Agilent 81141A/81142A
Serial Pulse Data
Generator

Programming Guide

Agilent Technologies
Notices

© Agilent Technologies, Inc. 2007

No part of this manual may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Agilent Technologies, Inc. as governed by United States and international copyright laws.

Manual Part Number
81141-91030

Edition
Release edition, June 2007
Printed in Germany
Agilent Technologies, Deutschland GmbH
Herrenberger Str. 130
71034 Böblingen, Germany

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

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.

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

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

1  Programming Basics

   Programming Basics - Concepts  7

   Before You Begin  8

      Before You Begin - Concepts  8

   Instrument Behavior  10

      Instrument Behavior - Reference  10

2  A Typical Serial Pulse Data Generator Program

   A Typical Serial Pulse Data Generator Program - Concepts  13

   Prerequisites  14

      Prerequisites - Concepts  14

   Initializing the Connection to the Serial Pulse Data Generator  14

      Initializing the Connection - Concepts  14

      Initializing the Connection - Procedures  15

   Working with the IVI-COM Objects  15

      Working with the IVI-COM Objects - Concepts  15

      Working with the IVI-COM Objects - Procedures  16

   Changing Instrument Parameters  16

      Changing Instrument Parameters - Procedures  16

3  Recommended Programming Techniques

   Recommended Programming Techniques - Concepts  19

   Output Protection  19

      Output Protection  19

      Speed DUT Switching  20

   Controlling the Output Levels  21

      Controlling the Output Levels - Concepts  21

      Controlling the Output Levels - Procedures  21
Programming Basics - Concepts

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

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 Pulse Data Generator, 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
- Data Out
- Clock Out
- 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 Pulse Data Generator” on page 9

Communication Overview

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

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

Depending on the options, your Serial Pulse Data Generator 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 Pulse Data Generator**

To communicate with the Serial Pulse Data Generator 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 Pulse Data Generator. For complete instructions on how to establish connections to the Serial Pulse Data Generator, 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 Pulse Data Generator:

- **LAN**
  The Serial Pulse Data Generator'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 Pulse Data Generator's OS is Windows XP). Contact your network administrator if you need help in defining the network settings.

- **GPIB**
  To connect to the Serial Pulse Data Generator via GPIB, you have to have the Serial Pulse Data Generator'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 Pulse Data Generator 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 Pulse Data Generator via USB, you need the Serial Pulse Data Generator'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 Pulse Data Generator behaves as follows when it is turned on (or after a power-cycle):

Instrument Mode

At power on, the Serial Pulse Data Generator will return to the same mode as it was powered down. Normally, once it has booted, the Serial Pulse Data Generator 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 Pulse Data Generator 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 Pulse Data Generator Uses Status Registers” on page 24 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 Pulse Data Generator User Guide (or online Help) for details.

### Operation Modes

The Serial Pulse Data Generator 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 Pulse Data Generator 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.
Programming Basics
A Typical Serial Pulse Data Generator Program - Concepts

The Serial Pulse Data Generator 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 Pulse Data Generator remotely, the client computer (your PC, the Serial Pulse Data Generator is the host) must meet the following prerequisites:

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

Initializing the Connection to the Serial Pulse Data Generator

Initializing the Connection - Concepts

The first step in setting up a program for controlling the Serial Pulse Data Generator is to create an object that corresponds to the instrument. You can either use the Serial Pulse Data Generator 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 Sub Form_Load()
Set myN490X = New AgilentN490x
Set myBERT = myN490x.IAgilentBert
myBERT.Initialize ("TCPIP1::10.0.0.207::inst0::INSTR", True, True, "QueryInstrStatus=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 Pulse Data Generator 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 Pulse Data Generator. 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")
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
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

End Sub

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

For ix = 1 To Len(DataString)

Select Case Format

Case AgilentBertEDPatternFormatBin
    myPattern(ix - 1) = CByte(Mid(DataString, ix, 1))

Case AgilentBertEDPatternFormatHex
    myPattern(ix - 1) = CByte("&H" & Mid(DataString, ix, 1))

End Select

End Function
2 A Typical Serial Pulse Data Generator Program

Case AgilentBertPGPatternFormatRaw
  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 Pulse Data Generator.

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:

- \( V_{\text{Term}} \) remains unchanged.
3 Recommended Programming Techniques

- \( V_{\text{High}} = V_{\text{Term}} + 1 \text{ V} \)
- \( V_{\text{Low}} = V_{\text{Term}} + 0.9 \text{ V} \)

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

**NOTE**

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

   For example, if \( V_{\text{Term}} = 1.0 \text{V} \), 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:
   
   SOURcel: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 Pulse Data Generator's data and clock output ports, four parameters are changed:

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

The Serial Pulse Data Generator 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:

```vba
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
```

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, $V_{lo} = -0.25$).

Changing the Voltages with SCPI
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 Pulse Data Generator to Settle

Allowing Serial Pulse Data Generator to Settle - Concepts

When patterns are sent to the pattern generator, the Serial Pulse Data Generator 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 Pulse Data Generator requires some time to change the patterns at the pattern generator. 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 Pulse Data Generator's Operation Complete status after changing the pattern.

Allowing Serial Pulse Data Generator 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 AgilentN490x.Interop.IAgilentN490xStatus
Dim myED As AgilentN490x.Interop.IAgilentN490xED2
Dim myPG As AgilentN490x.Interop.IAgilentN490xPG2
Set BERTStatus = myBERT.Status
```
Set myPG = myBERT.PGs.Item("PG1")
' First enable the register of interest:
' Operation register, bit 13, positive transition
BERTStatus.Register(AgilentN490xStatusRegisterOperation, _
    AgilentN490xStatusSubRegisterEnable) = &H2000
BERTStatus.Register(AgilentN490xStatusRegisterOperation, _
    AgilentN490xStatusSubRegisterPositiveTransition) = &H0
BERTStatus.Register(AgilentN490xStatusRegisterOperation, _
    AgilentN490xStatusSubRegisterNegativeTransition) = &H2000
' Now clear the registers
BERTStatus.Clear
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

Reading the Serial Pulse Data Generator’s Status

Reading the Serial Pulse Data Generator’s Status - Concepts

The Serial Pulse Data Generator 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 Pulse Data Generator 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 Pulse Data Generator's pattern generator has lost the clock signal.

**How the Serial Pulse Data Generator 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 Pulse Data Generator's status system.

**Overview of the Serial Pulse Data Generator's Status System**

The Serial Pulse Data Generator 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 Pulse Data Generator's Status System Structure**

The Serial Pulse Data Generator'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 Pulse Data Generator's events and conditions. All other individual registers are used to determine the specific events or conditions.

For figures showing Serial Pulse Data Generator's status registers, see “Serial Pulse Data Generator Register Model” on page 26.

**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.
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.

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 Pulse Data Generator's Status - Reference

Depending on the options of your Serial Pulse Data Generator, 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 Pulse Data Generator Register Model

The following figure shows the Serial Pulse Data Generator's status register hierarchy.
**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**

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

**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**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OPC</td>
<td>Operation Complete bit. It is set in response to the *OPC command, but only if the instrument has completed all its pending operations.</td>
</tr>
<tr>
<td>1</td>
<td>Not used</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>QYE</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>DDE</td>
<td>Device-dependent error bit. It is set when an instrument-specific error has occurred.</td>
</tr>
</tbody>
</table>
### Table 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>EXE</td>
<td>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.</td>
</tr>
<tr>
<td>5</td>
<td>CME</td>
<td>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.).</td>
</tr>
<tr>
<td>6</td>
<td>Not used</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>PON</td>
<td>Power-on bit. It is set each time the instrument is powered from off to on.</td>
</tr>
<tr>
<td>8-15</td>
<td>Not used</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PAT GEN</td>
<td>Clock loss condition at the pattern generator.</td>
</tr>
<tr>
<td>2-15</td>
<td>Not used</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>CLOCK LOSS</td>
<td>This bit is set when the pattern generator receives no external clock signal.</td>
</tr>
<tr>
<td>7</td>
<td>PROTECT PG DELAY CTRL IN</td>
<td>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.</td>
</tr>
<tr>
<td>8</td>
<td>UNCAL</td>
<td>This bit is set when the serial number of the installed pattern generator does not match the calibration file in the instrument.</td>
</tr>
</tbody>
</table>

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.
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.

Table 5

<table>
<thead>
<tr>
<th>Bit</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-2</td>
<td>Not used</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>OVERHEAT</td>
<td>The pattern generator has a higher-than-normal temperature.</td>
</tr>
<tr>
<td>5-6</td>
<td>Not used</td>
<td></td>
</tr>
</tbody>
</table>
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

The Serial Pulse Data Generator can use 12 user patterns (UPATtem<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 Pulse Data Generator 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 Pulse Data Generator 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

<table>
<thead>
<tr>
<th>Source</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>LLevel</td>
<td>The signal at <strong>Aux In</strong> controls which half of the pattern is output.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>If</strong> <strong>Aux In</strong>=logic high, pattern B is sent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>If</strong> <strong>Aux In</strong>=logic low, pattern A is sent.</td>
</tr>
<tr>
<td>Internal</td>
<td>Alternate</td>
<td>The AlternatePattern/Select command controls which half of the pattern is output. The options are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pattern A only (A half)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pattern B only (B half)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>alternating A B (AB half)</td>
</tr>
<tr>
<td>Oneshot</td>
<td></td>
<td><strong>IVI-COM:</strong> The BShot command inserts one instance of pattern B into the output.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>SCPI:</strong> The :APCHange:IBHalf command inserts one instance of pattern B into the output.</td>
</tr>
</tbody>
</table>
Table 6

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

How the Serial Pulse Data Generator Sends Triggers

The Serial Pulse Data Generator 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.
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 80.

### 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
- Sends triggers according to the input at AUX IN
- Sends a PRBS pattern to the pattern generator
Working with User Patterns in IVI-COM

Creating Alternate Patterns

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

```vba
Private Sub DefinePatternFile()
' Define the classes;
' myBERT is the already created Serial Pulse Data Generator object
Dim myPG As IAgilentN490xPG
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 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

' Now send the pattern to the instrument
    .SelectData
End With
```

```
Private Sub AlternatePatterns()
    ' Define the classes;
    #39; myBERT is the already created Serial Pulse Data Generator 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 myPGDataOut = 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

Private Sub UsePRBS()
    Dim myPG As IAgilentN490xPG
    Dim myPGOut As IAgilentN490xPGOutput
    Dim myPGTrig As IAgilentN490xPGTrigger
    Dim myPGTrigPos As IAgilentN490xPGPosition
    Dim myPattern() As String
    Dim ix As Integer
    Set myPG = myBERT.PGs("PG1")
    Set myPGOut = myPG.Outputs.Item("PGOutput1")
    Set myPGTrig = myPG.Trigger

    myPGOut.TriggerEnable = AgilentN490xPGOutputTriggerEnableOn
    myPGOut.TriggerSource = AgilentN490xPGOutputTriggerSourceInternal
    myPGOut.TriggerMode = AgilentN490xPGOutputTriggerModeUp
    myPGOut.TriggerThreshold = 50
    myPGOut.TriggerCount = 10
    myPGOut.TriggerInterval = 0.001
    myPGOut.TriggerHoldoff = 0.001
    myPGOut.TriggerZerocross = AgilentN490xPGOutputTriggerZerocrossOn
    myPGOut.TriggerPolarity = AgilentN490xPGOutputTriggerPolarityPositive
    myPGOut.TriggerActive = AgilentN490xPGOutputTriggerActiveOn

    Set myPattern = String(myPGOut.TriggerCount * 100, "0"
    For ix = 0 To myPGOut.TriggerCount - 1
        myPGOut.SetPoint(ix) = myPattern(ix * 100 + 0)
    Next ix
End Sub
Set myPGTrigPos = myPG.Trigger.Position
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.
   SOURcel:PATTern:UPATtern<n>:USE STRaight

2 Set the length of the pattern.
   SOURcel:PATTern:UPATtern<n>[::LENGTH] <NR1>

3 Define how the data is to be packed.
   SOURcel:PATTern:FORMat:DATA PACKed, 1|4|8

4 Define the pattern data.
   SOURcel: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>
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:PAAttern:UPATtern<n>:USE APATtern`

2. Define the length of the pattern.
   
   `SOURce1:PAAttern:UPATtern<n>[:LENGth] <NR1>`

3. Define how the data is to be packed.
   
   `SOURce1:PAAttern:FORMat:DATA PACKed, 1|4|8`

4. Define the data in pattern A.
   
   `SOURce1:PAAttern:UPATtern<n>:DATA A, <block data>`

5. Define the data in the pattern B.
   
   `SOURce1:PAAttern:UPATtern<n>:DATA B, <block data>`

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`

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:PAAttern:SELect UPATtern<n>`

2. Define the source for switching.
   
   `SOURce1:PAAttern:APCHange:SOURce EXTernal | INTernal | BLANking`

3. Define the mode for switching.
   
   `SOURce1:PAAttern:APCHange:MODE ALTernate | ONEShot | LLEVel | REDGe`

4. Use the following command to define which half of the pattern should be sent.
   
   `SOURce1:PAAttern:APCHange:SELect AHALf | BHAlf | ABHALf`
Examples for Using User Patterns in SCPI

To set up a user pattern using SCPI:

1. Define the file 'ALT1s0s.ptrn' to be an alternate pattern.
   
   \texttt{SOURce1:PATTern:UFILe:USE 'ALT1s0s.ptrn', APATtern}

2. Define the input data format to be binary (1s and 0s).
   
   \texttt{SOURce1:PATTern:FORMat:DATA PACKed, 1}

3. Set the pattern length to 8 bits.
   
   \texttt{SOURce1:PATTern:UFILe:LENGth 'ALT1s0s.ptrn', 8}

4. Define pattern A.
   
   \texttt{SOURce1:PATTern:UFILe:DATA A, 'ALT1s0s.ptrn', #1810101010}

5. Define pattern B.
   
   \texttt{SOURce1:PATTern:UFILe:DATA B, 'ALT1s0s.ptrn', #1800000000}

6. Load the pattern to the pattern generator.
   
   \texttt{SOURce1:PATTern:SELect FILENAME, 'ALT1s0s.ptrn'}

Switching at Aux In

With these commands, pattern A is sent when the input at \textbf{Aux In} is low. When the input is high, pattern B is sent.

1. Load the previously defined pattern to the pattern generator.
   
   \texttt{SOURce1:PATTern:SELect FILEName, 'ALT1s0s.ptrn'}

2. Select the source for switching patterns to \textbf{Aux In}.
   
   \texttt{SOURce1:APCHange:SOURce EXTernal}

3. Define that alternate patterns should be sent.
   
   \texttt{SOURce1:APCHange:MODE ALTernate}

Generating a Trigger

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

1. Define the trigger output mode.
   
   \texttt{SOURce3:TRIGger:MODE PATTern}

2. Set up the trigger for pattern changes.
   
   \texttt{SOURce3:TRIGger:APATtern ABCHange}

Switching on the Rising Edge

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

1. Load the pattern to the pattern generator.
Programmatically Switching

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

1. Load the pattern to the pattern generator.
   SOURcel:PATTern:SELect FILENAME, 'ALT1s0s.ptrn'
2. Select the source for changing patterns to be internal.
   SOURcel:APCHange:SOURce INTernal
3. Define that alternate patterns should be sent.
   SOURcel:APCHange:MODE ALTernate
4. Send pattern A continuously.
   SOURcel:PATTern:APCHange:SELect AHALf
5. After some event occurs, change to pattern continuous B.
   SOURcel:PATTern:APCHange:SELect BHALf
6. And then set up output to automatically alternate between pattern A and pattern B.
   SOURcel: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.
   SOURcel:PATTern:SELect FILENAME, 'ALT1s0s.ptrn'
2. Select the source for changing patterns to be internal.
   SOURcel:APCHange:SOURce INTernal
3. Select the mode to insert a single instance of pattern B.
   SOURcel: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.
   SOURcel: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.

```csharp
using System;
using Agilent.AgilentN490x.Interop;
namespace IviSequencing
{
    /// <summary>
    /// Summary description for Class1.
    /// </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 Agilent8114x 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:\81141A\Pattern\MyHeader.ptrn; Block 2= PRBS23, 512, TrigOn;
   Block 3= C:\81141A\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.
SCPI Command Language - Concepts

The Serial Pulse Data Generator is compatible with the standard language for remote control of instruments. Standard Commands for Programmable Instruments (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:
IEEE 488.2 Mandatory and Optional Commands

In order to comply with the SCPI model as described in IEEE 488.2, the Serial Pulse Data Generator 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 58 and “Optional Commands” on page 64.

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 Pulse Data Generator 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]: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 Pulse Data Generator, 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 Pulse Data Generator 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 Pulse Data Generator 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 (#ff, #FF, etc.)

- **BOOLEAN**
  Boolean values can be sent to the Serial Pulse Data Generator as either TRUE | FALSE or 0 | 1. The Serial Pulse Data Generator 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 Pulse Data Generator 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 Pulse Data Generator Register Model” on page 26.

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.

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 Pulse Data Generator - Concepts” on page 55.

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 55 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[:SELection] PRBS7
```

The structure of this command can be illustrated as follows:
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 Serial PULSE Data Generator - 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.

```c
#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::SN81140AXXX::0::INSTR]", VI_NULL, VI_NULL, &vi); */

    /* Or if you have assigned an alias 81140A-Lab */
    /* viOpen (defaultRM, "81140A-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,81140A, 3331U00101,A.01.01".
SCPI Command Reference

Serial Pulse Data Generator 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.

**NOTE**  
The inverted clock and data outputs track the standard outputs. For example, the pattern generator’s 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>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>*CLS</td>
<td>“*CLS “ on page 59</td>
</tr>
<tr>
<td>*ESE[?]</td>
<td>“*ESE[?] “ on page 60</td>
</tr>
<tr>
<td>*ESR?</td>
<td>“*ESR? “ on page 60</td>
</tr>
<tr>
<td>*IDN?</td>
<td>“*IDN? “ on page 60</td>
</tr>
<tr>
<td>*OPC</td>
<td>“*OPC “ on page 61</td>
</tr>
<tr>
<td>*OPC?</td>
<td>“*OPC? “ on page 62</td>
</tr>
<tr>
<td>*RST</td>
<td>“*RST “ on page 63</td>
</tr>
<tr>
<td>*SRE[?]</td>
<td>“*SRE[?] “ on page 63</td>
</tr>
<tr>
<td>*STB?</td>
<td>“*STB? “ on page 63</td>
</tr>
<tr>
<td>*TST?</td>
<td>“*TST? “ on page 64</td>
</tr>
<tr>
<td>*WAI</td>
<td>“*WAI “ on page 64</td>
</tr>
</tbody>
</table>

**CLS**

**IVI-COM Equivalent**

IAgilent8114xStatus.Clear (not IVI-compliant)

**Syntax**

*CLS

**Description**

This command clears all status data structures in a device. For the Serial Pulse Data Generator, 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 Pulse Data Generator’s Status - Concepts” on page 23 for more information about the Status Byte.

**ESE[?]**

**Syntax**

*ESE <Num.>

*ESE?

**IVI-COM Equivalent**

IAgilent8114xStatus.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 Pulse Data Generator’s Status - Reference” on page 26 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**

IAgilent8114xStatus.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**

IlviDriverIdentity (IVI-compliant)

**Syntax**

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
<td>Agilent Technologies</td>
</tr>
<tr>
<td>Model</td>
<td>8114x</td>
</tr>
<tr>
<td>Serial Number</td>
<td>DExxxxxx</td>
</tr>
<tr>
<td>Firmware Level</td>
<td>A.x.x.xxx</td>
</tr>
</tbody>
</table>

*OPC

IVI-COM Equivalent

IAgilent8114xSystem.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 Pulse Data Generator, *OPC can be used with overlapped commands. For more information, see "Overlapped and Sequential Commands” on page 48 the .

---

*OPC?

**IVI-COM Equivalent**
IAgilent8114xSystem.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 Pulse Data Generator, *OPC(?) can be used with overlapped commands. For more information, see “Overlapped and Sequential Commands” on page 48 the .
**RST**

**IVI-COM Equivalent**

IviDriverUtility.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**

IAgilent8114xStatus.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 Pulse Data Generator's Status - Reference” on page 26 for details.

---

**STB?**

**IVI-COM Equivalent**

IAgilent8114xStatus.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 Pulse Data Generator's Status - Reference” on page 26 for details.

*TST?

**IVI-COM Equivalent**
IlviDriverUtility.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 135.

*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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*OPT?</td>
<td>Option Identification Query</td>
</tr>
<tr>
<td>*PSC</td>
<td>Power On Status Clear Command</td>
</tr>
<tr>
<td>*PSC?</td>
<td>Power On Status Clear Query</td>
</tr>
<tr>
<td>*RCL</td>
<td>Recall device setup</td>
</tr>
<tr>
<td>*SAV</td>
<td>Save device setup</td>
</tr>
</tbody>
</table>

*OPT?担负着设备选项的问题。它会返回一个字符串，例如：

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 IAgilent8114xSystem.RecallState (IVI-compliant)

Syntax *RCL <numeric value | string>
This command loads the setup from a numbered store or from a full path filename that was previously stored with "SAV" on page 66. 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.

Depending on the patterns that are saved with the setup, the instrument may require up to half a minute to settle. See "Allowing Serial Pulse Data Generator to Settle - Procedures" on page 22 for details.

*SAV

IVI-COM Equivalent IAgilent8114xSystem.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 65 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>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands</td>
<td></td>
</tr>
<tr>
<td>:PM</td>
<td>&quot;[SOURce[1]]:PM[:STATe][?] &quot; on page 67</td>
</tr>
<tr>
<td>Subnodes</td>
<td></td>
</tr>
<tr>
<td>:PATTern</td>
<td>&quot;[SOURce[1]]:PATTern Subnode&quot; on page 68</td>
</tr>
<tr>
<td>:VOLTage</td>
<td>&quot;[SOURce[1]]:VOLTage Subnode&quot; on page 97</td>
</tr>
</tbody>
</table>

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

**IVI-COM Equivalent**

IAgilent8114xPGDelayControlInput.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>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:EADDition[?]</td>
<td>“[SOURce[1]]:PATTern:EADDition[?]” on page 69</td>
</tr>
<tr>
<td>Name</td>
<td>Description under</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[:EADDition:RATE]?</td>
<td>&quot;[SOURce[1]]:PATTern:EADDition:RATE [?] &quot; on page 70</td>
</tr>
<tr>
<td>[:EADDition:SOURce]?</td>
<td>&quot;[SOURce [1]]:PATTern:EADDition:SOURce[?] &quot; on page 71</td>
</tr>
<tr>
<td>[:FORMat[:DATA]]?</td>
<td>&quot;[SOURce[1]]:PATTern:FORMat[:DATA] [?] &quot; on page 71</td>
</tr>
<tr>
<td>[:MDENsity[:DENSity]]?</td>
<td>&quot;[SOURce[1]]:PATTern:MDENsity [:DENSity][?] &quot; on page 72</td>
</tr>
<tr>
<td>[:SElect]?</td>
<td>&quot;[SOURce[1]]:PATTern[:SElect][?] &quot; on page 72</td>
</tr>
<tr>
<td>[:ZSUBstitut[:ZRUN]]?</td>
<td>&quot;[SOURce[1]]:PATTern:ZSUBstitut [:ZRUN][?]&quot; on page 74</td>
</tr>
</tbody>
</table>

**Subnodes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[:APCHange]</td>
<td>&quot;[SOURce[1]]:PATTern:APCHange Subnode&quot; on page 75</td>
</tr>
<tr>
<td>SEQuence</td>
<td>&quot;[SOURce[1]]:PATTern:SEQuence Subnode&quot; on page 79</td>
</tr>
<tr>
<td>[:UFILe]</td>
<td>&quot;[SOURce[1]]:PATTern:UFILe Subnode&quot; on page 86</td>
</tr>
<tr>
<td>:UPATtern&lt;n&gt;</td>
<td>&quot;[SOURce[1]]:PATTern:UPATtern Subnode&quot; on page 91</td>
</tr>
</tbody>
</table>

This subnode has the following commands:

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

**IVI-COM Equivalent**

IAgilent8114xPGEAddition.InsertManually (not IVI-compliant)
This command is a contraction of the phrase Error ADDition. 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 71). The following table lists the dependencies:

<table>
<thead>
<tr>
<th>:EADD</th>
<th>:EADD:SOUR</th>
<th>:EADD ONCe</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EXT</td>
<td>Active</td>
</tr>
<tr>
<td></td>
<td>FIX</td>
<td>Active</td>
</tr>
<tr>
<td>1</td>
<td>EXT</td>
<td>Active</td>
</tr>
<tr>
<td></td>
<td>FIX</td>
<td>Not active (command has no effect)</td>
</tr>
</tbody>
</table>

The query returns the current state of error addition.

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

**IVI-COM Equivalent**

IAgilent8114xPGErrorAddition.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.

\[[\text{SOURce}[1]]:\text{PAT}tern:EA\text{DD}ition:S\text{OURce}\,?]\]

**IVI-COM Equivalent**
IAgilent8114xPGErrorAddition.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.

\[[\text{SOURce}[1]]:\text{PAT}tern:FORMat[:DATA]\,?]\]

**IVI-COM Equivalent**
IAgilent8114xPGPatternfile.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, :PATTTern: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 40 for descriptions on how to use the data packing.

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

**IVI-COM Equivalent**

IAgilent8114xPGOoutput.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**

IAgilent8114xPGOoutput.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
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**: Zero SUBstitution; 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 74.

**MDENsity**: Mark DENsity; 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 72.

**UPATtern<n>**: User PATtern; used to define the contents of a pattern store. For the Serial Pulse Data Generator, \(<n>\) can be 1 to 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 Pulse Data Generator.

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.
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 79 or loaded from a file with “[SOURce[1]]:PATTern:SEQuence:RCL[?]” on page 84.

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

**IVI-COM Equivalent**

IAgilent8114xPGOutput.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.

![Example of ZSUB pattern](image)

This command is only active when a ZSUB pattern has been selected (see “[SOURce[1]]:PATTern[:SELect][?]” on page 72).
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>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:IBHalf</td>
<td>&quot;[SOURce [1]]:PATTern:APCHange:IBHalf &quot; on page 75</td>
</tr>
<tr>
<td>:MODE[?]</td>
<td>&quot;[SOURce [1]]:PATTern:APCHange:MODE[?] &quot; on page 76</td>
</tr>
<tr>
<td>:SELect[?]</td>
<td>&quot;[SOURce [1]]:PATTern:APCHange:SELect[?] &quot; on page 77</td>
</tr>
<tr>
<td>:SOURce[?]</td>
<td>&quot;[SOURce [1]]:PATTern:APCHange:SOURce[?] &quot; on page 78</td>
</tr>
</tbody>
</table>

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

IVI-COM Equivalent IAigilent8114xPGAuxIn.BShot (not IVI-compliant)
This command is short for **Insert 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 Pulse Data Generator Uses Alternate Patterns” on page 32 for more information.

**Syntax**

[SOURce[1]]:PATTern:APChange:IBHalf ONCe

**Description**

This command is short for **Insert 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 Pulse Data Generator Uses Alternate Patterns” on page 32 for more information.
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:SELoc[?]" on page 77 and "[SOURce [1]]:PATTern:APCHange:SOURce[?]" on page 78.

For instructions on how to use these commands, refer to "How the Serial Pulse Data Generator Uses Alternate Patterns" on page 32.

---

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

**IVI-COM Equivalent**

IAgilent8114xPGAuxIn.AlternatePattern (not IVI-compliant)

**Syntax**

[SOURce[1]]:PATTern:APCHange:SELoc AHALf | BHALf | ABHalf

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

**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 76 and "[SOURce [1]]:PATTern:APCHange:SOURce[?]" on page 78.

For instructions on how to use these commands, refer to “How the Serial Pulse Data Generator Uses Alternate Patterns” on page 32.
The selection ABHalf is new for the Serial Pulse Data Generator.

\[
\text{[SOURce[1]]:PATTern:APChange:SOURce[?]}
\]

<table>
<thead>
<tr>
<th>IVI-COM Equivalent</th>
<th>IAgilent8114xPGAuxIn.Source (not IVI-compliant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>[SOURce[1]]:PATTern:APChange:SOURce EXTernal</td>
</tr>
<tr>
<td>[SOURce[1]]:PATTern:APChange:SOURce?</td>
<td>EXT</td>
</tr>
<tr>
<td>Return Range</td>
<td>EXT</td>
</tr>
<tr>
<td>*RST Value</td>
<td>EXTernal</td>
</tr>
<tr>
<td>Description</td>
<td>This command defines how the Serial Pulse Data Generator determines the pattern to be output. The following alternatives are available:</td>
</tr>
<tr>
<td></td>
<td>• INTernal</td>
</tr>
<tr>
<td></td>
<td>Alternate pattern output is determined internally by the instrument (for example, from the user interface or SCPI commands).</td>
</tr>
<tr>
<td></td>
<td>• EXTernal</td>
</tr>
<tr>
<td></td>
<td>Alternate pattern output is determined by the signal at Aux In. This can either be edge-sensitive or level-sensitive.</td>
</tr>
<tr>
<td></td>
<td>• BLANking</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

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

\[\text{NOTE}\] This command must be used together with the “[SOURce [1]]:PATTern:APChange:MODE[?]” on page 76 and “[SOURce [1]]:PATTern:APChange:SELect[?]” on page 77.

For instructions on how to use these commands, refer to “How the Serial Pulse Data Generator Uses Alternate Patterns” on page 32.
[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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:DATA[?]</td>
<td>&quot;[SOURce [1]]:PATTern:SEQuence:DATA[?]&quot; on page 79</td>
</tr>
<tr>
<td>:EVENt</td>
<td>&quot;[SOURce [1]]:PATTern:SEQuence:EVENt&quot; on page 83</td>
</tr>
<tr>
<td>:RCL</td>
<td>&quot;[SOURce[1]]:PATTern:SEQuence:RCL &quot; on page 84</td>
</tr>
<tr>
<td>:SAVE</td>
<td>&quot;SOURce[1]:PATTern:SEQuence:SAVE &quot; on page 84</td>
</tr>
<tr>
<td>:STATe</td>
<td>&quot;[SOURce [1]]:PATTern:SEQuence:STATe? &quot; on page 85</td>
</tr>
</tbody>
</table>

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

**IVI-COM Equivalent**  IAgilent8114xPG2.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:SEQuence.

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

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:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>optional</td>
</tr>
<tr>
<td>Description</td>
<td>optional</td>
</tr>
<tr>
<td>Start</td>
<td>optional</td>
</tr>
<tr>
<td>Block #</td>
<td>repeated for each block, numbered</td>
</tr>
<tr>
<td>Loop</td>
<td>repeated for every loop, not numbered</td>
</tr>
</tbody>
</table>

The whole sequence expression must be surrounded by paranthesis. 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:\81141A\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 | AuxInHi | AuxInLo | AuxInRising | AuxInFalling | Manual
```

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 \(2^n-1\).
  
  - UserFilename: The path and name of the file that contains the pattern (for example, C:\81141A\Pattern\Upat1.ptrn). Flavor A|B: The first or second half of the pattern to be generated from that file (see “How the Serial Pulse Data Generator Uses Alternate Patterns” on page 32 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 \(<\text{PatternLength}>\times<\text{LoopCount}>\) 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).
  - `INF`: 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:ONCE` is received. Then, sequence execution continues with the next block.

You can use the query `SOURce[1]:PATTern:SEQ: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 (except 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:SEQ:ONCE**

**IVI-COM Equivalent**

`IAgilent8114xPGSequence.Event()`

**Syntax**

```
SOURce[1]:PATTern:SEQ: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 80.

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
IAgilent8114xPGSequence.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:\81141A\Sequences\seq01.seq).

NOTE
The contents of the specified file overwrites the present SequenceExpression. See also “SequenceExpression for User-Defined Sequences” on page 80. You can download the new SequenceExpression to the pattern generator with the command SOURce[1]:PATTern:SElec SEQuence.

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

SOURce[1]:PATTern:SEQuence:SAVE

IVI-COM Equivalent
IAgilent8114xPGSequence.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:\81141A\Sequences\seq01.seq).
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 80.

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

**IVI-COM Equivalent**

IAgilent8114xPGSequence.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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>No user sequence activated</td>
</tr>
<tr>
<td>0</td>
<td>Sequence start condition not fulfilled (init state, no data)</td>
</tr>
<tr>
<td>1 ... 4</td>
<td>The number of the block currently executed</td>
</tr>
</tbody>
</table>

**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).
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:DATA[?]</td>
<td>&quot;[SOURce[1]:PATTern:UFILe:DATA[?] ” on page 86</td>
</tr>
<tr>
<td>:IDATA[?]</td>
<td>&quot;[SOURce[1]:PATTern:UFILe:IDATa “ on page 88</td>
</tr>
<tr>
<td>[:LENGth][?]</td>
<td>&quot;[SOURce[1]:PATTern:UFILe[:LENGth] [?] ” on page 89</td>
</tr>
<tr>
<td>:LABel[?]</td>
<td>&quot;[SOURce[1]:PATTern:UFILe:LABel[?] ” on page 90</td>
</tr>
<tr>
<td>:NAME?</td>
<td>&quot;[SOURce[1]:PATTern:UFILe:NAME? ” on page 90</td>
</tr>
<tr>
<td>:USE[?]</td>
<td>&quot;[SOURce[1]:PATTern:UFILe:USE[?] ” on page 90</td>
</tr>
</tbody>
</table>

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

**IVI-COM Equivalent**: IAgilent8114xLocalPatternfile.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 40 for a detailed description on how to edit user patterns.

The parameters have the following meanings:

Table 21

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A</td>
<td>B]</td>
</tr>
<tr>
<td>&lt;filename&gt;</td>
<td>Name of the file being defined. If the file does not exist, it is created.</td>
</tr>
<tr>
<td>&lt;block data&gt;</td>
<td>The data that describes the pattern (see the following for the description).</td>
</tr>
</tbody>
</table>

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 71.

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.

The <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>
<table>
<thead>
<tr>
<th>#</th>
<th>Start of the header.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of decimal digits to follow to form the length.</td>
</tr>
<tr>
<td>9</td>
<td>Length of the data block (in bytes) that follows.</td>
</tr>
<tr>
<td>&lt;data&gt;</td>
<td>The pattern data, packed according the DATA:PACKed command.</td>
</tr>
</tbody>
</table>

- 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**
IAgilent8114xLocalPatternfile.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 Incremental DATA and is used to download a part of a user-defined pattern.

The parameters have the following meanings:
Table 23

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A</td>
<td>B]</td>
</tr>
<tr>
<td>&lt;filename&gt;</td>
<td>Name of the file being defined. If the file does not exist, it is created.</td>
</tr>
<tr>
<td>&lt;start bit&gt;</td>
<td>First bit to be overwritten (starting with 0).</td>
</tr>
<tr>
<td>&lt;length_in_bits&gt;</td>
<td>Number of bits to be overwritten.</td>
</tr>
<tr>
<td>&lt;block data&gt;</td>
<td>The data that describes the pattern (see “[SOURce[1]]:PATTern:UFILe:DATA[?]” on page 86 for the description).</td>
</tr>
</tbody>
</table>

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 40 for more information on using this command.

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

**IVI-COM Equivalent**

IAgilent8114xLocalPatternfile.Length (IVI-compliant)
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 40 for information on using this command.

```
[SOURce[1]]:PATTern:UFILe:LABel[?]
```

**IVI-COM Equivalent**

IAgilent8114xLocalPatternfile.Description (IVI-compliant)

```
[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 40 for information on using this command.

```
[SOURce[1]]:PATTern:UFILe:NAME?
```

**IVI-COM Equivalent**

IAgilent8114xLocalPatternfile.Location (IVI-compliant)

```
[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:SElec? returns UPAT.

```
[SOURce[1]]:PATTern:UFILe:USE[?]
```

**IVI-COM Equivalent**

IAgilent8114xLocalPatternfile.Alternate (IVI-compliant)

```
[SOURce[1]]:PATTern:UFILe:USE <filename>, STRaight | APATtern
[SOURce[1]]:PATTern:UFILe:USE? <filename>
```
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 Pulse Data Generator Uses Alternate Patterns” on page 32 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 40 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>[:LENGth][?]</td>
<td>“[SOURce[1]]::PATTern:UPATtern&lt;n&gt;:LENGth[?]” on page 92</td>
</tr>
<tr>
<td>[:LABel][?]</td>
<td>“[SOURce[1]]::PATTern:UPATtern&lt;n&gt;:LABel[?]” on page 92</td>
</tr>
</tbody>
</table>
### Table 24

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:USE[?</td>
<td>&quot;[SOURce [1]]:PATTern:UPATtern&lt;n&gt;:USE[?] &quot; on page 93</td>
</tr>
<tr>
<td>:DATA[?</td>
<td>&quot;[SOURce [1]]:PATTern:UPATtern&lt;n&gt;:DATA[?] &quot; on page 93</td>
</tr>
<tr>
<td>:IDATa[?</td>
<td>&quot;[SOURce [1]]:PATTern:UPATtern&lt;n&gt;:IDATa[?] &quot; on page 95</td>
</tr>
</tbody>
</table>

**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][?]**

<table>
<thead>
<tr>
<th>IVI-COM Equivalent</th>
<th>IAgilent8114xPGPatternfile.Length (IVI-compliant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>[SOURce[1]]:PATTern:UPATtern&lt;n&gt;[:LENGth] &lt;numeric value&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce[1]]:PATTern:UPATtern&lt;n&gt;[:LENGth]?</td>
</tr>
<tr>
<td>Description</td>
<td>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 40 for information on using this command.</td>
</tr>
</tbody>
</table>

**[SOURce[1]]:PATTern:UPATtern<n>:LABel[?]**

<table>
<thead>
<tr>
<th>IVI-COM Equivalent</th>
<th>IAgilent8114xPGPatternfile.Description (IVI-compliant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>[SOURce[1]]:PATTern:UPATtern&lt;n&gt;:LABel &lt;string&gt;</td>
</tr>
</tbody>
</table>
[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 40 for information on using this command.

[SOURce[1]]:PATTern:UPATtern<n>:USE[?]  

IVI-COM Equivalent IAgilent8114xPGPatternfile.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 Pulse Data Generator Uses Alternate Patterns” on page 32 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 40 for information on using this command.

[SOURce[1]]:PATTern:UPATtern<n>:DATA[?]

IVI-COM Equivalent IAgilent8114xPGPatternfile.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.
This command is used to set the bits in user pattern files. See “Working with User Patterns in SCPI” on page 40 for a detailed description on how to edit user patterns.

The parameters have the following meanings:

### Table 25

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>([A</td>
<td>B])</td>
</tr>
<tr>
<td>&lt;filename&gt;</td>
<td>Name of the file being defined. If the file does not exist, it is created.</td>
</tr>
<tr>
<td>&lt;block data&gt;</td>
<td>The data that describes the pattern (see the following for the description).</td>
</tr>
</tbody>
</table>

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 71.

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.

The <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>

<table>
<thead>
<tr>
<th>#</th>
<th>Start of the header.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of decimal digits to follow to form the length.</td>
</tr>
<tr>
<td>9</td>
<td>Length of the data block (in bytes) that follows.</td>
</tr>
</tbody>
</table>
The pattern data, packed according to 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
IAgilent8114xPGPatternfile.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 Incremental DATA and is used to download part of a user-defined pattern.

The parameters have the following meanings:

Table 27

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A</td>
<td>B]</td>
</tr>
<tr>
<td>&lt;filename&gt;</td>
<td>Name of the file being defined. If the file does not exist, it is created.</td>
</tr>
</tbody>
</table>
Table 27

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;start bit&gt;</td>
<td>First bit to be overwritten (starting with 0).</td>
</tr>
<tr>
<td>&lt;length_in_bits&gt;</td>
<td>Number of bits to be overwritten.</td>
</tr>
<tr>
<td>&lt;block data&gt;</td>
<td>The data that describes the pattern (see “[SOURce[1]]:PATTern:UFILe:DATA[?]” on page 86 for the description).</td>
</tr>
</tbody>
</table>

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 40 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]?
    [:LEEVel]?
```

This subnode has the following commands:

**Table 28**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>[:ECL]</td>
<td>“[SOURce[1]]:VOLTage:ECL “ on page 97</td>
</tr>
<tr>
<td>[:LEVEL][:IMMediate][:AMPLitude]?</td>
<td>“[SOURce[1]]:VOLTage[:LEVEL][:IMMediate][:AMPLitude]? ” on page 98</td>
</tr>
<tr>
<td>[:LEVEL][:IMMediate]:HIGH?</td>
<td>“[SOURce[1]]:VOLTage[:LEVEL][:IMMediate]:HIGH? ” on page 98</td>
</tr>
<tr>
<td>[:LEVEL][:IMMediate]:LOW?</td>
<td>“[SOURce[1]]:VOLTage[:LEVEL][:IMMediate]:LOW? ” on page 98</td>
</tr>
<tr>
<td>[:LEVEL][:IMMediate]:OFFSet?</td>
<td>“[SOURce[1]]:VOLTage[:LEVEL][:IMMediate]:OFFSet? ” on page 99</td>
</tr>
<tr>
<td>[:LEVEL]:LEEVel?</td>
<td>“[SOURce[1]]:VOLTage[:LEVEL]:LEEVel [?] ” on page 99</td>
</tr>
</tbody>
</table>

**[SOURce[1]]:VOLTage:ECL**

*IVI-COM Equivalent*  
IAgilent8114xPGOutput.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 99).

[SOURce[1]]:VOLTage[:LEVel][:IMMediate][:AMPLitude][?]  

IVI-COM Equivalent  IAgilent8114xPGOutVoltage.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  IAgilent8114xPGOutVoltage.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  IAgilent8114xPGOutVoltage.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**
IAgilent8114xPGOutVoltage.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**
IAgilent8114xPGOutput.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
    - :X0Ver[?]
    - :DCYCle[?]
  - :DEEMphasis
    - :ENABle[?]
    - :MODE[?]
    - [:VALue][?]
    - :IDN?
  - :DELay[?]
  - :FORMat[?]
  - :HOLD[?]
  - :POLarity[?]
  - [:STATe][?]
  - :TERMination[?]
  - :WIDTh[?]
```

This subsystem has the following commands:
Table 29

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:CENTer</td>
<td>“OUTPut[1]:CENTer” on page 101</td>
</tr>
<tr>
<td>:COUPling[?]</td>
<td>“OUTPut[1]:COUPling ” on page 102</td>
</tr>
<tr>
<td>:DATA:XOVer[?]</td>
<td>“OUTPut[1]:DATA:XOVer[?] ” on page 102</td>
</tr>
<tr>
<td>:DCYCl[?]</td>
<td>“OUTPut[1]:DCYCl[?] ” on page 102</td>
</tr>
<tr>
<td>:DEEmphasis:ENABLE[?]</td>
<td>“OUTPut[1]:DEEmphasis:ENABLE[?] ” on page 103</td>
</tr>
<tr>
<td>:DEEmphasis:MODE[?]</td>
<td>“OUTPut[1]:DEEmphasis:MODE[?] ” on page 103</td>
</tr>
<tr>
<td>:DEEmphasis[:VALue][?]</td>
<td>“OUTPut[1]:DEEmphasis[:VALue][?] ” on page 103</td>
</tr>
<tr>
<td>:DEEmphasis:IDN[?]</td>
<td>“OUTPut[1]:DEEmphasis:IDN[?] ” on page 104</td>
</tr>
<tr>
<td>:DELay[?]</td>
<td>“OUTPut[1]:DELay[?] ” on page 104</td>
</tr>
<tr>
<td>:FORMat[?]</td>
<td>“OUTPut[1]:FORMat[?] ” on page 104</td>
</tr>
<tr>
<td>:HOLD[?]</td>
<td>“OUTPut[1]:HOLD[?] ” on page 105</td>
</tr>
<tr>
<td>:POLarity[?]</td>
<td>“OUTPut[1]:POLarity[?] ” on page 105</td>
</tr>
<tr>
<td>[:STATe][?]</td>
<td>“OUTPut[1][:STATe][?] ” on page 106</td>
</tr>
<tr>
<td>:TERMination[?]</td>
<td>“OUTPut[1]:TERMination[?] ” on page 106</td>
</tr>
<tr>
<td>:WIDTh[?]</td>
<td>“OUTPut[1]:WIDTh[?] ” on page 106</td>
</tr>
</tbody>
</table>

**OUTPut[1]:CENTer**

**IVI-COM Equivalent**  IAgilent8114xPGGlobal.OutputsEnabled (not IVI-compliant)
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**

<table>
<thead>
<tr>
<th>IVI-COM Equivalent</th>
<th>IAgilent8114xPGOutput.TerminationEnabled (IVI-compliant)</th>
</tr>
</thead>
</table>

**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[?]**

<table>
<thead>
<tr>
<th>IVI-COM Equivalent</th>
<th>IAgilent8114xPGOutput.Crossover (IVI-compliant)</th>
</tr>
</thead>
</table>

**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[?]**

<table>
<thead>
<tr>
<th>IVI-COM Equivalent</th>
<th>IAgilentBertPGPulse.DutyCycle</th>
</tr>
</thead>
</table>

**Syntax**

```
OUTPut[1]:DCYClE <NR1>
OUTPut[1]:DCYClE?
```
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]:DEEMphasis:ENABle[?]
```

- **IVI-COM Equivalent**: tbd
- **Syntax**: OUTPut[1]:DEEMphasis:ENABle 0 | 1 | OFF | ON
  OUTPut[1]:DEEMphasis:ENABle?

- **Description**: This command enables / disables an N4916A De-Emphasis Signal Converter that is connected between the Data Out of the pattern generator and the DUT. The command is equivalent to pressing the Connect / Disconnect buttons.

  The query returns the present state: 0 | 1.

```
OUTPut[1]:DEEMphasis:MODE[?]
```

- **IVI-COM Equivalent**: tbd
- **Syntax**: OUTPut[1]:DEEMphasis:MODE DB | PERCent
  OUTPut[1]:DEEMphasis:MODE?

- **Description**: This command toggles the unit of the de-emphasis value between dB and percent. The mode is used for interpreting the value.

  The query returns the present state: DB | PERC.

```
OUTPut[1]:DEEMphasis[:VALue][?]
```

- **IVI-COM Equivalent**: tbd
- **Syntax**: OUTPut[1]:DEEMphasis[:VALue] <NR2>
  OUTPut[1]:DEEMphasis[:VALue]?

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]:DEEMphasis?

**Description**: This command sets the de-emphasis value of the N4916A De-Emphasis Signal Converter. The value must conform to the present mode (dB or percent). For example, a de-emphasized signal that reduces the normal amplitude of a bit to one quarter would require a value of 25% or 12 dB. The query returns the present value in floating point format. The value refers to the current mode.

**OUTPut[1]:DEEMphasis:IDN?**

**IVI-COM Equivalent**: tbd

**Syntax**: OUTPut[1]:DEEMphasis:IDN?

**Description**: This query returns the identification of the N4916A De-Emphasis Signal Converter as a string.

**OUTPut[1]:DELay[?]**

**IVI-COM Equivalent**: IAgilent8114xPGOutput.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?
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 123.

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**
IAgilent8114xPGOutput.Polarity (IVI-compliant)

**Syntax**
OUTPut[1]:POLarity NORMal | INVerted
OUTPut[1]:POLarity?

**Return Range**
NORM | INV
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][?]**

**Description**
The command is kept for compatibility reasons. Due to the lack of relays, it is not possible to disable the Serial Pulse Data Generator'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 101.

**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 102).

**OUTPut[1]:WIDTh[?]**

**Description**
This command sets the pulse width of the pattern generator's Data Out port. The response form returns the pulse width.
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.

This subsystem has the following SCPI structure:

```
SOURce9
  :FREQuency[:CW|:FIXed][?]
  :OUTPut[:STATe][?]
```

This subsystem has the following commands:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>on page 108</td>
</tr>
</tbody>
</table>
Table 30

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:OUTPut[:STATe]?</td>
<td>“SOURce9:OUTPut[:STATe]?” on page 108</td>
</tr>
</tbody>
</table>

### SOURce9:FREQuency[:CW | FIXed][?]

**IVI-COM Equivalent**  
IAgilent8114xPGClock.Frequency (IVI-compliant)

**Syntax**  
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**  
IAgilent8114xPGClockIn.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 123. The latter should be preferred.

SOURce2 Subsystem

SOURce2 Subsystem - Reference

The SOURce2 Subsystem represents the pattern generator's Clock Out port.

This subsystem has the following SCPI structure:

```
SOURce2
  :FREQuency [:CW|:FIXed]?
  :VOLTage :ECL [:LEVel][:IMMediate][:AMPLitude][:HIGH][:LOW][:OFFSet][:LLEVel]
```

This subsystem has the following commands:
### Table 31

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:VOLTage:ECL</td>
<td>“SOURce2:VOLTage:ECL” on page 110</td>
</tr>
<tr>
<td>:VOLTage[:LEVEL][:IMMediate][:AMPLitude]?</td>
<td>“SOURce2:VOLTage[:LEVEL][:IMMediate][:AMPLitude]?” on page 111</td>
</tr>
<tr>
<td>:VOLTage[:LEVEL][:IMMediate]:HIGH[?]</td>
<td>“SOURce2:VOLTage[:LEVEL][:IMMediate]:HIGH[?]” on page 111</td>
</tr>
<tr>
<td>:VOLTage[:LEVEL][:IMMediate]:LOW[?]</td>
<td>“SOURce2:VOLTage[:LEVEL][:IMMediate]:LOW[?]” on page 111</td>
</tr>
<tr>
<td>:VOLTage[:LEVEL][:IMMediate]:OFFSet[?]</td>
<td>“SOURce2:VOLTage[:LEVEL][:IMMediate]:OFFSet[?]” on page 112</td>
</tr>
<tr>
<td>:VOLTage[:LEVEL]:LLEVel</td>
<td>“SOURce2:VOLTage:LLEVel[?]” on page 112</td>
</tr>
</tbody>
</table>

**SOURce2:FREQuency[:CW|:FIXed]?**

**IVI-COM Equivalent**

IAgilent8114xPGClockIn.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**

IAgilent8114xPGClock.LogicLevel (not IVI-compliant)
SOURce2:VOLTage:ECL

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 112).

\[
\text{SOURce2:VOLTage[[:LEVel][[:IMMediate]][:AMPLitude]][?]}
\]

IVI-COM Equivalent

IAgilent8114xPGClockVoltage.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.

\[
\text{SOURce2:VOLTage[[:LEVel][[:IMMediate]:HIGH[?]}
\]

IVI-COM Equivalent

IAgilent8114xPGClockVoltage.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.

\[
\text{SOURce2:VOLTage[[:LEVel][[:IMMediate]:LOW[?]}
\]

IVI-COM Equivalent

IAgilent8114xPGClockVoltage.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**

IAgilent8114xPGClockVoltage.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**

IAgilent8114xPGClockVoltage.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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:CENTer</td>
<td>“OUTPut2:CENTer” on page 114</td>
</tr>
<tr>
<td>:COUPling[?]</td>
<td>“OUTPut2:COUPling[?] ” on page 114</td>
</tr>
<tr>
<td>[:STATe][?]</td>
<td>“OUTPut2:[STATe][?] ” on page 114</td>
</tr>
<tr>
<td>:TERMination[?]</td>
<td>“OUTPut2:TERMination[?] ” on page 115</td>
</tr>
</tbody>
</table>
**OUTPut2:CENTer**

**IVI-COM Equivalent**  IAgilent8114xPGGlobal.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 101.

**OUTPut2:COUPling[?]**

**IVI-COM Equivalent**  IAgilent8114xPGClockVoltageTerminationEnabled

**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**  IAgilent8114xPGClock2Enabled

**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 Pulse Data Generator'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 101.
OUTPut2:TERMination[?]

IVI-COM Equivalent
IAgilent8114xPGClockVoltageVTermination

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 114).

SOURce3 Subsystem

SOURce3 Subsystem - Reference

The SOURce3 Subsystem represents the pattern generator's Trigger Out port.

This subsystem has the following SCPI structure:
This subsystem has the following commands:

**Table 33**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger[:MODE][?]</td>
<td>&quot;SOURce3:TRIGger[:MODE][?] &quot; on page 117</td>
</tr>
<tr>
<td>:TRIGger:DCDRatio</td>
<td>&quot;SOURce3:TRIGger:DCDRatio &quot; on page 117</td>
</tr>
<tr>
<td>:TRIGger:CTDRatio</td>
<td>&quot;SOURce3:TRIGger:CTDRatio? &quot; on page 118</td>
</tr>
<tr>
<td>:TRIGger:APATtern&lt;n&gt;[?]</td>
<td>&quot;SOURce3:TRIGger:APATtern&lt;n&gt;[?] &quot; on page 118</td>
</tr>
<tr>
<td>:TRIGger:MDENsity&lt;n&gt;[?]</td>
<td>&quot;SOURce3:TRIGger:MDENsity&lt;n&gt;[?] &quot; on page 119</td>
</tr>
<tr>
<td>:TRIGger:ZSUBstitut&lt;n&gt;[?]</td>
<td>&quot;SOURce3:TRIGger:ZSUBstitut&lt;n&gt;[?] &quot; on page 119</td>
</tr>
<tr>
<td>:TRIGger:PRBN&lt;n&gt;[?]</td>
<td>&quot;SOURce3:TRIGger:PRBN&lt;n&gt;[?] &quot; on page 119</td>
</tr>
<tr>
<td>:TRIGger:PRBS&lt;n&gt;[?]</td>
<td>&quot;SOURce3:TRIGger:PRBS&lt;n&gt;[?] &quot; on page 120</td>
</tr>
</tbody>
</table>
Table 33

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:UPATtern&lt;n&gt;</td>
<td>“SOURce3:TRIGger:UPATtern&lt;n&gt;” on page 120</td>
</tr>
</tbody>
</table>

**NOTE**

See “How the Serial Pulse Data Generator Sends Triggers” on page 35 for details about trigger signals are generated.

### SOURce3:TRIGger[:MODE][]{?}

**IVI-COM Equivalent**

IAgilent8114xPGTrigger.Mode (IVI-compliant)

**Syntax**

SOURce3:TRIGger[:MODE] DCLock | PATTer[n] | SEQuence

SOURce3:TRIGger[:MODE]?

**Return Range**

DCL | PATT | SEQ

**Description**

This command sets the pattern generator's Trigger Out to one of three modes:

- **DCLock**: Sets the Trigger Out to divided clock mode. The divider is set with the command SOURce3:TRIGger:DCDRatio.
- **PATTer[n]**: 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 79 and “SequenceExpression for User-Defined Sequences” on page 80.

### SOURce3:TRIGger:DCDRatio

**IVI-COM Equivalent**

IAgilent8114xPGTrigger.DivisionRate (IVI-compliant)
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
IAgilent8114xPGTrigger.DivisionRate (IVI-compliant)

SOURce3:TRIGger:APATtern<n>

Syntax
SOURce3:TRIGger:APATtern<n> ABCHange | SOPattern

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.

NOTES
This command is for alternate patterns only.
**NOTE**

See “How the Serial Pulse Data Generator Uses Alternate Patterns” on page 32 for additional information on how to work with alternate patterns.

---

**SOURce3:TRIGger:MDENsity<n>**

<table>
<thead>
<tr>
<th>IVI-COM Equivalent</th>
<th>IAgilent8114xPGPosition.Bit (not IVI-compliant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>SOURce3:TRIGger:MDENsity&lt;n&gt; &lt;Num.&gt;</td>
</tr>
<tr>
<td></td>
<td>SOURce3:TRIGger:MDENsity&lt;n&gt;?</td>
</tr>
</tbody>
</table>

**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>**

<table>
<thead>
<tr>
<th>IVI-COM Equivalent</th>
<th>IAgilent8114xPGPosition.Bit (not IVI-compliant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>SOURce3:TRIGger:ZSUBstitut&lt;n&gt; &lt;Num.&gt;</td>
</tr>
<tr>
<td></td>
<td>SOURce3:TRIGger:ZSUBstitut&lt;n&gt;?</td>
</tr>
</tbody>
</table>

**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>**

<table>
<thead>
<tr>
<th>IVI-COM Equivalent</th>
<th>IAgilent8114xPGPosition.Bit (not IVI-compliant)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>SOURce3:TRIGger:PRBN&lt;n&gt; &lt;Num.&gt;</td>
</tr>
<tr>
<td></td>
<td>SOURce3:TRIGger:PRBN&lt;n&gt;?</td>
</tr>
</tbody>
</table>
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 \( \text{<Num>} \) 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**
IAgilent8114xPGPosition.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**
IAgilent8114xPGPosition.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[?]**

<table>
<thead>
<tr>
<th>IVI-COM Equivalent</th>
<th>IAgilent8114xPGClock2.SubRateDivider</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
<td>SOURce5:DIVider &lt;NR1&gt;</td>
</tr>
<tr>
<td></td>
<td>SOURce5:DIVider?</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This command is used to set the clockrate divider. The range is 2 to 128. The query returns the current divider setting.</td>
</tr>
<tr>
<td><strong>NOTE</strong></td>
<td>The subrate clock has a fixed amplitude and no fixed phase correlation with the actual clock.</td>
</tr>
</tbody>
</table>
The SENSe6 Subsystem represents the pattern generator’s Clock In ports.

The Serial Pulse Data Generator 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MODE</td>
<td>“SENSe6:MODE ” on page 123</td>
</tr>
</tbody>
</table>
### SENSE6:_FREQUENCY[:CW | :FIXed]?

**IVI-COM Equivalent**

IAgilent8114xA_PGClockIn.GetFrequency (IVI-compliant)

**Syntax**

SENSe6:FRQENCY [: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 101.

**NOTE**

This query supersedes the following 716xxB command:

- SOURce2:FREQuency[:CW | :FIXed]? <Num.>

---

### SENSE6:MODE

**IVI-COM Equivalent**

IAgilent8114xA_PGClockIn.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 Pulse Data Generator's internal clock oscillator.

- **EXTernal**
  
  Selects the clock signal at the CLK IN port. The frequency is automatically measured and used.

- **EXTMANual**
  
  This command is required for generating signals in RZ or R1 mode while using an external source clock.
In this mode, you can measure the frequency of the external source clock (see “SENSe6:FREQuency[:CW][:FIXed]?” on page 123) and set the clock frequency explicitly (see “SOURce9:FREQuency[:CW][:FIXed]?” on page 108).

- **REFe rence (10 MHz Reference clock)**

  This is the clock signal at the 10 MHz Ref In port. A frequency of 10 MHz is required.

### 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 Pulse Data Generator’s Status - Reference” on page 26.

This subsystem has the following SCPI structure:

```
STATus
  :CLOSs
  ...  
  :OPERation
  ...  
  :PRESet
  :QUEStionable
```

This subsystem has the following commands and subnodes:

<table>
<thead>
<tr>
<th>Table 35</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>Commands</td>
</tr>
<tr>
<td>STATus:PRESet</td>
</tr>
</tbody>
</table>
### STATus:PRESet

**IVI-COM Equivalent**  
IAgilent8114xStatus.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 36**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:CONDition</td>
<td>“STATus:CLOSs:CONDition” on page 126</td>
</tr>
<tr>
<td>:ENABle[?]</td>
<td>“STATus:CLOSs:ENABle[?]” on page 127</td>
</tr>
<tr>
<td>[:EVENT]?</td>
<td>“STATus:CLOSs[:EVENT]?” on page 127</td>
</tr>
<tr>
<td>:NTRansition[?]</td>
<td>“STATus:CLOSs:NTRansition[?]” on page 127</td>
</tr>
<tr>
<td>:PTRansition[?]</td>
<td>“STATus:CLOSs:PTRansition[?]” on page 128</td>
</tr>
</tbody>
</table>

**STATus:CLOSs:CONDition**

**IVI-COM Equivalent**

IAgilent8114xStatus.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 29 for the layout of the Clock Loss register.
## STATus:CLOSs:ENABle[?]

**IVI-COM Equivalent**

IAgilent8114xStatus.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 29 for the layout of the Clock Loss register.

## STATus:CLOSs[:EVENt]?  

**IVI-COM Equivalent**

IAgilent8114xStatus.Register (not IVI-compliant)

**Syntax**

STATus:CLOSs[:EVENt]?

**Description**

The bits in this register indicate pattern generator clock loss. This query returns whether the pattern generator has experienced the clock loss. See “Clock Loss Register” on page 29 for the layout of the Clock Loss register.

## STATus:CLOSs:NTRansition[?]

**IVI-COM Equivalent**

IAgilent8114xStatus.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 29 for the layout of the Clock Loss register.
STATus:CLOSs:PTRansition

**STATus:CLOSs:PTRansition**

**Syntax**

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATus:CLOSs:PTRansition &lt;Num.&gt;</td>
</tr>
<tr>
<td>STATus:CLOSs:PTRansition?</td>
</tr>
</tbody>
</table>

**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 29 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>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:CONDition?</td>
<td>“STATus:OPERation:CONDition?” on page 129</td>
</tr>
<tr>
<td>:ENABle[?]</td>
<td>“STATus:OPERation:ENABle[?]” on page 129</td>
</tr>
<tr>
<td>[:EVENt]?</td>
<td>“STATus:OPERation[:EVENt]?” on page 129</td>
</tr>
</tbody>
</table>

**IVI-COM Equivalent**

IAgilent8114xStatus.Register (not IVI-compliant)
### Table 37

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:NTRansition?</td>
<td>“STATus:OPERation:NTRansition[?]” on page 130</td>
</tr>
<tr>
<td>:PTRansition?</td>
<td>“STATus:OPERation:PTRansition[?]” on page 130</td>
</tr>
</tbody>
</table>

**STATus:OPERation:CONDition?**

- **IVI-COM Equivalent**: IAgilent8114xStatus.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 30 for the layout of the Operation Status register.

**STATus:OPERation:ENABle[?]**

- **IVI-COM Equivalent**: IAgilent8114xStatus.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 30 for the layout of the Operation Status register.

**STATus:OPERation[:EVENt]?**

- **IVI-COM Equivalent**: IAgilent8114xStatus.Register (not IVI-compliant)
- **Syntax**: STATus:OPERation[:EVENt]?
This query returns the contents of the Operation Status event register. See "Operation Status Register * on page 30 for the layout of the Operation Status register.

Note that reading the event register clears it.

STATus:OPERation:NTRansition[?]

IVI-COM Equivalent IAgilent8114xStatus.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 30 for the layout of the Operation Status register.

STATus:OPERation:PTRansition[?]

IVI-COM Equivalent IAgilent8114xStatus.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 30 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 38**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ENABLE[?]</td>
<td>“STATus:QUEStionable:ENABLE[?] “ on page 132</td>
</tr>
<tr>
<td>[:EVENt]?</td>
<td>“STATus:QUEStionable[:EVENt]?” on page 132</td>
</tr>
<tr>
<td>:NTRansition[?]</td>
<td>“STATus:QUEStionable:NTRansition[?]” on page 132</td>
</tr>
<tr>
<td>:PTRansition[?]</td>
<td>“STATus:QUEStionable:PTRansition[?]” on page 133</td>
</tr>
</tbody>
</table>

**STATus:QUEStionable:CONDition?**

**IVI-COM Equivalent**

IAgilent8114xStatus.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 30 for the layout of the Questionable Status register.
STATus:QUESTionable:ENABle[?]

IVI-COM Equivalent  IAgilent8114xStatus.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 30 for the layout of the Questionable Status register.

STATus:QUESTionable[:EVENt]?

IVI-COM Equivalent  IAgilent8114xStatus.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 30 for the layout of the Questionable Status register.

STATus:QUESTionable:NTRansition[?]

IVI-COM Equivalent  IAgilent8114xStatus.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 30 for the layout of the Questionable Status register.
STATus:QUESTionable:PTRansition[?]

IVI-COM Equivalent  IAgilent8114xStatus.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 30 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
    :GPIB[?]
    :HELP
    :HEADers?
    :VERSion?
```

This subsystem has the following commands:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:GPIB[?]</td>
<td>“SYSTem:GPIB[?]” on page 134</td>
</tr>
</tbody>
</table>
Table 39

<table>
<thead>
<tr>
<th>Name</th>
<th>Description under</th>
</tr>
</thead>
<tbody>
<tr>
<td>:HELP:HEADers?</td>
<td></td>
</tr>
<tr>
<td>:VERSion?</td>
<td>“SYSTem:VERSion?” on page 134</td>
</tr>
</tbody>
</table>

**SYSTem:GPIB[?]**

- **IVI-COM Equivalent**: IAgilent8114xUtilities.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:VERSion?**

- **Syntax**: SYSTem:VERSion?
- **Description**: This query returns the version of the SCPI programming language, which supports the GPIB commands.
TEST Subsystem

TEST Subsystem - Reference

The TEST Subsystem represents the instrument’s selftest functions.

```
TEST
  :EXECute?
  :MESSages?
```

**TEST:EXECute?**

**IVI-COM Equivalent**  
IviDriverUtility.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 40**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGENerator</td>
<td>Pulse generator module self test is started.</td>
</tr>
<tr>
<td>PGCal</td>
<td>Auto calibration of pulse generator delay.</td>
</tr>
</tbody>
</table>

**NOTE**  
Use TEST:MESS? to read the result of the self tests.
**TEST:MESSages?**

**IVI-COM Equivalent**
IviDriverUtility.ErrorQuery (IVI-compliant)

**Syntax**
TEST:MESSages? PGPOn | PGEN

**Description**
Returns a comma-separated list of messages. This command has the following options:
- PGPOn: Pattern Generator Power On messages
- PGEN: Pattern Generator selftest messages
- PGCal: Pattern Generator Calibration Results
## Index

### C
- **Clock In**
  - Pattern Generator, 122
- **Clock Out**
  - Pattern Generator (OUTPut 2), 113
  - Pattern Generator (SOURce2), 109
  - Pattern Generator (SOURce9), 107
- Command subsystems, 57

### D
- **Data Out**
  - OUTPut[1], 100
  - SOURce[1], 66
- **De-Emphasis**
  - SCPI commands, 103

### I
- **IEEE commands**
  - mandatory, 58
  - optional, 64
- **Interrupts**, 31
- **IVI-COM**, 13

### O
- **Operation Modes**, 11

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPut2</td>
<td>113</td>
</tr>
<tr>
<td>OUTPut[1]</td>
<td>100</td>
</tr>
</tbody>
</table>

### P
- **Pattern Generator**
  - Clock In port, 122
  - Clock Out (volt) commands, 113
  - Clock Out commands (SOURce2), 109
  - Clock Out commands (SOURce9), 107
  - Data Out commands (OUTPut[1]), 100
  - Data Out commands (SOURce[1]), 66
  - Sub Clock Out commands (SOURce5), 121
  - Trigger Out commands, 115
  - User sequence, 80
  - Programming
    - Data Types, 50

### R
- **Register Model**, 26
- **Remote Control**
  - Communication, 8
  - Connections, 9

### S
- **SCPI Commands**, 47
  - Separators, 53
  - Syntax, 51
- **SENSe subsystems**, 57
- **SENSe6**, 122
- **SequenceExpression for User-Defined Sequences**, 80
- **SOURce subsystems**, 57
- **SOURce2**, 109
- **SOURce3**, 115
- **SOURce5**, 121
- **SOURce9**, 107
- **SOURce[1]**, 66
- **Status subsystem**, 58
- **Sub Clock Out**
  - Pattern Generator (SOURce5), 121
- **System subsystem**, 58

### T
- **Test subsystem**, 58
- **Trigger Out**
  - Pattern Generator, 115