

MEAS:JTOL:COMP(*):PAR:BTESt:NERRors?

Description: This command is needed if the verification method is set to ABSolute. It sets the number of errored bits. As soon as this number is exceeded, the measurement proceeds to the next frequency. The setting takes effect when the mode ABSolute is enabled by means of the MEAS:JTOL:COMP(*):PAR:BTESt command.

The query returns the present number.

MEAS:JTOL:COMP(*):PARameter:STANdard[:SELect][?]

IVI-COM Equivalent: IAgilentN490xJComplianceMeasurement.Standard

Syntax MEAS:JTOLerance:COMP(*):PARameter:STANdard[:SELect] <standard enum>

MEAS:JTOLerance:COMP(*):PARameter:STANdard[:SELect]?

Description This command selects the standard specified by <standard enum> for the Jitter Tolerance Compliance measurement.

The Serial BERT is equipped with a set of standards provided by Agilent. Every standard is uniquely identified by an abbreviation (<standard enum>). The MEAS:JTOL:COMP (*):PAR:STANdard:NAMes? query lists <standard enum>, <name> pairs, where <name> is the standard's name.

The query returns the <standard enum> of the currently selected standard.

MEAS:JTOL:COMP(*):PAR:STANdard:USERdefined[?]

IVI-COM Equivalent IAgilentN490xJComplianceMeasurement.UserDefinedStandardPoints ()

Syntax MEAS:JTOL:COMP(*):PAR:STANdard:USERdefined (<NR3>, <NR3>, ...)

MEAS:JTOL:COMP(*):PAR:STANdard:USERdefined?

Description The command specifies a user-defined standard. The input parameter is a comma-separated list of jitter frequencies (Hz) and associated jitter amplitudes (UI). For example, (2e3, 110, 3.24e4, 50, 71.234e4, 10, ...).

The <standard enum> of a standard loaded with this command is automatically set to "USER"

Entering a name or a description for the standard is only possible via the user interface. However, the description can be *modified* by means of the MEAS:JTOL:COMP(*):PAR:STANdard:DESCription command.

The query returns a comma-separated list of the current user-defined standard.

MEAS:JTOL:COMP(*):PAR:STANdard:FMIN?

IVI-COM Equivalent IAgilentN490xJComplianceMeasurement.GetFreqMin()

Syntax MEAS:JTOL:COMP(*):PAR:STANdard:FMIN?

Description This query returns the lowest frequency of the frequency range of the selected standard.

MEAS:JTOL:COMP(*):PAR:STANdard:FMAX?

IVI-COM Equivalent IAgilentN490xJComplianceMeasurement.GetFreqMax()

Syntax MEAS:JTOL:COMP(*):PAR:STANdard:FMAX?

Description This query returns the highest frequency of the frequency range of the selected standard.

MEAS:JTOL:COMP(*):PAR:STANdard:NAMes?

IVI-COM Equivalent IAgilentN490xJComplianceMeasurementGetNames

Syntax MEAS:JTOL:COMP(*):PAR:STANdard:NAMes?

Description This query returns a list of all defined standards.

- Every standard provided by Agilent is identified by an abbreviation (<standard enum>) and the standard's name (<name>).

  For example:

  <standard enum> : `CEI6GSR`

  <name> : `CEI 6 Gb/s Short Reach`

- When a *user-defined* standard is added to the list of predefined standards, a <standard enum> is automatically assigned.

The <standard enum> is used for the following commands

- MEAS:JTOL:COMP(*):PAR:STANdard[:SELect]
- MEAS:JTOL:COMP(*):PAR:STANdard:PLOT?

- MEAS:JTOL:COMP(*):PAR:STANdard:DESCription[?].

MEAS:JTOL:COMP(*):PAR:STANdard:PLOT?

IVI-COM Equivalent IAgilentN490xJComplianceMeasurementGetStandardData

Syntax MEAS:JTOL:COMP(*):PAR:STANdard:PLOT? [<standard enum>]

Description Returns a comma-separated list of jitter frequencies (Hz) and associated jitter amplitudes (UI) of the standard specified by <standard enum>. If no input parameter is given, the list of the currently selected standard is returned.

The list of <standard enum> is returned by the MEAS:JTOL:COMP (*):PAR:STANdard:NAMes? query.

The <standard enum> of the currently selected standard is returned by means of the MEAS:JTOL:COMP(*):PARameter:STANdard [:SELect]? query.

MEAS:JTOL:COMP(*):PAR:STANdard:DESCription[?]

IVI-COM Equivalent

Syntax MEAS:JTOL:COMP(*):PAR:STANdard:DESCription USER, <string>

MEAS:JTOL:COMP(*):PAR:STANdard:DESCription? [<standard enum>]

Description The command allows you to *modify* the description of the user-defined standard.

NOTE The descriptions of predefined standards that have been provided by Agilent cannot be modified.

The query returns the description of the standard specified by <standard enum>. If no input parameter is given, the description of the currently selected standard is returned.

The list of <standard enum> is returned by the MEAS:JTOL:COMP (*):PAR:STANdard:NAMes? query.

MEAS:JTOL:COMP(*):PAR:MARGin

IVI-COM Equivalent IAgilentN490xJComplianceMeasurement.Margin

Syntax MEAS:JTOL:COMP(*):PAR:MARGin <NR3>

Description The command sets the margin. The margin increases the jitter amplitudes of the selected standard. A margin of 0.25, for example, has the effect that each amplitude value is multiplied by 1.25.

MEAS:JTOL:COMP(*):PAR:RTIMe[?]

IVI-COM Equivalent IAgilentN490xJComplianceMeasurement.DUTRelaxTime

Syntax MEAS:JTOL:COMP(*):PAR:RTIMe <NR3>

MEAS:JTOL:COMP(*):PAR:RTIMe?

Description The command sets the DUT relax time. This is the time before the measurement proceeds to measure the BER after setting the jitter frequency and amplitude. The next BER measurement starts after this time has elapsed. The unit is seconds.

The query returns the present value in seconds.

MEAS:JTOL:COMP(*):FETC:DATA:AVAilable?

IVI-COM Equivalent IAgilentN490xJComplianceMeasurement.GetDataCount()

Syntax MEAS:JTOLerance:COMP(*):FETC:DATA:AVAilable?

Description This query returns the number of measured points in integer format (NR1). The number is continually updated until the measurement has finished.

MEAS:JTOL:COMP(*):FETC:DATA:POINts?

IVI-COM Equivalent IAgilentN490xJComplianceMeasurement.GetDataPoints()

Syntax MEAS:JTOLerance:COMP(*):FETC:DATA:POINts? <NR1>,<NR1>

Description This query returns the results of measured points.

The first parameter specifies the number of points (must be between 1 and the number of measured points, as returned by MEAS:JTOL:COMP(*):FETC:DATA:AVAilable?), the second parameter specifies the start point (0 is the point at the start frequency).

For example, if the number of measured points is 119, you can request

- all points: MEAS:JTOL:COMP(*):FETC:DATA:POIN? 119,0
- a single point: MEAS:JTOL:COMP(*):FETC:DATA:POIN? 1,42
- the first 12 points: MEAS:JTOL:COMP(*):FETC:DATA:POIN? 12,0
- 40 points in the middle: MEAS:JTOL:COMP(*):FETC:DATA:POIN? 40,40

The results for each point are returned as a comma-separated list with the following contents:

- Frequency in Hz (NR3)
- Jitter amplitude in UI (NR3)
- the measured BER (NR3)
- Passed/failed flag (NR1); 0 = failed, 1 = passed
- Invalid flag (NR1); 0 = invalid, 1 = valid

MEAS:JTOL:COMP(*):FETC:DATA:CAPability?

IVI-COM Equivalent IAgilentN490xJComplianceMeasurement.GetCapability()

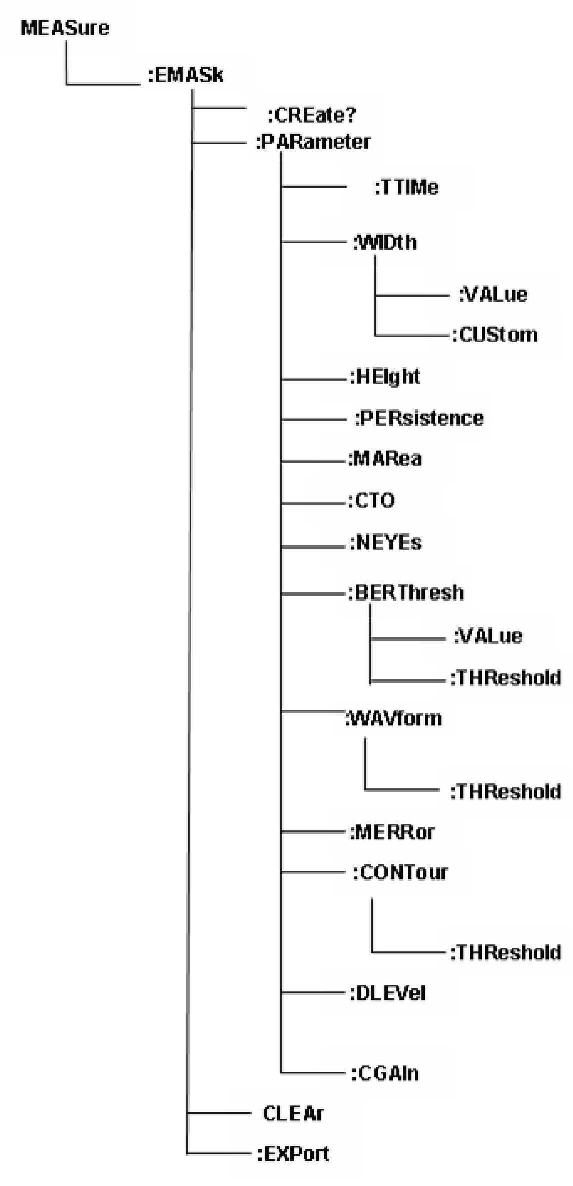
Syntax MEAS:JTOLerance:COMP(*):FETC:DATA:CAPability?

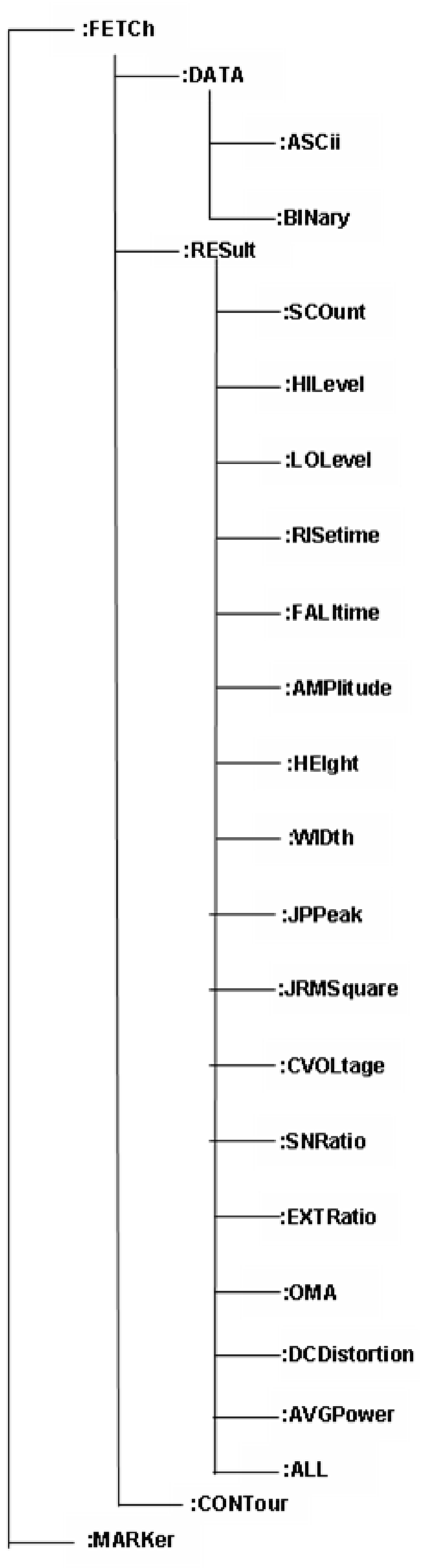
Description This query returns a comma-separated list that contains the maximum jitter amplitude the Serial BERT could generate for each measured frequency.

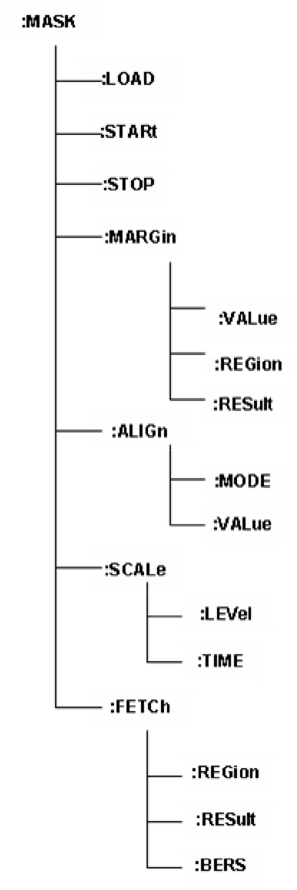
The unit is UI.

MEASure:EMASk(*) Subnode

This subnode has the following SCPI structure:




:FETCh
:DATA
:ASCii
:BINary
:RESult
:SCOunt
:HILevel
:LOLevel
:RISetime
:FALltime
:AMPlitude
:HEIght
:WIDth
:JPPeak
:JRMSquare
:CVOLtage
:SNRatio
:EXTRatio
:OMA
:DCDistortion
:AVGPower
:ALL
:CONTour
:MARKer



This sub node has the following commands:

Table 77

SCPI Name	Description Under
:CREate?	“:MEASure:EMASk(*):CREate?” on page 298
:CLEAr	“:MEASure:EMASk(*):CLEAr” on page 299
:EXPort	“:MEASure:EMASk(*):EXPort” on page 300
:FETCh:DATA?	“:MEASure:EMASk(*):FETCh:DATA?” on page 299

Table 77

SCPI Name	Description Under
:FETCh:CONTour?	“:MEASure:EMASk(*):FETC:CONTour?” on page 306
:MARKer?	“:MEASure:EMASk(*):MARKer?” on page 300
:MASK:LOAD?	“:MEASure:EMASk(*):MASK:LOAD?” on page 301
:MASK:STARt	“:MEASure:EMASk(*):MASK:STARt” on page 301
:MASK:STOP	“:MEASure:EMASk(*):MASK:STOP” on page 301
:MASK:MARGin	“:MEASure:EMASk(*):MASK:MARGin” on page 302
:MARGin:FETch:REGion	“:MEASure:EMASk (*):MARGin:FETch:REGion?” on page 302
:MARGin:FETch:RESult	“:MEASure:EMASk (*):MARGin:FETch:RESult?” on page 303
:MASK:ALIGn	“:MEASure:EMASk(*):MASK:ALIGn” on page 303
:MASk:ALIGn:VALue?	“:MEASure:EMASk (*):ALIGn:VALue:MASk?” on page 304
:MASk:SCALe:LEVel	“:MEASure:EMASk (*):MASk:SCALe:LEVel” on page 304
:MASk:SCALe:TIME	“:MEASure:EMASk (*):MASk:SCALe:TIMe” on page 304
:MASk:FETCh:REGion?	“:MEASure:EMASk (*):MASk:FETCh:REGion?” on page 305

Table 77

SCPI Name	Description Under
:MASk:FETCh:RESult?	“:MEASure:EMASk (*):MASk:FETCh:RESult?” on page 305
:MASk:FETCh:BERS?	“:MEASure:EMASk (*):MASk:FETCh:BERs?” on page 306
:FETCh:RESult:SCOunt?	“:MEASure:EMASk (*):FETCh:RESult:SCOunt?” on page 307
:FETCh:RESult:HILevel?	“:MEASure:EMASk (*):FETCh:RESult:HILevel?” on page 307
:FETCh:RESult:LOLevel?	“:MEASure:EMASk (*):FETCh:RESult:LOLevel?” on page 308
:FETCh:RESult:RISetime?	“:MEASure:EMASk (*):FETCh:RESult:RISetime?” on page 308
:FETCh:RESult:FALtime?	“:MEASure:EMASk (*):FETCh:RESult:FALltime?” on page 308
:FETCh:RESult:AMPlitude?	“:MEASure:EMASk (*):FETCh:RESult:AMPlitude?” on page 309
:FETCh:RESult:HEIght?	“:MEASure:EMASk (*):FETCh:RESult:HEIght?” on page 309
:FETCh:RESult:WIDth?	“:MEASure:EMASk (*):FETCh:RESult:WIDth?” on page 309
:FETCh:RESult:JPPeak?	“:MEASure:EMASk (*):FETCh:RESult:JPPeak?” on page 310

Table 77

SCPI Name	Description Under
:FETCh:RESult:JRMSquare?	“:MEASure:EMASk (*):FETCh:RESult:JRMSquare?” on page 310
:FETCh:RESult:CVOLtage?	“:MEASure:EMASk (*):FETCh:RESult:CVOLtage?” on page 310
:FETCh:RESult:SNRatio?	“:MEASure:EMASk (*):FETCh:RESult:SNRatio?” on page 311
:FETCh:RES:EXTRatio?	“:MEASure:EMASk (*):FETCh:RESult:EXTRatio?” on page 311
:FETCh:RES:OMA?	“:MEASure:EMASk (*):FETCh:RESult:OMA?” on page 312
:FETCh:RES: DCDistortion?	“:MEASure:EMASk (*):FETCh:RESult:DCDistortion?” on page 312
:FETCh:RES: AGVPower?	“:MEASure:EMASk (*):FETCh:RESult:AVGPower?” on page 311
:FETChs:RES:ALL?	“:MEASure:EMASk (*):FETCh:RESult:ALL?” on page 312
:PARameter:TTIMe	“:MEASure:EMASk (*):PARameter:TTIMe” on page 313
:PARameter:WIDth (crossing)	“:MEASure:EMASk (*):PARameter:WIDth” on page 313
:PARameter:WIDth:Custom	“:MEASure:EMASk (*):PARameter:WIDth:Cus” on page 313
:PARameter:HEIght	“:MEASure:EMASk (*):PARameter:HEIght” on page 314

Table 77

SCPI Name	Description Under
:PARameter:PERSistence	“:MEASure:EMASk (*):PARameter:PERsistence” on page 314
:PARameter:MARea	“:MEASure:EMASk (*):PARameter:MARea” on page 314
:PARameter:CTO	“:MEASure:EMASk(*):PARameter:CTO” on page 315
:PARameter:NEYEs	“:MEASure:EMASk (*):PARameter:NEYEs” on page 315
:PARameter:BERT	“:MEASure:EMASk (*):PARameter:BERT” on page 315
:PARameter:BERT:THReshold	“:MEASure:EMASk (*):PARameter:BERTHReshold” on page 316
:PARameter:WAVe:THReshold	“:MEASure:EMASk (*):PARameter:WAVform:THReshold” on page 316
:PARameter:MERRor	“:MEASure:EMASk (*):PARameter:MERRor” on page 316
:PARameter:CONTour:THReshold	“:MEASure:EMASk (*):PARameter:CONTour:THReshold” on page 317
:PARameter:DLEVel	“:MEASure:EMASk (*):PARameter:DLEVel” on page 317
:PARameter:CGAin	“:MEASure:EMASk (*):PARameter:CGAin” on page 317

:MEASure:EMASk(*):CREate?

Syntax :MEASure:EMASk(*):CREate?

Description Creates a Eye Mask measurement object. The return value is the handle used to identify this object by any call to this object. If, for example, 2 is returned, typical calls to this object would be:

- :MEAS:EMAS2:EXP

Input None

Output Integer number representing the measurement handle.

:MEASure:EMASk(*):FETCh:DATA?

Syntax :MEASure:EMASk(*):FETCh:DATA?

Description Gets the measurement data.

Input Integer specifying the number of pixels to be fetched. This number should be larger than the measurement grid, 289 * 353 = 102017.

Output Step Number, Min Threshold, Max Threshold, Min delay, Max Delay, Pixel X location, Pixel Y location, buffer of x*y size representing the pixel values.

e.g.1000,1,1,353,288,-0.579875,0.578419,- 9.69697e-011,9.69697e-011,0.000000000000e(5+number of points fetched).

:MEASure:EMASk(*):CLEAr

Syntax :MEASure:EMASk(*):CLEAr

Description This command clears the eye display data, and restarts the measurement.

Input None

Output None

:MEASure:EMASk(*):EXPort

Syntax :MEASure:EMASk(*):EXPort

Description This command exports the measurement data into the directory, and file name specified by the user. The exported data is in the CSV format. This is the default behavior. The options specified below are for export data with different combinations of BER values.

Input Filename ,ExportDataType : BER, ALL, ExportErrorOption: ALL, EXP1, EXP0, ALLEXP1, EXP1EXP0, ALLEXP0, ALLEXP1EXP0 Delimiter Clipboard : indicates 0 for saving data on file, and 1 for saving on clipboard.

e.g. ‘c:\\Exp\\file18.txt',ALL,ALLEXP1EXP0,9,1

Output None

:MEASure:EMASk(*):MARKer?

Syntax :MEASure:EMASk(*):MARKer?

Description Returns the threshold delay and the BER at the left and right intersections of the marker lines.

Input x coordinate of left marker intersection, y coordinate of left marker intersection, x coordinate of right marker intersection, y coordinate of right marker intersection.

Output First four values are for the eye boundaries, Left Crossover, Right Crossover, and the Edges of the eye (in the vertical direction), followed by the threshold delay, BER of the left intersection, and BER of the right interaction.

For Example:

Input:

meas:emas10:marker? 20,30,40,30

Output:

40,246,89,265,-8.94076628256E-2,-3.717877994884E
+0,4.999990623529E-1,-8.94076628 256E-2,-3.604241631248E
+0,5.000009568395E-1

:MEASure:EMASk(*):MASK:LOAD?

Syntax :MEASure:EMASk(*):MASK:LOAD?

Description Loads the Mask File from the mask file folder provided with the software.

Input A valid mask file.

Output Returns four values, MARGIN | NOMARGIN, Signal time period specified in the mask file, and waveform time period.

In the input file, if min_margin and max_margin are defined for all regions, then the output is MARGIN; and using the margin SCPIs the mask test can resize the margins. If the margins are not defined in the input file, then the output is NOMARGIN, and the margin SCPIs are ineffective.

:MEASure:EMASk(*):MASK:STARt

Syntax :MEASure:EMASk(*):MASK:STARt

Description This aligns the mask according to the parameters set in either the mask file, or the mask properties tab. Then it starts counting the mask violations.

Input None

Output None

:MEASure:EMASk(*):MASK:STOP

Syntax :MEASure:EMASk(*):MASK:STOP

Description Stops counting the mask violations.

Input None

Output None

:MEASure:EMASk(*):MASK:MARGin

Syntax :MEASure:EMASk(*):MASK:MARGin

:MEASure:EMASk(*):MASK:MARGin?

Description The mask definition in the loaded mask file is used to calculate the coordinates of the margin. The mask margin stretches, and shrinks the mask. There is a maximum margin, and a minimum margin defined in the mask file.

Input An input value within the range of -100 to 100.

Output An output result withing the range of -100 to 100.

:MEASure:EMASk(*):MARGin:FETch:REGion?

Syntax :MEASure:EMASk(*):MARGin:FETch:REGion?

Description Fetches the coordinates of the bounding polygon.

For Example: Block of comma separated values

Number of mask regions defined in the file.

- Region Number
- Number of Vertices
- Pixel Coordinates

NOTE The above values will be returned for each mask region.

Input None

Output Scope Coordinates (x,y)

:MEASure:EMASk(*):MARGin:FETch:RESult?

Syntax :MEASure:EMASk(*):MARGin:FETch:RESult?

Description Fetches the coordinates of the mask violations.

For Example: Block of comma separated values

Number of mask regions defined in the file.

- Region Number
- Number of pixels
- Pixel Coordinates for each green pixel
- Number of errors in this region (coordinates)

NOTE The above values will be returned for each mask region.

Input None

Output Scope Coordinates (x,y)

:MEASure:EMASk(*):MASK:ALIGn

Syntax :MEASure:EMASk(*):MASK:ALIGn

:MEASure:EMASk(*):MASK:ALIGn?

Description Sets the mask alignment to either the eye boundaries, or display boundaries. If the loaded mask is of absolute type, then an error message appears, and the alignment remains the same.

Input DISPlay | LEVel | ABSOlute

If the input file type is "ABSO"lute, then the input is ABSOlute, otherwise it is LEVel. The default setting is LEVel.

NOTE If the alignment voltages are mentioned in the mask file then this has no effect.

Output DISPlay | LEVel | ABSOlute

:MEASure:EMASk(*):ALIGn:VALue:MASk?

Syntax :MEASure:EMASk(*):ALIGn:VALue:MASk?

Description Gives the voltage values to which the mask is aligned.

Input None

Output Voltage values to which which the mask in currently aligned.

:MEASure:EMASk(*):MASk:SCALe:LEVel

Syntax :MEASure:EMASk(*):SCALe:MASk:LEVel

:MEASure:EMASk(*):SCALe:MASk:LEVel?

Description This stretches/shrinks, or moves the mask around. When levels (0 and 1) change, the relative coordinates move accordingly. Tracking indicates constant amplitude between 0 and 1 level. Therefore, when tracking is enabled the value of 1 level changes in reference to the value of 0 level, to maintain the constant amplitude.

Input Any real number specifying the voltage value for which the scaling will be done: 0-level, 1-level, OFF|ON

Output Any real number specifying the voltage value for which the scaling is done.

:MEASure:EMASk(*):MASk:SCALe:TIMe

Syntax :MEASure:EMASk(*):MASk:SCALe:TIMe

:MEASure:EMASk(*):MASk:SCALe:TIMe?

Description When the 'position' (left cross over) changes, the position of mask shifts accordingly. The delta Time(dT) defines the position of the right point offset from the 'position', and is used to stretch or shrink the mask.

Input Time in seconds for which scaling will be done: position, delta time.

Output Time in seconds for which scaling has been done.

:MEASure:EMASk(*):MASk:FETCh:REGion?

Syntax :MEASure:EMASk(*):MASk:FETCh:REGion?

Description Fetches the current pixel coordinates of the mask polygons defined in the mask file according to the time offset, dT and 0-1levels, either defined in the mask file or as defined in the scaling options.

For Example: Block of comma separated values

Number of mask regions defined in the file.

- Region Number
- Number of Vertices
- Pixel Coordinates

NOTE The above values will be returned for each mask region.

Input None

Output Coordinates of the Mask Region.

:MEASure:EMASk(*):MASk:FETCh:RESult?

Syntax :MEASure:EMASk(*):MASk:FETCh:RESult?

Description Fetches the pixel coordinates falling wihtin the mask regions.

For Example: Block of comma separated values

Number of mask regions defined in the file.

- Region Number
- Number of pixels
- Pixel Coordinates for each green pixel

- Number of errors in this region (coordinates)

NOTE The above values will be returned for each mask region.

Input None

Output Mask violations.

:MEASure:EMASk(*):MASk:FETCh:BERs?

Syntax :MEASure:EMASk(*):MASk:FETCh:BERs?

Description Gives the highest BER violation in all the regions.

For Example: Block of comma separated values

Number of mask regions defined in the file.

- Region Number
- The worst BER (The greatest BER in that region).

NOTE The above values will be returned for each mask region.

Input None

Output Number of BER violations

:MEASure:EMASk(*):FETC:CONTour?

Syntax :MEASure:EMASk(*):FETC:CONTour?

Description It returns comma separated values. It starts with a series of sizes which create the contours requested, followed by the actual coordinates, which draw the contour. For example, if input is 672, it means 3 contours are requested, at 1e-6, 1e-8 and 1e-10 BER threshold. So, the output will have sizes of each contour, first followed by the points of outermost contour line, then followed by the middle one at a BER threshold of 1e-8 & then the contour line for the 1e-6.

Input Integer input. Switch on the bits for which contours are of interest. For example, to see contour of the BER threshold 0.1 switch on the bit 1. To see contours for BER 1e-6, 1e-8 & 1e-10 the input is, 0x2a0 – decimal values of which is 672.

Output Four values are returned. The first one is the BER value at which the Automated Eye Parameter Measurements are calculated. The second result denotes the lowest BER value. The third result lists the sizes of each contour requested by the user. The fourth result gives the coordinates of each point on the contour.

Results

The following SCPIs are all Queries:

:MEASure:EMASk(*):FETCh:RESult:SCOunt?

Syntax :MEASure:EMASk(*):FETCh:RESult:SCOunt?

Description Gets the sample count.

Input None

Output Sample Count

:MEASure:EMASk(*):FETCh:RESult:HILevel?

Syntax :MEASure:EMASk(*):FETCh:RESult:HILevel?

Description Returns the measurement of the mean value of the logical 1 in the eye diagram. This is directly affected by the values set for Eye Boundaries. See ":MEASure:EMASk(*):PARameter:MARea" on page 314

Input None

Output High Level value.

:MEASure:EMASk(*):FETCh:RESult:LOLevel?

Syntax :MEASure:EMASk(*):FETCh:RESult:LOLevel?

Description Returns the measurement of the mean value of the logical 1 in the eye diagram. This is directly affected by the values set for Eye Boundaries. See ":MEASure:EMASk(*):PARameter:MARea" on page 314

Input None

Output Low Level Value.

:MEASure:EMASk(*):FETCh:RESult:RISetime?

Syntax :MEASure:EMASk(*):FETCh:RESult:RISetime?

Description Rise time is a measurement of the mean transition time of the data on the upward slope of an eye diagram. This transition time is either 2080 or 1090. See ":MEASure:EMASk(*):PARameter:TTIMe" on page 313

Input None

Output Rise Time

:MEASure:EMASk(*):FETCh:RESult:FALltime?

Syntax :MEASure:EMASk(*):FETCh:RESult:FALltime?

Description Fall time is a measurement of the mean transition time of the data on the downward slope of an eye diagram. This transition time is either 2080 or 1090. See ":MEASure:EMASk(*):PARameter:TTIMe" on page 313

Input None

Output Fall Time

:MEASure:EMASk(*):FETCh:RESult:AMPlitude?

Syntax :MEASure:EMASk(*):FETCh:RESult:AMPlitude?

Description Eye amplitude is the difference between the mean logic 1 level values and the mean logic 0 level values in a histogram of an eye diagram.

Input None

Output Eye Amplitude

:MEASure:EMASk(*):FETCh:RESult:HEIght?

Syntax :MEASure:EMASk(*):FETCh:RESult:HEIght?

Description Eye height is a measurement of the vertical opening of an eye diagram. This opening is affected by the BER threshold that is set by the BER threshold command. See “:MEASure:EMASk (*):PARameter:BERT” on page 315

Input None

Output Eye Height

:MEASure:EMASk(*):FETCh:RESult:WIDth?

Syntax :MEASure:EMASk(*):FETCh:RESult:WIDth?

Description Eye width is a measurement of the horizontal opening of an eye diagram. This opening is affected by the BER threshold that is set by the BER threshold command. See “:MEASure:EMASk (*):PARameter:BERT” on page 315

Input None

Output Eye Width

:MEASure:EMASk(*):FETCh:RESult:JPPeak?

Syntax :MEASure:EMASk(*):FETCh:RESult:JPPeak?

Description Jitter P-P is the full width of the eye diagram at the eye crossing point. This is affected by the BER threshold that is set by the BER threshold command. See ":MEASure:EMASk(*):PARameter:BERT" on page 315

Input None

Output Jitter P-P

:MEASure:EMASk(*):FETCh:RESult:JRMSquare?

Syntax :MEASure:EMASk(*):FETCh:RESult:JRMSquare?

Description Jitter Root-Mean Square is the standard deviation of the normal distribution of random jitter. It is dependent on the BER Threshold.

Input None

Output Jitter RMS

:MEASure:EMASk(*):FETCh:RESult:CVOLtage?

Syntax :MEASure:EMASk(*):FETCh:RESult:CVOLtage?

Description Cross Voltage is the measurement of the crossing point level in relation to the logic 1 level and logic 0 level.

Input None

Output Cross Voltage

:MEASure:EMASk(*):FETCh:RESult:SNRatio?

Syntax :MEASure:EMASk(*):FETCh:RESult:SNRatio?

Description Signal-to-Noise Ratio is a measurement of the signal difference between 1 level, and 0 level in relation to the rms value of 1-level noise + rms value of 0-level noise.

Input None

Output Signal to Noise Ratio

:MEASure:EMASk(*):FETCh:RESult:AVGPower?

Syntax :MEASure:EMASk(*):FETCh:RESult:AVGPower?

Description 1-level and 0-level are convereted into power values using Conversion Gain and Dark Level. Then average of 0-level and 1-level power values give Average Power.

NOTE The average power measurement is only available when using the modules with integrated optical detectors (optical/electrical modules)

Input None

Output Average Power

:MEASure:EMASk(*):FETCh:RESult:EXTRatio?

Syntax :MEASure:EMASk(*):FETCh:RESult:EXTRatio?

Description Extinction ratio is the ratio of the energy required to transmit a logic '1' level by the energy required to transmit a logic '0' level. This is dependent on the Dark Level, and Conversion Gain.

Input None

Output	Extinction ratio

:MEASure:EMASk(*):FETCh:RESult:OMA?

Syntax	:MEASure:EMASk(*):FETCh:RESult:OMA?
Description	The Optical Modulation Amplitude (OMA) is the average power measurement, and is only available when using the modules with integrated optical detectors (optical/electrical modules).
Input	None
Output	OMA

:MEASure:EMASk(*):FETCh:RESult:DCDistortion?

Syntax	:MEASure:EMASk(*):FETCh:RESult:DCDistortion?
Description	This value is the difference between the period of a 1 bit and a 0 bit.
Input	None
Output	In unit of time, can be in ps or UI

:MEASure:EMASk(*):FETCh:RESult:ALL?

Syntax	:MEASure:EMASk(*):FETCh:RESult:ALL?
Description	Returns the comma separated list of all the numerical results and their corresponding values.
Input	None
Output	Returns all the numerical results

Parameters

The following SCPIs are both Commands and Queries, the input parameter is for the command and the output parameter is for the queries:

:MEASure:EMASk(*):PARameter:TTIMe

Syntax :MEASure:EMASk(*):PARameter:TTIMe

:MEASure:EMASk(*):PARameter:TTIMe?

Description Sets the transition time.

Input Transition time, can be either 8020 or 9010.

Output Transition time, can be either 8020 or 9010.

:MEASure:EMASk(*):PARameter:WIDth

Syntax :MEASure:EMASk(*):PARameter:WIDth

:MEASure:EMASk(*):PARameter:WIDth?

Description Defines the rule to calculate the eye width. It can be either at the eye crossing point or custom defined (i.e. 40%).

Input Crossing

Output Crossing

:MEASure:EMASk(*):PARameter:WIDth:Cus

Syntax :MEASure:EMASk(*):PARameter:WIDth:Cus

:MEASure:EMASk(*):PARameter:WIDth:Cus?

Description Input parameter for the eye width calculation. The valid range is between 1 and 100. In this case 50% is in the middle of the eye in the verticle direction.

Input A number (percentage) which will be used to calculate the width of the eye.

Output A number which will be the width of the eye.

:MEASure:EMASk(*):PARameter:HEIght

Syntax :MEASure:EMASk(*):PARameter:HEIght

:MEASure:EMASk(*):PARameter:HEIght?

Description Input parameter for the eye height calculation. The valid range is between 1 and 100. In this case 50% is in the middle of the eye in the horizontal direction.

Input A number (percentage), which will be used to calculate the height of the eye.

Output A number, which was set using the command.

:MEASure:EMASk(*):PARameter:PERsistence

Syntax :MEASure:EMASk(*):PARameter:PERsistence

:MEASure:EMASk(*):PARameter:PERsistence?

Description Defines the measurement persistence in seconds.

Input Time in seconds. 2147483647 is used for infinite persistence.

Output Time in seconds.

:MEASure:EMASk(*):PARameter:MARea

Syntax :MEASure:EMASk(*):PARameter:MARea

:MEASure:EMASk(*):PARameter:MARea?

Description Defines the eye window boundaries used for calculating numerical results such as, 0-level, 1-level, SNRatio, rise time, fall time, Amplitude and so forth. The sum of the two boundaries should be 100.

Input	x, y representing the eye window boundaries, defaults are 40,60
Output	x, y representing the eye window boundaries, defaults are 40,60

:MEASure:EMASk(*):PARameter:CTO

Syntax	:MEASure:EMASk(*):PARameter:CTO :MEASure:EMASk(*):PARameter:CTO?
Description	Places the center of the screen either in middle of the eye or at the transition point.
Input	TRANsition | MIDDle
Output	TRANsition | MIDDle

:MEASure:EMASk(*):PARameter:NEYEs

Syntax	:MEASure:EMASk(*):PARameter:NEYEs :MEASure:EMASk(*):PARameter:NEYEs?
Description	Number of eyes to be displayed.
Input	Number of eyes to be shown in UI. Can be either 1.5 or 2.0.
Output	Number of eyes to be shown in UI. Can be either 1.5 or 2.0.

:MEASure:EMASk(*):PARameter:BERT

Syntax	:MEASure:EMASk(*):PARameter:BERT :MEASure:EMASk(*):PARameter:BERT?
Description	This value is used to calculate the measurement results, such as, eye height, eye width, and JPPeak.

Input BER value

Output BER value

:MEASure:EMASk(*):PARameter:BERTHReshold

Syntax :MEASure:EMASk(*):PARameter:BERTHReshold

:MEASure:EMASk(*):PARameter:BERTHReshold?

Description Calculates the measurement parameters for the specified BER threshold. The valid range is 1e-1 to 1e-12.

Input BER threshold

Output BER threshold

:MEASure:EMASk(*):PARameter:WAVform:THReshold

Syntax :MEASure:EMASk(*):PARameter:WAVform:THReshold

:MEASure:EMASk(*):PARameter:WAVform:THReshold?

Description The waveform threshold is a BER value up to which the BER will be represented on the waveform graph. The valid range is 1e-1 to 1e-6.

Input Waveform threshold

Output Waveform threshold

:MEASure:EMASk(*):PARameter:MERRor

Syntax :MEASure:EMASk(*):PARameter:MERRor

:MEASure:EMASk(*):PARameter:MERRor?

Description Minimum number of errors eliminates the low BER values which are statistically uncertain.

Input Integer number specifying in the minimum number of errors you are interested in, and the default value is 2.

Output Integer number giving the minimum number of errors within the valid range between 1.1e18. (40)

:MEASure:EMASk(*):PARameter:CONTour:THReshold

Syntax :MEASure:EMASk(*):PARameter:CONTour:THReshold

:MEASure:EMASk(*):PARameter:CONTour:THReshold?

Description Contour Threshold eliminates the higher BER values for contour calculations, as there are still lots of deterministic jitter effects on those high values. The valid range is 1e-1 to 1e-9.

Input Contour BER threshold

Output The BER value upto which the Contours are calculated and measured.

:MEASure:EMASk(*):PARameter:DLEVel

Syntax :MEASure:EMASk(*):PARameter:DLEVel

:MEASure:EMASk(*):PARameter:DLEVel?

Description Dark Level

Input Dark Level Optical parameter.

Output Returns the Dark Level Optical Parameter.

:MEASure:EMASk(*):PARameter:CGAin

Syntax :MEASure:EMASk(*):PARameter:CGAin

:MEASure:EMASk(*):PARameter:CGAin?

Description Gain Control. Valid range should be greater than 0

Input	Gain Control Optical Parameter
Output	Returns the Gain Control Optical Parameter

TEST Subsystem

TEST Subsystem - Reference

The TEST Subsystem represents the instrument's selftest functions.

```
TEST
├─ :EXECute?
└─ :MESSages?
```

TEST:EXECute?

IVI-COM Equivalent	IIviDriverUtility.SelfTest (IVI-compliant)
Syntax	TEST:EXECute? [SelfTest_value] {,<SelfTest_value>}

Description

This command runs user-specified self tests. If no parameter is specified, all tests are run.

Successful completion of a self test returns 0. If a self test fails, 1 is returned.

SelfTest_value can be one of the parameters listed below.

Table 78

Parameter	Description
ALL	Error detector and pulse generator module self test is started.
PGENerator	Pulse generator module self test is started.

Table 78

Parameter	Description
EDETector	Error detector module self test is started.
PGCal	Auto calibration of pulse generator delay.
EDCal	Auto calibration of error detector delay.

NOTE

Use TEST:MESS? to read the result of the self tests.

TEST:MESSages?

IVI-COM Equivalent: IIviDriverUtility.ErrorQuery (IVI-compliant)

Syntax: TEST:MESSages? PGPOn | EDPOn | EDET | PGEN

Description: Returns a comma-separated list of messages. This command has the following options:

- PGPOn: Pattern Generator Power On messages
- EDPOn: Error Detector Power On messages
- PGEN: Pattern Generator selftest messages
- EDET: Error Detector selftest messages
- PGCal: Pattern Generator Calibration Results
- EDCal: Error Detector Calibration Results.

Index

N4903-91030

Agilent Technologies