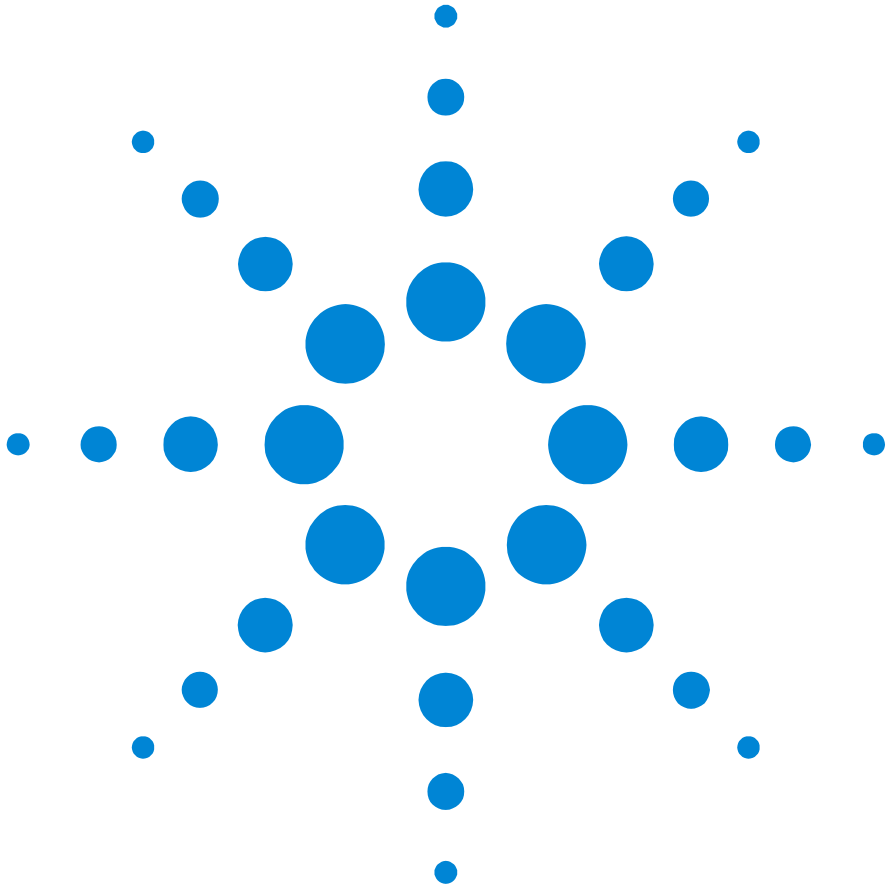


Agilent 81250 Parallel Bit Error Ratio Tester

System Programming Guide and SCPI Reference

Agilent Technologies

www.valuetronics.com

Important Notice

© Agilent Technologies, Inc. 2003

Manual Part Number

5988-4886EN

Revision

Revision 5.1, February 2003

Printed in Germany

Agilent Technologies
Herrenberger Straße 130
D-71034 Böblingen
Germany

Authors: t3 medien GmbH

Warranty

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

Technology Licenses

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

Restricted Rights Legend

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

Safety Notices

CAUTION

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

WARNING

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

Trademarks

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

www.valuetronics.com

Contents

www.valuetronics.com

www.valuetronics.com

www.valuetronics.com

www.valuetronics.com

www.valuetronics.com

www.valuetronics.com

www.valuetronics.com

www.valuetronics.com

www.valuetronics.com

www.valuetronics.com

www.valuetronics.com

www.valuetronics.com

www.valuetronics.com

www.valuetronics.com

Programming Interfaces

The Agilent 81250 provides, besides the user interface, the following control possibilities:

- VXI PnP

  This is the recommended method for controlling the Agilent 81250. For details on using VXI PnP, see the `hp81200.hlp` provided in the help subdirectory of your Agilent 81250 installation.

- Agilent IO Libraries

  You can use the Agilent IO Libraries for controlling the Agilent 81250 over a GPIB or emulated GPIB interface. For details, see the documentation delivered with the Agilent IO Libraries. The *LAN Programming Guide* (available in the online Help) also provides SICL- and VISA-based examples that can be used for non-LAN configurations.

- Agilent String Interface Library

  The Agilent String Interface Library is a simple library available in C and VisualBasic for sending ASCII commands (such as SCPI commands) to and querying ASCII strings from the Agilent 81250.

  "Agilent 81250 String Interface Library" on page 20 summarizes the major aspects to be considered when implementing programs for local control or for remote control via LAN using the string interface.

- Command Line Interface

  In addition to these programming models, the Command Line Interface window provided by the user interface can be used to try out commands, and to store and replay simple sequences of commands (see *"Command Line Interface" on page 26*).

- Telnet and Socket Programs

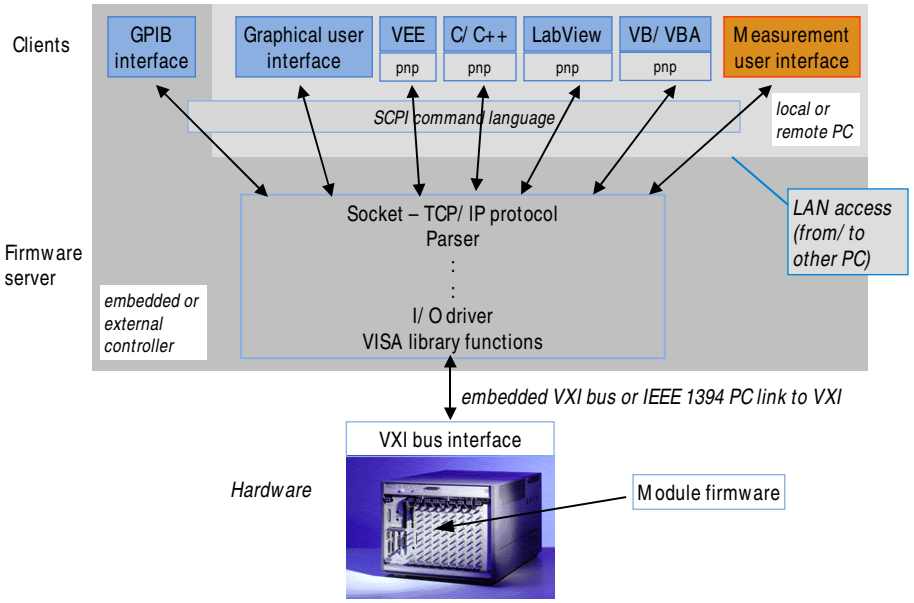
  The *LAN Programming Guide* (available in the online Help) provides the information necessary for controlling the Agilent 81250 using telnet or socket programming.

www.valuetronics.com

Programming Models

NOTE Instead of using the SCPI commands, we recommend to use the plug & play programming interface. For an online help on the VXIPNP functions, refer to `hp81200.hlp` provided in the help subdirectory of your Agilent 81250 installation.

The following picture shows the basic structure of the Agilent 81250's programming interfaces.



Firmware Server and SCPI Commands

The firmware server communicates with the hardware using the SCPI language. It implements a client-server architecture, allowing multiple clients to connect simultaneously.

The Agilent 81250 plug & play drivers also communicate with the firmware server using the SCPI language. Therefore, everything that can be done over the user interface can also be done using the other programming interfaces (with, of course, loss of comfort). For detailed information on the Agilent 81250 plug & play drivers, please refer to the corresponding online help files.

Tools and Programming Languages

Depending on your configuration, tools like VEE or LabView, and the programming languages C/C++ and VisualBasic can be used for implementation.

www.valuetronics.com

NOTE You can also send SCPI commands directly to the system by using the Agilent String Interface Library. Examples and reference information are provided under *"Agilent 81250 String Interface Library" on page 20*.

Locking Mechanisms

Different clients may operate on the same system concurrently using different interfaces. To avoid undesired interrupts, the Agilent 81250 features a locking mechanism, allowing each client to gain exclusive access to the system during critical operations.

The system can either be locked or blocked:

- With a locked system, commands from other clients are returned with an error message.
- With a blocked system, commands from other clients do not return until the system has been unlocked and the command has been executed.

www.valuetronics.com

Agilent 81250 String Interface Library

The Agilent 81250 String Interface library provides basic functions for connecting to an Agilent 81250 firmware server either on the local machine or via LAN.

C Implementation To implement the Agilent 81250 String Interface library in C, include the file `hp81200.h`.

VB Implementation To implement the Agilent 81250 String Interface library in VisualBasic, import the file `hp81200.bas` into your project.

The following commands are available:

C Command	VB Command	Described under:
Connect_HP81200	VB_CONNECT_HP81200	*"Connect_HP81200" on page 21*
Call_HP81200	VB_CALL_HP81200	*"Call_HP81200" on page 22*
GetErrStr_HP81200	VB_GETERRSTR_HP81200	*"GetErrStr_HP81200" on page 23*
Disconnect_HP81200	VB_DISCONNECT_HP81200	*"Disconnect_HP81200" on page 23*

A program using the Agilent 81250 String Interface library has the following structure:

- Connect to the firmware server.
- Send any SCPI commands or queries.
- Handle errors as necessary.
- Disconnect from the firmware server.

NOTE It is recommended that you explicitly disconnect from the firmware server at the end of your program. Otherwise, it may not be possible to connect again.

See *"Example Implementations" on page 24* for code examples in C and VisualBasic.

www.valuetronics.com

Error Handling

All functions in this library return an error code.

- A return value of 0 always means that the function call succeeded.
- A negative number indicates a server error. Use the function `GetErrStr_HP81200` with the error code as an argument to get a human-readable error string.
- A positive number indicates that the command execution failed. In this case, query the error queue corresponding to the last command. That means, if the last command started with `:dvt:`, send the command `:dvt:syst:err?` until the error queue is empty. If this does not locate the problem, send `:<Handle>:syst:err?` using the handle of your system.

Connect_HP81200

C Syntax `int HP81200_CDECL Connect_HP81200(const char *ServerName)`

VB Syntax `VB_CONNECT_HP81200(ByVal ServerName As String) As Long`

Description Connects to an Agilent 81250 server. This must be the first call when using the Agilent 81250 string interface library.

Parameters **ServerName** The machine name or IP address of the machine to which you want to connect. If left empty (NULL pointer or empty string), the local host is connected.

In general `ServerName` has the structure `[<host>[<sep><port>]]`

- `<host>` is the hostname in text (mypc.bbn.hp) or IP notation (15.100.10.20). The default is 127.0.0.1 (“localhost”).
- `<sep>` is a separator; this can be a null byte, space, comma, semicolon, colon, tab, or return. For clarity, a semicolon is used as separator in the examples.
- `<port>` is the port number as decimal string. The hard-coded default for port is 2203 or the contents of the environment variable DVTDSRPORT, if this is set.

Examples Examples of the server name are:

- `""`: all defaults, resolves to “127.0.0.1; 2203” (assuming environment variable DVTDSRPORT is not set)
- `"localhost"`: resolves to “localhost; 2203” (assuming environment variable DVTDSRPORT is not set)

www.valuetronics.com

- "; 7": override port, resolves to “127.0.0.1; 7”
- "15.100.10.20; 2208": fully specified host
- "mypc.bbn.hp.com; 2208": fully specified host

Call_HP81200

C Syntax

```
int HP81200_CDECL Call_HP81200(const char *Command,
                               char *Response,
                                 int *BufferSize)
```

VB Syntax

```
VB_CALL_HP81200(ByVal Command As String,
                ByRef Response As String,
                 ByRef BufferSize As Long) As Long
```

Description Sends a command to the Agilent 81250 server and gets the response.

Parameters

Command Any valid SCPI command, as described in *“Overview of the SCPI Command Structure” on page 31*. If the command is a query, the response and size of the response are returned.

Response If the command is a query, the response is returned in this buffer.

BufferSize Size of the buffer allocated to receive the query result. On return, the variable holds the actual number of valid bytes in the buffer.

Return Value Returns 0 on success and a negative number for server problems (use *“GetErrStr_HP81200” on page 23* to get an error message).

A positive number indicates that the command execution failed. Error messages can be read from the corresponding error queue (see *“Error Handling” on page 21*).

NOTE The parameter `Response` is a buffer area provided by the caller. The caller is responsible for reserving enough memory to hold the response.

To prevent buffer overruns, `BufferSize` defines the maximum number of bytes that can be written. This variable is set upon return to reflect the number of bytes actually written. If the response does not fit into the buffer, an error is returned.

www.valuetronics.com

GetErrStr_HP81200

C Syntax

```
int HP81200_CDECL GetErrStr_HP81200(int ErrCode,
                                    char *ErrMessage,
                                      int *BufferSize)
```

VB Syntax

```
VB_GETERRSTR_HP81200(ByVal ErrNum As Long,
                     ByRef ErrString As String,
                      ByRef BufferSize As Long) As Long
```

Description Gets the error string if one of the calls returns a negative value (see *"Error Handling" on page 21*). This usually happens if the connection to the server or the creation of a handle fails.

Parameters

ErrCode Error code returned by one of the other functions.

ErrMessage Buffer to receive a human-readable error string describing the error represented by `ErrCode`.

BufferSize Size of the buffer allocated to receive the error message (`ErrMessage`). On return, the variable holds the actual number of valid characters in the buffer.

NOTE The parameter `Response` is a buffer area provided by the caller. The caller is responsible for reserving enough memory to hold the response.

To prevent buffer overruns, `BufferSize` defines the maximum number of bytes that can be written. This variable is set upon return to reflect the number of bytes actually written. If the response does not fit into the buffer, an error is returned.

Disconnect_HP81200

C Syntax

```
int HP81200_CDECL Disconnect_HP81200()
```

VB Syntax

```
VB_DISCONNECT_HP81200() As Long
```

Description Disconnects from the Agilent 81250 server. This must be the last call when using the Agilent 81250 String Interface Library.

www.valuetronics.com

Example Implementations

The following examples show how these commands could be implemented in a C or VisualBasic environment.

C Example

The following code example connects to the firmware server on the defined machine at the defined port, queries the `:DVT:IDN?`, outputs the results, and disconnects.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <hp81200.h>

#define BUFF_SIZE 1024

int main(int argc, char* argv[])
{
   int err = 0;
   char buffer[BUFF_SIZE];
   int bufSize= BUFF_SIZE;
   char* host;

   // Define host
   switch(argc)
   {
      case 1:
         host = "";
         break;
      case 2:
         host = argv[1];
      default:
         host = argv[1];
         strncat(host, "; ", 2);
         strncat(host, argv[2], strlen(argv[2]));
         break;
   }

   // Connect to host
   err = Connect_HP81200(host);
   if (err=0)
   {
      printf("Could not connect to host\n");
      return 0;
   }

   // Send :DVT:IDN?
   bufSize= BUFF_SIZE;
```


www.valuetronics.com

```
   err = Call_HP81200(":DVT:IDN?", buffer, &bufSize);
   if (err!=0)
      printf("Connected to: %s\n", buffer);
   else
   {
      // Get and output the error.
      bufSize= BUFF_SIZE;
      err = GetErrStr_HP81200(err, buffer, &bufSize);
      printf("Error generated: %s", buffer);
   }

   // Disconnect
   err = Disconnect_HP81200();
   return 0;
}
```

VisualBasic Example

NOTE To implement the example, you have to import the file hp81200.bas into your project.

The following code connects to the entered server, sends SCPI commands, and outputs the results.

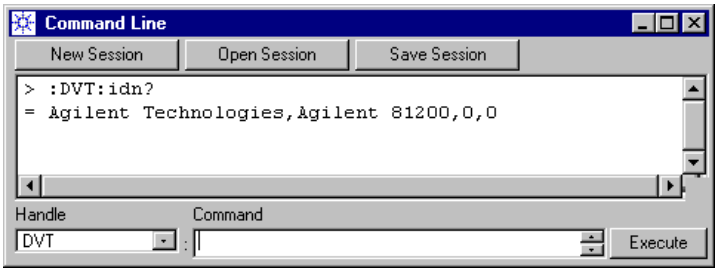
```
Sub myHP81200()
' hp81200.bas has to be imported into your project
Dim err As Long
Dim buffer As String * 256
Dim bufSize As Long
Dim Cmd As String
Dim ServerName As String

' Request a server
ServerName = InputBox("Enter server name:", "Connect to Server")
err = VB_CONNECT_HP81200(ServerName)
' If the server is not available, output the error.
If err <> 0 Then
   buffer = ""
   bufSize = 256
   VB_GETERRSTR_HP81200 err, buffer, bufSize
   MsgBox "Could not connect due to error: " & vbCrLf & _
          buffer
   Exit Sub
End If

' Request and output commands
Do
   buffer = ""
   bufSize = 256
   Cmd = InputBox("Enter command:", "SCPI Command Input")
```


www.valuetronics.com

```
   If Cmd = "" Then Exit Do
   err = VB_CALL_HP81200(Cmd, buffer, bufSize)
   ' Output the results ...
   If err = 0 Then
      MsgBox "Command " & Cmd & " succeeded." & vbCrLf & _
             vbCrLf & buffer
      ' ... or any errors
   Else
      VB_GETERRSTR_HP81200 err, buffer, bufSize
      MsgBox "Command " & Cmd & " failed." & vbCrLf & _
             vbCrLf & buffer
   End If
Loop
' Disconnect before closing
err = VB_DISCONNECT_HP81200
End Sub
```

Command Line Interface

The Agilent 81250 user interface features a Command Line window (*Command Line* in the *Go* menu) that allows you to enter SCPI commands and displays query results.



In this window, you can store and replay sequences of commands. This is very useful when implementing control programs, because you can test your SCPI commands and sequences before implementing them in your code.

www.valuetronics.com

Overview of the SCPI Command Structure

The device commands for the Agilent 81250 System follow the rules for SCPI Commands.

Some basic peculiarities of the command syntax used in this programming references are described in *"Command Syntax" on page 28*.

The overall structure of the commands is described in *"The Command Tree" on page 30*.

A basic and a more advanced programming example using the Agilent 81250 SCPI commands are given in *"Example Programs" on page 35*.

"SCPI Commands Overview" on page 41 provides application-oriented access to the available SCPI commands. The individual commands are described in more detail in *"Commands in the DVT Subsystem" on page 65* and *"Commands in DSR Application Subsystems" on page 75*.

NOTE In this programming reference, it is assumed that you have a basic knowledge of the SCPI language. For an introduction to SCPI and SCPI programming techniques, refer to:

- The SCPI Consortium: IEEE488.2 Standard Commands for Programmable Instruments, published periodically by various publishers. To obtain a copy of this manual, contact your Agilent representative.

www.valuetronics.com

Command Syntax

The Agilent 81250 System does *not* support IEEE 488.2 Common Commands. The device commands for the Agilent 81250 System follow the rules for SCPI Commands.

The name of a command consists of one or more *keywords*. A keyword can have a long or a short form. The short form is denoted in upper case letters.

The following sections highlight some important aspects of the Agilent 81250 command syntax.

Hierarchical Structure

The language used is based on a hierarchical structure (or tree system). Associated commands are grouped together under a common node in the hierarchy. Associated branches and commands are grouped into higher level branches until they meet at the root of the command tree. To obtain a particular command, the full path to it must be specified. This path is constructed by appending keywords to the path as the command tree is descended. Keywords are separated by colons (:). A typical example is:

```
:DVT:INST:LIST?
```

Conventions Used in the Command Reference

The following conventions are used when describing commands and their syntax:

- The command syntax shows the commands as a mixture of upper and lower case letters. The upper case letters indicate the abbreviated spelling for the command. For shorter program lines, you only need to send the abbreviated form. For better program readability, you can send the entire command.
- Square brackets ([ ]) are used to enclose a keyword that is optional when using the command. Such a node is called a default node.

www.valuetronics.com

- Numeric suffixes are represented by (*) in command descriptions.
- Queries are denoted by a question mark (?) at the end of the header.

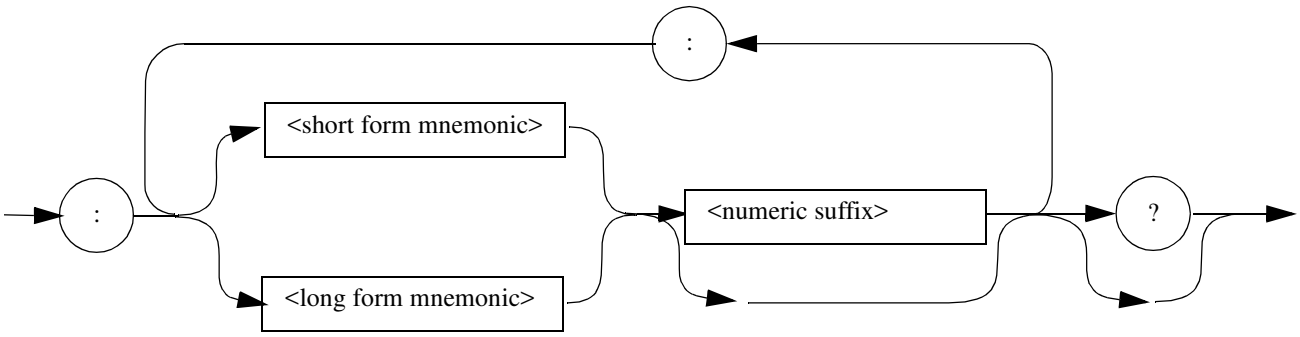
NOTE The instrument accepts either the abbreviated command form or the entire command form, but not any other forms.

The commands are not case sensitive, so upper or lower case letters or a mix of them are acceptable.

Instrument Commands and Queries Syntax

The instrument command syntax is as follows:

```
<instrument command> := :<short form mnemonic> [<numeric suffix>]
                        :<long form mnemonic> [<numeric_suffix>]
```



Queries are denoted by a command header with an appended question mark.

```
<instrument command> :=
                     :<short form mnemonic> [<numeric suffix>] '?'
                     :<long form mnemonic>  [<numeric_suffix>] '?'
```

Linking Commands

It is possible to link multiple device commands.

To link multiple device commands, use a semicolon and a colon between the commands. Because the Agilent 81250 System allows to configure the available hardware resources as virtual instruments, it is necessary that each individual command sent to the system specifies also the virtual instrument that has to execute the command. This is done by a leading node, the <Handle>. When linking commands, leading nodes can be omitted, as long as they are identical.

www.valuetronics.com

Example The following commands could be linked:

```
:_TEST:cgr1:mod2:conn4:puls:del 6e-9
:_TEST:cgr1:mod2:conn4:puls:width 20e-9
:_TEST:cgr1:mod2:conn4:puls:tran:lead 5e-9
:_TEST:cgr1:mod2:conn4:puls:tran:trail 2e-9
```

The linked form would be:

```
:_TEST:cgr1:mod2:conn4:puls:del 6e-9;width 20e-9;
tran:lead 5e-9;trail 2e-9
```

The Command Tree

The Agilent 81250 SCPI command interface consists of a tree of commands. The branches and sub-branches of this tree are referred to as subsystems. Each subsystem contains a group of related commands.

At the root of the command tree, there are two types of branches (subsystems):

- the DVT subsystem (always available)
- virtual instrument subsystems identified by the handle associated to the instrument

Both are described in *"DVT Subsystem and Virtual Instruments" on page 31*.

Many parameters can be specified on port, terminal and connector level. These aspects are discussed in *"Multiple Parameter Access" on page 32*.

www.valuetronics.com

DVT Subsystem and Virtual Instruments

At first—after connecting to the instrument—there is only one subsystem available: the DVT subsystem. The commands in this subsystem provide basic administration features.

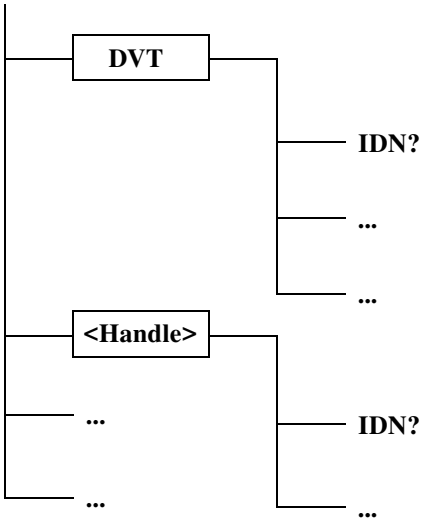
The major administrative task is to manage so-called **virtual instruments**. A virtual instrument is a combination of

- a system resource as defined in the dvtsys.txt file
- a software application

  With this release of the Agilent 81250, the only application available is the DSR application (Digital Stimulus and Response). Future releases might provide more applications, for example BERT (Bit Error ratio Test).

Virtual instruments can be controlled as if they were separate instruments.

Each virtual instrument is identified by a **handle**. As shown in the following picture, this handle forms the root for a new branch (subsystem) in the command tree. The commands available within this subsystem depend on the selected software application.



See *"Typical SCPI Programming Sequence" on page 36* for an example.

NOTE Each virtual instrument has its own error/event queue.

The <Handle>:SYSTem:ERRor? query (see *":<Handle>:SYSTem:ERRor?" on page 77*) is used to retrieve the next error in the error queue.

www.valuetronics.com

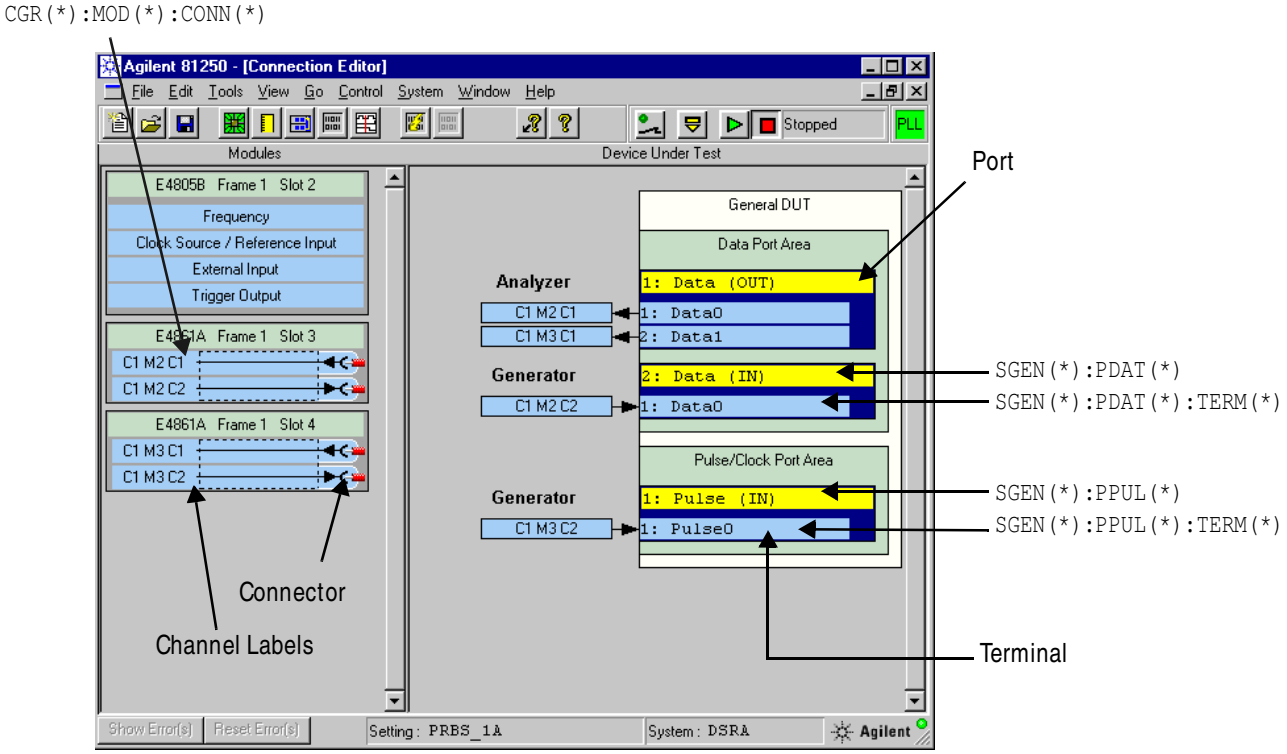
Multiple Parameter Access

The command tree provides comfortable means of accessing certain parameters of more than one terminal at a time. According to the system architecture, there are

- **terminals** (DUT input or output pins)
- **ports** (groups of DUT input or output pins with identical or similar properties, such as a data bus)
- **connectors** (input or output connectors of a frontend)

So, for example, if you have a data bus consisting of several terminals, you can group them together in a port, and then set their timing parameters with only one command.

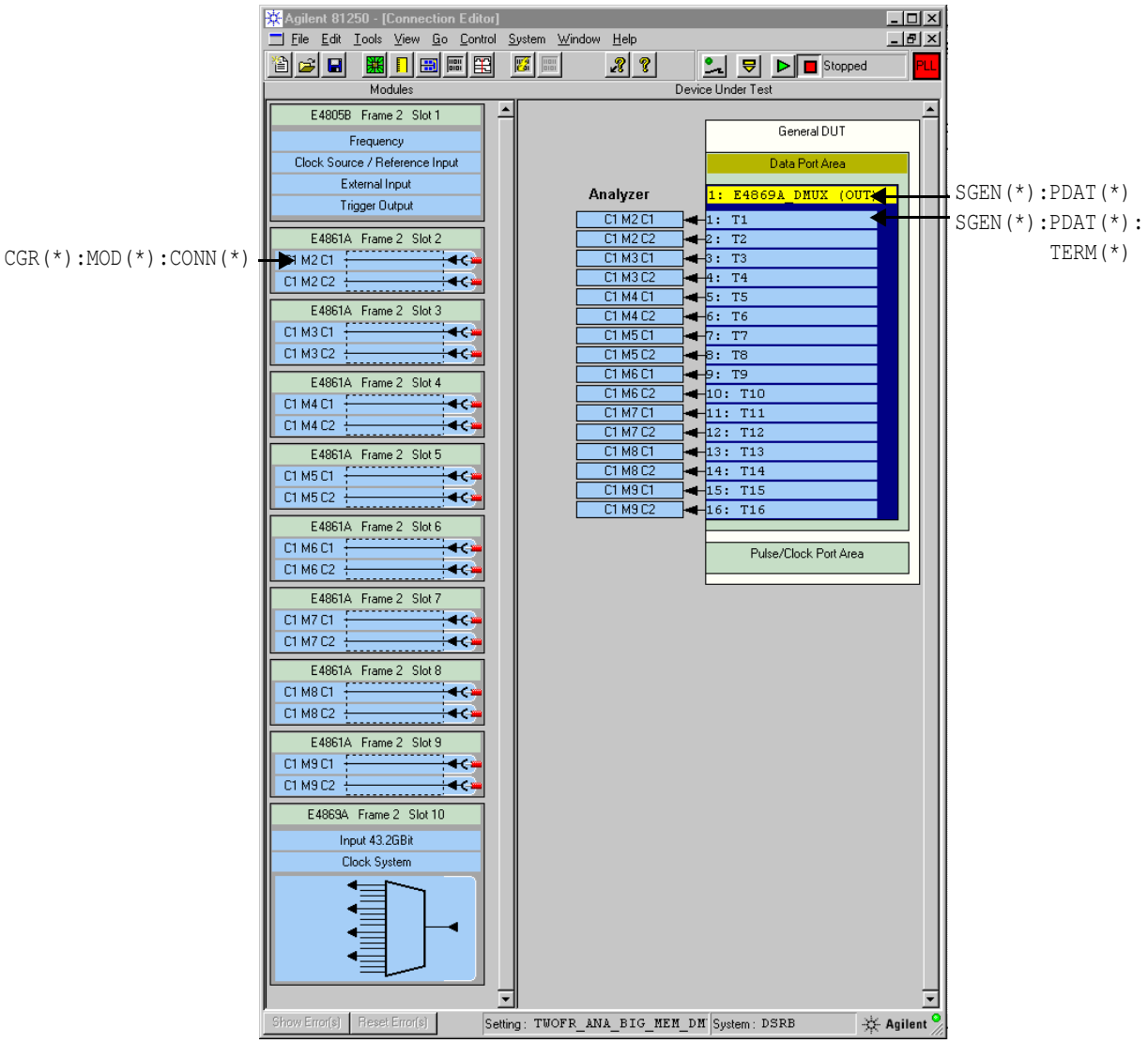
This is shown in the following figure:



www.valuetronics.com

Typical 43.2Gbit Analyzer Commands

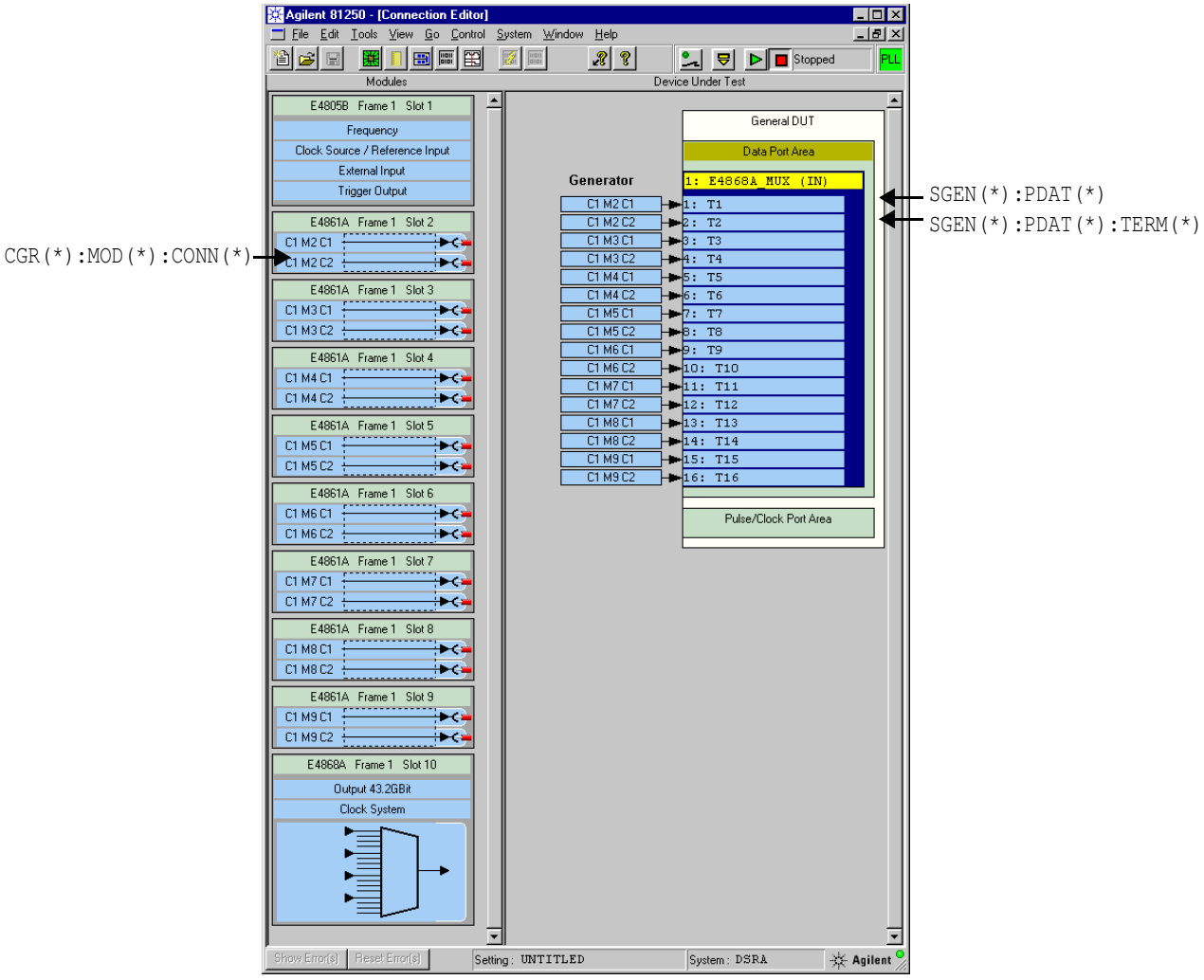
The following figure shows some typical commands and their corresponding elements for a 43.2Gbit data analyzer system in the Agilent 81250 user interface:



www.valuetronics.com

Typical 43.2Gbit Generator Commands

The following figure shows some typical commands and their corresponding elements for a 43.2Gbit data generator system in the Agilent 81250 user interface:



Data ports and pulse ports

These figures also show how you can distinguish between data ports (:PDATa(*):) and pulse ports (:PPULse(*):) on command level.

www.valuetronics.com

Types of Parameters

The following types of parameters can be accessed on terminal, port, and connector level:

- timing parameters
- level parameters
- output parameters
- input parameters
- format parameters

The corresponding commands are available on each level. In this reference they are described only once, summarized according to the parameter type. The same principle is used for some port, terminal and connector administration commands and for error handling commands.

Example

For example, to set the delay for all terminals in a port (for example, pins in a data bus), you can

- either address each terminal individually, using
  :SGENeral:PDATa(*):TERMinal(*):PULSe:DELay
- or set the delay for the complete port, using
  :SGENeral:PDATa(*):PULSe:DELay
  Both commands are described in *":PULSe:DELay" on page 179*.

Example Programs

Two example programs are discussed in the following:

- *"Typical SCPI Programming Sequence" on page 36* shows a typical sequence of SCPI commands used for programming the Agilent 81250.
- *"Example C++ Program" on page 38* guides you through an example program and a class library supplied with the Agilent 81250.

www.valuetronics.com

Typical SCPI Programming Sequence

When implementing programs for the Agilent 81250 using SCPI commands, you should keep to the following sequence:

1. Create instrument
2. Set system parameters
3. Create model of DUT
4. Connect DUT to instrument
5. Set signal parameters
6. Create sequence
7. Run test
8. Destroy instrument

NOTE Always make sure that you destroy the handle at the end of your program.

Here is a little example program showing some of these steps.

```
:DVT:INST:RES:LIST? "DSR"     # Ask what DSR instruments are
                              # pre-configured in the
                              # System (see
                              # d:\81200\cfg\dvtsys.txt file)
                              # Returns: "DSRA","DSRB"


                              # Create a handle, better ask for a
                              # handle: give TEST as a suggestion
                              # for the handle:
:DVT:INST:HAND:CRE? TEST, "DSR", "DSRA"
                              # When the suggested handle is not
                              # used, then the system will return:
                              # _TEST
:_TEST:SGEN:GLOB:PER 50e-9    # set period:
```


www.valuetronics.com

```
:_TEST:SGEN:GLOB:MUX 1        # set system BLG factor, check that
                              # period and memory depth match:
                              # set an input data port named
                              # AddrBus with 4 Terminals:
:_TEST:SGEN:PDAT:APP "INPUT_PORT", 4, "AddrBus"

                              # add a 5th terminal with name
                              Addr4:
:_TEST:SGEN:PDAT1:TERM5:APP "Addr4", 5

                              # connect terminal 1 of port AddrBus
                              # to connector 1 of module 2:
:_TEST:SGEN:CONN:PDAT1:TERM1:TO (@0102001)

                              #set pulse width of terminal 1 (T1):
:_TEST:SGEN:PDAT1:TERM1:PULS:WIDT 20e-9

                              # set delay of T1:
:_TEST:SGEN:PDAT1:TERM1:PULS:DEL 5e-9

                              # set leading edge of T1:
:_TEST:SGEN:PDAT1:TERM1:PULS:TRANS 1e-9

                              # set high level of T1:
                              # High level has to be higher than
                              # the actual low level.
:_TEST:SGEN:PDAT1:TERM1:VOLT:HIGH 3

                              # set low level of T1:
:_TEST:SGEN:PDAT1:TERM1:VOLT:LOW 1

                              # set load impedance for T1:
:_TEST:SGEN:PDAT1:TERM1:OUTP:IMP:EXT 50

:DVT:INST:HAND:DEST _TEST     # When the virtual device is no
                              # longer required, the handle can be
                              # eliminated:
```


www.valuetronics.com

Example C+ + Program

This example is intended to show the more sophisticated steps of analyzing the system configuration and setting up the system as required, importing segment data, and running a measurement in "compare and acquire around error" mode. Furthermore, the Edit subsystem is used to handle the captured data.

The example program is divided into *"The main() Procedure" on page 39* and *"The doIt() Procedure" on page 39*. The program uses a small interface class (see *"The lib.cpp Interface Class Library" on page 38*) that provides methods for initializing the system, dealing with handles, executing the SCPI commands, and logging errors.

By default, the files with the example code (main.cpp, lib.cpp, and lib.h) can be found in the Agilent 81250 installation directory:

```
C:\Program Files\Agilent\Agilent 81200\samples\ecap
```

The lib.cpp Interface Class Library

The file lib.cpp defines the HP 81200 class, which is an interface class to the Agilent 81250 system. The class itself uses the *"Agilent 81250 String Interface Library" on page 20*.

For a complete listing, refer to *"Lib.cpp Interface Class Library Code" on page 331*.

The class provides the following public methods:

- `Init()`

  The `Init()` method connects to an Agilent 81250 server on the local machine or to a specified server. It creates a handle for the specified system. This handle is only used within the class. An extra parameter specifies a file to be used for error logging.

- `Exit()`

  The `Exit()` method releases the handle and disconnects from the system.

- `Call()`

  There are two `Call()` methods, distinguished by the number of parameters: one to execute SCPI commands, one to execute SCPI queries.

  Both methods return "false" in case of an error. The query method features an additional parameter to return the result of the query.

www.valuetronics.com

The main() Procedure

The file main.cpp contains the application-specific code. It uses the lib.cpp interface class to communicate with the Agilent 81250 system.

For a complete listing, refer to *"Main.cpp Application Code" on page 336*.

The `main()` procedure implements the following steps:

- Creates an HP 81200 object and calls the `Init()` method to create a handle and to connect to the system. stdout is used for error logging.
- Invokes the `doIt()` procedure, which prepares and executes all required SCPI commands.
- Calls the `Exit()` method to release the handle and to disconnect from the system.

The doIt() Procedure

The code for the `doIt()` procedure is part of the file main.cpp. It prepares and executes all SCPI commands required for this example.

For a complete listing, refer to *"Main.cpp Application Code" on page 336*.

These are the most important steps performed by the `doIt()` procedure:

- Stops and resets the system.
- Analyzes the system configuration.

  To create a list of all analyzers and generators, the numbers of clock groups, modules within each clock group, and connectors within each module are determined.

  In loops over all clock groups, modules, and connectors, the type of each connector is determined. The connectors are appended to the list of analyzers or to the list of exercisers.

www.valuetronics.com

- Connects generators and analyzers to one port each.

  The number of generators (aGenCnt) and the list of generators (aGenerators) are used to create an input port with the required number of terminals. The terminals are connected to the generator channels, starting at terminal 1.

  Number and list of analyzers (aAnaCnt and aAnalyzers) are used to create and connect output port and terminals.

  This step only makes sense for this example. In real life you know the number of terminals and ports required by the DUT and set up the Agilent 81250 accordingly.

- Switches on all generator and analyzer ports, and applies levels and thresholds.
- Imports segment data from the file walk64.txt.
- Sets the period.
- Sets up a sequence that uses the imported segment.

  First the number of available loop levels is determined (depending on the clock module used).

  The highest loop level is used to set up an infinite loop. The imported segment is used as data for the sequence.

  The start of the sequencer is used to generate a trigger signal at the trigger output.

- Sets measurement mode to “compare and acquire around error”
- Saves the setting for later use.
- Starts the measurement.
- Waits until the measurement is done (only works online).
- Stops the measurement.
- Exports the captured data and the error memory.

  This demonstrates how the :EDIT:SEGMent(*) subsystem can be used to handle captured data.

  First, the segments are opened and information on the coding used for the captured data is queried. Then the captured data are printed trace by trace.

  Finally, the segments are saved, exported, and closed.

www.valuetronics.com

SCPI Commands Overview

This section provides an application-oriented overview of the SCPI commands available in the DVT subsystem and in the subsystems for DSR applications.

The following categories are available:

www.valuetronics.com

Generic Administration Commands

Command (Link to detailed description)	Description
":DVT:IDN?" on page 67	Returns the device identity.
":DVT:SYSTem:ERRor?" on page 68	Returns the first entry of the error queue.
":DVT:INSTrument:LIST?" on page 69	Returns a comma-separated list of items containing the names of all applications.
":DVT:INSTrument:RESource:LIST?" on page 69	Returns a list of available systems names. A system name can be used to create an association between a set of modules (system) and an application software to build an virtual instrument.
":DVT:INSTrument:HANDle:CREate?" on page 70	Creates a handle for a new virtual instrument.
":DVT:INSTrument:HANDle:LIST?" on page 71	Returns a list of all virtual instrument handles.
":DVT:INSTrument:HANDle:DESTroy" on page 71	Destroys the handle for a virtual instrument and frees all resources.

Virtual Instruments Administration Commands

Command (Link to detailed description)	Description
":<Handle>:IDN?" on page 76	Returns the device identity.
":<Handle>:OPC?" on page 76	The query returns when all modules report that they are ready.
":<Handle>:SYSTem:ERRor?" on page 77	Returns the first entry of the error queue.
":<Handle>:SYSTem:CINFormation?" on page 78	Returns the configuration information of an Agilent 81250 system.
":<Handle>:SYSTem:CLIent[:HANDle]?" on page 82	Returns the name of the virtual instrument currently connected to the server.
":<Handle>:SYSTem:CLIent:LOCK" on page 82	Used to take over the control of the virtual instrument explicitly.
":<Handle>:SYSTem:CLIent:BLOCk" on page 83	Used to take over the control of the virtual instrument explicitly. Other clients will be blocked until the instrument is unlocked.
":<Handle>:SYSTem:CLIent:UNLock" on page 83	Used to "unlock" or to "set free" the virtual instrument.
":<Handle>:CONFiguration:CGRoups(*)?" on page 84	Returns the number of clock groups that are available in the system.
":<Handle>:CONFiguration:CGRoups(*):MODules(*)?" on page 84	Returns the number of modules that are available in a clock group.

www.valuetronics.com

Command (Link to detailed description)	Description
":<Handle>:CONFiguration:CGRoups(*):MODules(*):CONNectors(*)?" on page 85	Returns the number of connectors of a module.
":<Handle>:CONFiguration:CGRoups(*):MODules(*):CONNectors(*):TYPE?" on page 85	Returns the type of the connector.
":<Handle>[:CGRoup(*)]:CINFormation?" on page 122	Returns the clock group configuration information.
":<Handle>[:CGRoup(*)]:MODule(*):CINFormation?" on page 124	Returns the module configuration information.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CINFormation?" on page 127	Returns the connector configuration information.
":<Handle>:CONFiguration:STYPes?" on page 85	Returns a list of Scheme TYPes supported by the Agilent 81250 System.
":<Handle>:CONFiguration:PROFile[:VALue]" on page 86	Sets a profile parameter to a specified value.
":<Handle>:CONFiguration:PROFile[:VALue]?" on page 86	Returns an expression containing the value of a profile parameter.
":<Handle>:CONFiguration:PROFile:REMove" on page 86	Removes the specified parameter from the profile.
":<Handle>:CONFiguration:PROFile:LIST?" on page 87	Lists the parameters contained in the profile.
":<Handle>:SYSTem:TEST:MODule" on page 79	Starts a module selftest. The results of this test are stored in the message queue.
":<Handle>:SYSTem:TEST:GLOBal" on page 80	Starts a system-wide selftest. The results of this test are stored in the message queue.
":<Handle>:MODule:SREVisions" on page 80	Checks the actual BIOS Software revisions against the expected revision located in the firmware.
":<Handle>:SYSTem:MQUeue[:READ]?" on page 81	Returns the first item contained in the message queue.
":<Handle>:SYSTem:MQUeue:LENGth?" on page 81	Returns the length of the message queue.
":<Handle>:SYSTem:PON[:STATus]" on page 81	Moves the messages, which are generated by the modules at system start up, to the message queue.
":<Handle>:SGENeral:INFormation:PCLasses?" on page 138	Returns the list of implemented port classes in a comma separated quoted string list.

www.valuetronics.com

Mass Memory Commands

Command (Link to detailed description)	Description
":<Handle>:MMEMory:LIST?" on page 88	Returns an expression containing a comma separated list of quoted strings. Each string contains one element found in the directory <"Absolute Path">.
":<Handle>:MMEMory:INFormation?" on page 89	Returns detailed information about the specified object.
":<Handle>:MMEMory:SEGMent:LOAD" on page 90	Loads new segments into or updates segments in the Agilent 81250 System's database
":<Handle>:MMEMory:SEGMent:SAVE" on page 91	Stores segments to file.
":<Handle>:MMEMory:SEGMent:GET?" on page 92	Returns an expression that contains the contents of segment(s) (vector definition).
":<Handle>:MMEMory:SETTing:NAME?" on page 92	Returns the current setting name.
":<Handle>:MMEMory:SETTing:LOAD" on page 93	Loads the current setting or a specified setting into the Agilent 81250 System's database.
":<Handle>:MMEMory:SETTing:SAVE" on page 93	Saves changes either into the current setting or into a new setting.
":<Handle>:MMEMory:SETTing:NEW" on page 94	Performs a reset and creates an "untitled" segment.
":<Handle>:MMEMory:SETTing:DELete" on page 94	Deletes a specified setting from the Agilent 81250 System's database.
":DVT:MMEMory:SETTing:IMPort" on page 73	Allows to import a setting into the Agilent 81250 System.
":SETTing:IMPort:DVT:MMEMory:SETTing:EXPort" on page 72	Exports a setting into a newly created file.
":DVT:MMEMory:SETTing:EXPort?" on page 72	Returns the setting data that would normally be written to a file.

Segment Editing Commands

Command (Link to detailed description)	Description
":<Handle>:EDIT:SEGMent(*):CREate?" on page 98	Creates a new segment.
":<Handle>:EDIT:SEGMent:OPEN?" on page 96	Opens the specified segment.
":<Handle>:EDIT:SEGMent(*):SAVE" on page 97	Saves the changes made to the current segment name, or creates a new segment name.
":<Handle>:EDIT:SEGMent(*):CLOSe" on page 101	Closes the segment editor specified by the segment inspector index number without saving possible changes made to the current segment name.
":<Handle>:EDIT:SEGMent(*):DELete" on page 98	Deletes the specified segment.
":<Handle>:EDIT:SEGMent(*):EXISts?" on page 99	Checks whether the specified segment name is located in the Agilent 81250 System's database.
":<Handle>:EDIT:SEGMent(*):RPATh?" on page 100	Returns the absolute path for one segment.
":<Handle>:EDIT:SEGMent(*):PATTern:CODing?" on page 105	Returns the segments coding.
":<Handle>:EDIT:SEGMent(*):PATTern:CODing" on page 104	Sets a segment's coding.
":<Handle>:EDIT:SEGMent(*):PATTern:ERRor:FIRSt?" on page 105	Returns the coordinates (trace, vector) of the first found error.

www.valuetronics.com

Command (Link to detailed description)	Description
":<Handle>:EDIT:SEGMent(*):PATTern:ERRor:NEXT?" on page 106	Returns the coordinates (trace, vector) of the first found error, starting at the given coordinates.
":<Handle>:EDIT:SEGMent(*):PATTern:ERRor:PREVious?" on page 106	Returns the coordinates (trace, vector) of the first found error, starting at the given coordinates, searching backwards.
":<Handle>:EDIT:SEGMent(*):PATTern:FIND:FIRSt?" on page 107	Returns the coordinates (trace, vector) of the first found instance of the given pattern.
":<Handle>:EDIT:SEGMent(*):PATTern:FIND:NEXT?" on page 109	Returns the coordinates (trace, vector) of the first found instance of the given pattern, starting at the given coordinates.
":<Handle>:EDIT:SEGMent(*):PATTern:FIND:PREVious?" on page 109	Returns the coordinates (trace, vector) of the first found instance of the given pattern, starting at the given coordinates, searching backwards.
":<Handle>:EDIT:SEGMent(*):PATTern:DATA" on page 101	Creates a state range and fills it with the required pattern.
":<Handle>:EDIT:SEGMent(*):PATTern:DATA?" on page 102	Returns from the segment inspector the vector stream bounded by the specified rectangle.
":<Handle>:EDIT:SEGMent(*):PATTern:LENGth?" on page 110	Returns the numbers of vectors of a segment.
":<Handle>:EDIT:SEGMent(*):PATTern:WIDTh?" on page 110	Returns the numbers of traces of a segment.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:COPY" on page 111	Copies the bounded rectangle of data from the segment into the clipboard.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:PASTE" on page 111	Pastes the data currently stored in the clipboard to the specified segment.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:FILL" on page 112	Sets the bounded area of the segment with the same state value.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:INVert" on page 113	Inverts the states information in the bounded area.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:MIRRor" on page 114	Rotates the specified block about horizontal (VECTor) or vertical axis (TRACe).
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:INSert" on page 115	Inserts new traces or vectors into a segment.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:DELete" on page 115	Deletes traces or vectors from a segment.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:CONVerse" on page 116	Changes the current coding for the segment and converts existing data in the segment according to the specified mapping
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:SERialize" on page 117	Serializes a parallel data stream.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:DESerialize" on page 118	Deserializes a serial data stream.
":<Handle>:EDIT:SEGMent(*):PARAmeter:LENGth?" on page 118	Gets the length of the list of parameters available for this segment.
":<Handle>:EDIT:SEGMent(*):PARAmeter:LIST?" on page 119	Gets the list of parameters available for this segment.
":<Handle>:EDIT:SEGMent(*):PARAmeter[:VALue]?" on page 119	Returns the value for a specified parameter from the segment.
":<Handle>:EDIT:SEGMent(*):PARAmeter[:VALue]" on page 120	Sets the value for a specified parameter in the segment.
":<Handle>:EDIT:SEGMent(*):PARAmeter:REMove" on page 120	Removes a parameter from the list of parameters in the segment.
":<Handle>:EDIT:SEGMent(*):TYPE" on page 120	Sets the segment type for the current segment.
":<Handle>:EDIT:SEGMent(*):TYPE?" on page 121	Returns the segment type of the segment specified by the segment inspector index number.

www.valuetronics.com

Global Commands

Command (Link to detailed description)	Description
":<Handle>:SGENeral:GLOBal:INITiate:CONTinuous" on page 150	Starts/ stops the system.
":<Handle>:SGENeral:GLOBal:INITiate:CONTinuous?" on page 150	Returns the current state of the system.
":<Handle>:SGENeral:GLOBal:SYSTem:STATe?" on page 151	Returns the current state of the system.
":<Handle>:SGENeral:GLOBal:CALibration:SELF DELay" on page 140	Autocalibrates the system after any frontend have been exchanged.
":<Handle>:SGENeral:GLOBal:PERiod" on page 141	Sets the period of the Agilent 81250 System.
":<Handle>:SGENeral:GLOBal:PERiod?" on page 141	Returns the period of the Agilent 81250 System.
":<Handle>:SGENeral:GLOBal:FREQuency" on page 142	Sets the frequency of the Agilent 81250 System.
":<Handle>:SGENeral:GLOBal:FREQuency?" on page 142	Returns the frequency of the Agilent 81250 System.
":<Handle>:SGENeral:GLOBal:MUX" on page 143	Defines the global settings for the Segment Resolution.
":<Handle>:SGENeral:GLOBal:MUX?" on page 143	Returns the current MUX factor (Segment Resolution).
":<Handle>:SGENeral:GLOBal:DOFFset" on page 140	Specifies an offset value to the fixed delay for all connectors.
":<Handle>:SGENeral:GLOBal:DOFFset?" on page 140	Returns the currently set delay offset.
":<Handle>:SGENeral:GLOBal:CONNect" on page 139	Connects or disconnects all enabled connectors of the Agilent 81250 System.
":<Handle>:SGENeral:GLOBal:CONNect?" on page 139	Returns the actual connection status of the enabled connectors.
":<Handle>:SGENeral:GLOBal:FETCh:ERRor:ANY?" on page 141	Reports whether there has been found any error.

www.valuetronics.com

Clock Reference Input Commands

Command (Link to detailed description)	Description
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer][:SOURce]" on page 168	Sets the clock reference or the external clock mode.
":<Handle>:SGENeral:GLOBal:TRIGger?" on page 169	Returns the actual status of the Clock/ Reference Input of the Central resource.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:RCLock:DETect" on page 171	Measures the external clock reference and sets the corresponding mode automatically.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLock:MEASurement" on page 171	Measures the external clock and sets the corresponding mode EXTernal automatically.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:TVOLtage" on page 171	Specifies the termination voltage.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:TVOLtage?" on page 172	Returns the actual setting of the termination voltage.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLOCk[:VALue]?" on page 169	Measures and returns the supplied external clock at the clock input of the E4805B or E4808A central module.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLOCk:MULTiplier" on page 169	Sets the clock multiplier.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLOCk:MULTiplier?" on page 170	Returns the current clock multiplication factor.

External Input Commands

Command (Link to detailed description)	Description
":<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer][:SOURce]" on page 174	Controls the start mode of the Agilent 81250 System.
":<Handle>:SGENeral:GLOBal:ARM?" on page 175	Returns the actual status of the External Input connector at the front of the Central resource.
":<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:SENSe" on page 175	Specifies the input level condition which is used in the GATed | STARTsignal | STOPsignal modes.
":<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:SENSe?" on page 175	Returns the actual setting of the input level condition for the gate or start/ stop signal.
":<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:THReshold" on page 176	Specifies the threshold of the external input signal which is used in the GATed | STARTsignal | STOPsignal modes.
":<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:THReshold?" on page 176	Returns the actual threshold for the external input signal used in the gate or start/ stop mode.
":<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:TVOLtage" on page 176	Specifies the termination voltage of the external input signal which is used in the GATed | STARTsignal | STOPsignal modes.
":<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:TVOLtage?" on page 177	Returns the actual termination voltage for the external input signal used in the gate or start/ stop mode.

www.valuetronics.com

Trigger Output Commands

Command (Link to detailed description)	Description
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:DELay" on page 132	Varies the delay of the trigger output signal.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:DELay?" on page 133	Returns the current delay of the trigger output signal.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MUX" on page 133	Sets the frequency multiplier factor for the individual connector.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MUX?" on page 133	Returns the MUX factor associated with the trigger output.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:HIGH" on page 134	Sets the high voltage level of the trigger output signal.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:HIGH?" on page 134	Returns the high voltage level of the trigger output signal.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:LOW" on page 134	Sets the low voltage level of the trigger output signal.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:LOW?" on page 135	Returns the low voltage level of the trigger output signal.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:TVOLtage" on page 135	Specifies the termination voltage of a trigger output connector.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:TVOLtage?" on page 135	Returns the current termination voltage of the trigger output connector.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:IMPedance:EXTernal" on page 135	Specifies the external termination IMPedance.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:IMPedance:EXTernal?" on page 136	Returns the current programmed external termination impedance (load impedance).
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MODE" on page 136	Select the source of the trigger output.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MODE?" on page 136	Returns the actual source used for the trigger signal.

www.valuetronics.com

Port Administration Commands

The following table lists the commands available for the administration of pulse and data ports.

Command (Link to detailed description)	Description
":<Handle>:SGENeral:INFormation:PCLasses?" on page 138	Returns the list of implemented port classes in a comma separated quoted string list.
":APPend" on page 282	Creates and appends a new port to the list of ports.
":LIST?" on page 283	Returns a comma-separated list of port names.
":ATYPes?" on page 283	Returns a comma-separated list of available predefined port TYPes.
":DELete" on page 283	Deletes the port specified by the suffix of PPULse.
":REName" on page 284	Renames the port specified by the suffix of PPULse.
":NAME?" on page 284	Returns the name of the specified port.
":TYPE?" on page 284	Returns the type of the specified port.
":MUX" on page 196	Sets the frequency multiply factor (MUX factor) for the specified port.
":MUX?" on page 196	Returns the current frequency multiply factor (MUX factor) for the specified port.
":OUTPut:IMPedance:EXTernal" on page 263	Sets the external termination impedance
":OUTPut:IMPedance:EXTernal?" on page 263	Returns the currently programmed value for the termination (load) impedance for the connectors of the specified port.
":CALibration:CDELay" on page 285	Sets a cable delay for the specified port, to synchronize the signals at the DUT terminals.
":CALibration:CDELay?" on page 285	Returns the current cable delay for the specified port.
":SPECies?" on page 285	Returns the species (electrical or optical) of the specified port.

www.valuetronics.com

Terminal Administration Commands

The following table lists the commands available for administration of pulse and data port terminals.

Command (Link to detailed description)	Description
":APPend" on page 286	Creates and appends a new terminal to the list of terminals.
":LIST?" on page 286	Returns a comma-separated list of terminal names.
":DELete" on page 287	Deletes the terminal specified by the suffix of TERMinal.
":REName" on page 287	Renames the terminal specified by the suffix of TERMinal.
":NAME?" on page 287	Returns the name of the specified terminal.
":TYPE?" on page 288	Returns the type of the specified terminal.
":MOVE" on page 288	Moves the specified terminal to a new position.
":CALibration:CDELay" on page 289	Sets a cable delay for the specified terminal in the specified port to synchronize the signals at the DUT terminals.
":CALibration:CDELay?" on page 289	Returns the current cable delay for the specified terminal in the specified port.

www.valuetronics.com

Connector Administration Commands

Command (Link to detailed description)	Description
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CINFormation?" on page 127	Returns the connector configuration information.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):TYPE?" on page 128	Returns the product number of the frontend to which the connector belongs.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):NAME?" on page 128	Returns the name of this connector.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):TNAMe?" on page 128	Returns the Terminal NAMe to which this connector is connected.
":REMove" on page 290	Disconnects all terminals within the specified port or terminal.
":TERMinal(*):[TO]" on page 291	Connects the specified terminal to the specified channel or channels.
":TERMinal(*):[TO]?" on page 292	Returns the channel to which the specified terminal is connected.
":MUX" on page 196	Sets the MUX factor associated with this connector.
":MUX?" on page 196	Returns the MUX factor associated with this connector.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:CDELay" on page 129	Sets a cable delay for the specified connector to synchronize it with other signal applied to the DUT terminals.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:CDELay?" on page 129	Returns the current cable delay for the specified connector of a specific module.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:ZDELay" on page 129	Sets a zero delay for the specified connector of a specific the Agilent 81250 System's module.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:ZDELay?" on page 130	Returns the current zero delay for the specified connector.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):SPECies[:VALue]?" on page 130	Returns a comma-separated list of the species states that are supported by the connector.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):SPECies:STATe" on page 130	Switches the connector into either electrical (ELEC) or optical (OPT) mode.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):SPECies:STATe?" on page 131	Returns the current species of the connector.
":OUTPut:IMPedance:EXTernal" on page 263	Specifies the external termination impedance.
":OUTPut:IMPedance:EXTernal?" on page 263	Returns the currently programmed external termination impedance (load impedance).

www.valuetronics.com

Clockgroup Administration Commands

Command (Link to detailed description)	Description
":<Handle>[:CGRoup(*)]:CINFormation?" on page 122	Returns the clock group configuration information.
":<Handle>[:CGRoup(*)]:MCLock:SOURce[:VALue]" on page 123	Controls which of the clock modules generates the "master" clock.
":<Handle>[:CGRoup(*)]:MCLock:SOURce[:VALue]?" on page 123	Returns the state of the clock module specified.
":<Handle>[:CGRoup(*)]:MODule(*):CINFormation?" on page 124	Returns the module configuration information.
":<Handle>[:CGRoup(*)]:MODule(*):TYPE?" on page 125	Returns the product number in a quoted string.
":<Handle>[:CGRoup(*)]:MODule(*):SLOT?" on page 125	Returns the slot number in which the module is located.
":<Handle>[:CGRoup(*)]:MODule(*):FRAMe?" on page 125	Returns the frame number to which this module belongs.
":<Handle>[:CGRoup(*)]:MODule(*):NAME?" on page 125	Returns the name of this module.
":<Handle>[:CGRoup(*)]:MODule(*):CNAMes?" on page 126	Returns a quoted list of Connector names.

Sequence Commands

Command (Link to detailed description)	Description
":<Handle>:SGENeral:GLOBal:SEQuence[:VALue]" on page 153	Loads a data sequence in the form of an expression into the system.
":<Handle>:SGENeral:GLOBal:SEQuence[:VALue]?" on page 157	Returns the current data sequence of the system.
":<Handle>:SGENeral:GLOBal:SEQuence:EVENTs" on page 158	Defines events for the system.
":<Handle>:SGENeral:GLOBal:SEQuence:EVENTs?" on page 160	Returns a list of events defined for the system.
":<Handle>:SGENeral:GLOBal:SEQuence:FORCe" on page 161	Downloads the actual sequence immediately.
":<Handle>:SGENeral:GLOBal:SEQuence:LLEVel?" on page 161	Returns the actually available loop levels.
":<Handle>:SGENeral:GLOBal:SEQuence:PCONtrol" on page 162	Sets the value of the program control event.
":<Handle>:SGENeral:GLOBal:SEQuence:PCONtrol?" on page 162	Returns the value of the program control event.

www.valuetronics.com

Synchronization Commands

Command (Link to detailed description)	Description
":<Handle>:SGENeral:GLOBal:SYNChronization:USED?" on page 164	Returns whether a synchronization flag is found in the programmed sequence.
":<Handle>:SGENeral:GLOBal:SYNChronization:BERThreshold" on page 164	Sets the threshold for the bit error ratio to be accepted during synchronization.
":<Handle>:SGENeral:GLOBal:SYNChronization:BERThreshold?" on page 165	Returns the currently set bit error ratio threshold.
":<Handle>:SGENeral:GLOBal:SYNChronization:SMODe" on page 165	Specifies how the synchronization of the analyzers is achieved.
":<Handle>:SGENeral:GLOBal:SYNChronization:SMODe?" on page 166	Returns the currently set synchronization mode.
":<Handle>:SGENeral:GLOBal:SYNChronization:APAlignment" on page 166	Specifies whether an automatic phase optimization is done after a synchronization in infinite range.
":<Handle>:SGENeral:GLOBal:SYNChronization:APAlignment?" on page 167	Returns whether an automatic phase optimization is done after a synchronization in infinite range.
":<Handle>:SGENeral:GLOBal:SYNChronization:PACCuracy" on page 167	Specifies how accurate the phase optimization is done after a synchronization in infinite synchronization range.
":<Handle>:SGENeral:GLOBal:SYNChronization:PACCuracy?" on page 167	Returns the required phase optimization accuracy.

Analyzer Commands

Command (Link to detailed description)	Description
":<Handle>:SGENeral:GLOBal:CONFigure?" on page 147	Returns the last measurement mode setting in a quoted string.
":<Handle>:SGENeral:GLOBal:CONFigure:CAPTure" on page 147	Activates the Capture measurement mode.
":<Handle>:SGENeral:GLOBal:CONFigure[:ECOunt]" on page 148	Activates the Error Rate measurement mode.
":<Handle>:SGENeral:GLOBal:CONFigure:ECAPture" on page 148	Activates the Error Capture measurement mode.
":<Handle>:SGENeral:GLOBal:CONFigure:CCAPture" on page 149	Sets the instrument to Compare and CAPture mode.
":FETCh[:ECOunt]?" on page 293	Activates the Error Count measurement mode.
":ECOunt:RESet" on page 295	Resets the "received bit counter" and the "failed bit counter" to zero.
":<Handle>:SGENeral:GLOBal:FETCh:ERRor:ANY?" on page 141	Reports whether there has been found any error.
":FETCh:ERRor:ANY?" on page 293	Returns whether there has been found any error for the specified port or terminal.

www.valuetronics.com

Level Parameter Commands

The following table lists the commands available for pulse and data ports. The commands are available at port, terminal, and connector level.

Command (Link to detailed description)	Description
":OUTPut:TVOLtage" on page 262	Sets the external termination voltage.
":OUTPut:TVOLtage?" on page 262	Returns the current programmed value for the external termination voltage for the connectors of the specified port/ terminal/ connector.
":VOLTage[:LEVel][:IMMediate]:HIGH" on page 220	Sets the high level of the pulse.
":VOLTage[:LEVel][:IMMediate]:HIGH?" on page 221	Returns the high level value for the specified port/ terminal/ connector.
":VOLTage[:LEVel][:IMMediate]:LOW" on page 222	Sets the low level of the pulse.
":VOLTage[:LEVel][:IMMediate]:LOW?" on page 223	Returns the low level value for the specified port/ terminal/ connector.
":VOLTage[:LEVel][:IMMediate]:CAConfiguration:LOW" on page 224	Sets the additional low level of the pulse when channel addition mode is active.
":VOLTage[:LEVel][:IMMediate]:CAConfiguration:LOW?" on page 225	Returns the additional low level value for the specified port/ terminal/ connector when channel addition mode is active.
":VOLTage[:LEVel][:IMMediate]:SEMode" on page 226	Sets whether or not single-ended channel operation mode is used.
":VOLTage[:LEVel][:IMMediate]:SEMode?" on page 227	Returns whether or not single-ended channel operation mode is used.
":VOLTage[:LEVel][:IMMediate]:SEMode:AVAilable?" on page 227	Return whether or not single-ended channel operation mode is available.

www.valuetronics.com

Timing Parameter Commands

The following table lists the commands available for pulse and data ports. The commands are available at port, terminal, and connector level.

Command (Link to detailed description)	Description
":PULSe:DELay" on page 179	Sets the time from the start of the period to the first edge of the pulse.
":PULSe:DELay?" on page 180	Returns the delay value for the specified port/ terminal/ connector.
":PULSe:WIDTh" on page 185	Sets the width (duration) of the pulse.
":PULSe:WIDTh?" on page 185	Returns the width value for the specified port/ terminal/ connector.
":PULSe:DELay:CYCLes" on page 180	Sets the delay in cycles, where one cycle corresponds to the actual period/ frequency valid for the specified generator output connector.
":PULSe:DELay:CYCLes?" on page 181	Returns the number of delay cycles.
":PULSe:DELay:TIME" on page 182	Sets the delay in seconds.
":PULSe:DELay:TIME?" on page 182	Returns the delay.
":PULSe:DCYCle" on page 186	Sets the duty cycle of a repetitive pulse waveform.
":PULSe:DCYCle?" on page 186	Returns the duty cycle value for the specified port/ terminal/ connector.
":PULSe:HOLD" on page 187	Sets width or duty cycle to hold.
":PULSe:HOLD?" on page 187	Returns the hold parameter valid for the specified port/ terminal/ connector.
":PULSe:TRANsition[:LEADing]" on page 188	Sets the leading edge value for the specified port/ terminal/ connector.
":PULSe:TRANsition[:LEADing]?" on page 189	Returns the leading edge value for the specified port/ terminal/ connector.
":PULSe:TRANsition:TRAiling" on page 190	Sets the trailing edge value for the specified port/ terminal/ connector.
":PULSe:TRANsition:TRAiling?" on page 191	Returns the trailing edge value for the specified port/ terminal/ connector.
":PULSe:TRANsition:CAConfiguration[:LEADing]" on page 192	Sets the additional leading edge value for the specified port/ terminal/ connector, when channel addition mode is active.
":PULSe:TRANsition:CAConfiguration[:LEADing]?" on page 193	Returns the additional leading edge value for the specified port/ terminal/ connector, when channel addition mode is active.
":PULSe:TRANsition:CAConfiguration:TRAiling" on page 194	Sets the trailing edge value for the specified port/ terminal/ connector.
":PULSe:TRANsition:CAConfiguration:TRAiling?" on page 195	Returns the additional trailing edge value for the specified port/ terminal/ connector when channel addition mode is active.
":MUX" on page 196	SetS the frequency multiply factor for the specified port/ terminal/ connector.
":MUX?" on page 196	Returns the current frequency multiply factor (MUX factor) for the specified port/ terminal/ connector.

www.valuetronics.com

MUX Parameter Commands

The following table lists the commands for setting/getting MUX parameters.

Command (Link to detailed description)	Description
":VOLTage:AMPLitude" on page 201	Sets the amplitude of the multiplexer output signal.
":VOLTage:AMPLitude?" on page 201	Returns the amplitude of the multiplexer output signal.
":VOLTage:OFFSet" on page 202	Adds an offset voltage to the generated signal.
":VOLTage:OFFSet?" on page 202	Returns the amplitude of the offset voltage.
":VOLTage:OFFSet:ENABle" on page 203	Enables/ disables voltage offset.
":VOLTage:OFFSet:ENABle?" on page 203	Returns the status of voltage offset.
":OUTPut:EATTenuator" on page 204	Sets the attenuation value of an external attenuator.
":OUTPut:EATTenuator?" on page 204	Returns the entered attenuation value of an external attenuator.
":OUTPut:DELay:SWEep" on page 205	Sets the value for the additional delay.
":OUTPut:DELay:SWEep?" on page 206	Returns the value for the additional delay.
":OUTPut:DELay:SWEep:MODe" on page 206	Sets the delay correction or the delay calibration mode.
":OUTPut:DELay:SWEep:MODe?" on page 207	Returns the current mode for the delay.
":OUTPut:DELay:SWEep:PROM" on page 208	The delay values of a MUX module are stored in the module's EEPROM. They are independent of the loaded setting. This command writes them to or reads them from the EEPROM.
":TRIGger" on page 210	Sets the source of the clock signal used for the timing of the E4868A multiplexer or E4869A demultiplexer.
":TRIGger?" on page 210	Returns the source of the clock signal used for the timing of the E4868A multiplexer or E4869A demultiplexer.
":TRIGger:MODE" on page 211	Sets the speed of the subrate clock output signal.
":TRIGger:MODE?" on page 211	Returns the speed of the subrate clock output signal.
":INPut:THReshold" on page 242	Specifies the threshold voltage of an input connector (only available for the E4969B).
":INPut:THReshold?" on page 242	Returns the current threshold voltage of the specified input connector (only available for the E4969B).

www.valuetronics.com

DeMUX Parameter Commands

The following table lists the commands for setting/getting DeMUX parameters.

Command (Link to detailed description)	Description
":INPut:OCOMpensation" on page 209	Sets the decision threshold for sampling the incoming signal.
":INPut:OCOMpensation?" on page 209	Returns the decision threshold for sampling the incoming signal.
":TRIGger" on page 210	Sets the source of the clock signal used for the timing of the E4868A multiplexer or E4869A demultiplexer.
":TRIGger?" on page 210	Returns the source of the clock signal used for the timing of the E4868A multiplexer or E4869A demultiplexer.
":INPut:DELay:SWEep" on page 249	Sets the delay sweep value (only available for the E4969B).
":INPut:DELay:SWEep?" on page 249	Returns the current delay sweep value (only available for the E4969B).
":<Handle>:SGENeral:GLOBal:SYNChronization:REWire" on page 214	Switches the DeMUX rewiring on or off.
":<Handle>:SGENeral:GLOBal:SYNChronization:REWire?" on page 214	Returns the current state of the DeMUX rewiring.
":<Handle>:SGENeral:GLOBal:SYNChronization:RMODe" on page 215	Sets the modes of DeMUX rewiring.
":<Handle>:SGENeral:GLOBal:SYNChronization:RMODe?" on page 216	Returns the current mode of DeMUX rewiring.
":<Handle>:SGENeral:GLOBal:SYNChronization:RCH" on page 216	Switches the rewire verify mode on and off.
":<Handle>:SGENeral:GLOBal:SYNChronization:RCH?" on page 217	Returns the current rewire verify mode.
":<Handle>:SGENeral:PDATa:DMUX:STAGe" on page 217	Sets the number of stages of the demultiplexer.
":<Handle>:SGENeral:PDATa:DMUX:STAGe?" on page 218	Returns the number of stages of the demultiplexer.
":<Handle>:SGENeral:PDATa:DMUX:STAGe:OUT" on page 218	Sets the number of outputs per demultiplexer for a specified stage.
":<Handle>:SGENeral:PDATa:DMUX:STAGe:OUT?" on page 219	Returns the number of outputs per demultiplexer for a specified stage.

www.valuetronics.com

DeMUX Rewiring Commands

The following table lists the DeMUX rewiring commands.

Command (Link to detailed description)	Description
":<Handle>:SGENeral:GLOBal:SYNChronization:REWire" on page 214	Switches the DeMUX rewiring on or off.
":<Handle>:SGENeral:GLOBal:SYNChronization:REWire?" on page 214	Returns the current state of the DeMUX rewiring.
":<Handle>:SGENeral:GLOBal:SYNChronization:RMODe" on page 215	Sets the modes of DeMUX rewiring.
":<Handle>:SGENeral:GLOBal:SYNChronization:RMODe?" on page 216	Returns the current mode of DeMUX rewiring.
":<Handle>:SGENeral:GLOBal:SYNChronization:RCH" on page 216	Switches the rewire verify mode on and off.
":<Handle>:SGENeral:GLOBal:SYNChronization:RCH?" on page 217	Returns the current rewire verify mode.
":<Handle>:SGENeral:PDATa:DMUX:STAGe" on page 217	Sets the number of stages of the demultiplexer.
":<Handle>:SGENeral:PDATa:DMUX:STAGe?" on page 218	Returns the number of stages of the demultiplexer.
":<Handle>:SGENeral:PDATa:DMUX:STAGe:OUT" on page 218	Sets the number of outputs per demultiplexer for a specified stage.
":<Handle>:SGENeral:PDATa:DMUX:STAGe:OUT?" on page 219	Returns the number of outputs per demultiplexer for a specified stage.

Delay Control Parameter Commands

The following table lists the commands available for pulse and data ports. The commands are available at port, terminal, and connector level.

Command (Link to detailed description)	Description
":DCONtrol[:DEViation]?" on page 230	Returns the current maximum delay control deviation in seconds.
":DCONtrol:SENSitivity" on page 231	Sets the delay control sensitivity of the external delay control modulation signal.
":DCONtrol:SENSitivity?" on page 231	Returns the delay control sensitivity of the external delay control modulation signal.
":DCONtrol:SOURce" on page 232	Sets the delay control source.
":DCONtrol:SOURce?" on page 233	Returns the delay control source.
":DCONtrol:STATe" on page 234	Switches the specified external delay control input connector on or off.
":DCONtrol:STATe?" on page 235	Returns the current state of the specified external delay control input connector.

www.valuetronics.com

Input Parameter Commands

The following table lists the commands available for data ports only. The commands are available at port, terminal, and connector level.

Command (Link to detailed description)	Description
":INPut[:STATe]" on page 236	Enables or disables an analyzer input connector.
":INPut[:STATe]?" on page 236	Returns the current status of the specified analyzer input connector.
":INPut:TVOLtage" on page 241	Specifies the termination voltage of an analyzer input connector.
":INPut:TVOLtage?" on page 241	Returns the current termination voltage of the specified analyzer input connector.
":INPut:THReshold" on page 242	Specifies the threshold voltage of an analyzer input connector.
":INPut:THReshold?" on page 242	Returns the current threshold voltage of the specified analyzer input connector.
":INPut:DELay" on page 245	Specifies the delay of the sampling edge.
":INPut:DELay?" on page 245	Returns the current delay of the sampling edge for the specified analyzer input connector.
":INPut:DELay:CYCLes" on page 246	Specifies the delay of the sampling edge in terms of cycles corresponding to the actual period/ frequency valid for the specified analyzer input connector.
":INPut:DELay:CYCLes?" on page 246	Returns the current delay of the sampling edge for the specified analyzer input connector in terms of cycles.
":INPut:DELay:TIME" on page 247	Specifies the delay of the sampling edge.
":INPut:DELay:TIME?" on page 247	Returns the current delay of the sampling edge for the specified analyzer input connector.
":INPut:DELay:ACTual?" on page 248	Returns the actual delay after synchronization.
":INPut:DELay:SWEep" on page 249	Sets the delay sweep value.
":INPut:DELay:SWEep?" on page 249	Returns the current delay sweep value.
":INPut:POLarity" on page 237	Sets the input polarity of the specified data port/ terminal/ connector to either normal or inverted.
":INPut:POLarity?" on page 237	Returns the input polarity of the specified port/ terminal/ connector.
":INPut:TYPE" on page 238	Selects how the E4835A, E4863A, and E4865A differential input frontends are sampled.
":INPut:TYPE?" on page 239	Returns the operation mode of the E4835A, E4863A, and E4865A differential input frontend.
":INPut:MODE" on page 239	Selects between single-ended termination and differential termination mode.
":INPut:MODE?" on page 240	Returns the termination mode of an input frontend.
":SENSe:VOLTage:RANGe" on page 228	Selects the allowed input voltage range for the E4835A differential input frontend.
":SENSe:VOLTage:RANGe?" on page 229	Returns the currently selected input voltage range of an E4835A differential input frontend.
":INPut:IMPedance[:INTernal]" on page 243	Specifies the internal termination impedance of the analyzer input connector.

www.valuetronics.com

Command (Link to detailed description)	Description
":INPut:IMPedance[:INTernal]?" on page 243	Returns the current internal impedance of the specified analyzer input connector.
":INPut:SERial" on page 244	Specifies a serial impedance on this analyzer input connector.
":INPut:SERial?" on page 244	Returns the current serial impedance used for the specified analyzer input connector.
":INPut:TCONfig" on page 250	Selects the termination model for input frontends.
":INPut:TCONfig?" on page 251	Returns the termination model for input frontends.
":INPut:DIMPedance" on page 251	Sets the impedance between the IN and \ IN\ inputs of an E4835A, E4863A and E4865A frontend when the differential termination model is selected.
":INPut:DIMPedance?" on page 252	Returns the differential termination resistor.
":INPut:DISConnect:MODE?" on page 252	Returns how the specified connector is to be disconnected when the `SGENeral:GLOBal:CONNect OFF` command is executed.
":INPut:DISConnect:MODE" on page 253	Sets how the specified connector is to be disconnected when the `SGENeral:GLOBal:CONNect OFF` command is executed.

Optical Input Commands

The following table lists the commands available for data ports. The commands are available at port, terminal, and connector level.

Command (Link to detailed description)	Description
":INPut:OPTic[:STATe][:VALue]" on page 254	Sets the state of the optical channel(s).
":INPut:OPTic[:STATe][:VALue]?" on page 254	Queries the state of the optical channel(s).
":INPut:OPTic:POWer:UNIT[:VALue]" on page 255	Sets the unit for the 0/ 1 decision threshold of the selected optical input connector, port, or terminal.
":INPut:OPTic:POWer:UNIT[:VALue]?" on page 255	Queries the unit for the 0/ 1 decision threshold of the selected optical input connector, port, or terminal.
":INPut:OPTic:POWer:THReshold[:VALue]" on page 256	Sets the 0/ 1 decision threshold of the selected optical input connector, port, or terminal.
":INPut:OPTic:POWer:THReshold[:VALue]?" on page 256	Queries the 0/ 1 decision threshold of the selected optical input connector, port, or terminal.
":INPut:OPTic:MEASure" on page 256	Measures the average signal power and sets this as the 0/ 1 decision threshold on the selected optical input connector, port, or terminal.
":INPut:OPTic:DLCalibration" on page 257	Performs dark level calibration on the selected optical input connector, port, or terminal.

www.valuetronics.com

Output Parameter Commands

The following table lists the commands available for pulse and data ports. The commands are available at port, terminal, and connector level.

Command (Link to detailed description)	Description
":OUTPut[:STATe]" on page 258	Controls whether the connectors of the specified port/ terminal/ connector are opened or closed.
":OUTPut[:STATe]?" on page 259	Returns the current connection state for the connectors of the specified port/ terminal/ connector.
":OUTPut:CSTate" on page 261	Controls whether the complement connectors of the specified port/ terminal/ connector are opened or closed.
":OUTPut:CSTate?" on page 261	Returns the current connection state for the complement connectors of the specified port/ terminal/ connector.
":OUTPut:TVOLtage" on page 262	Sets the external termination voltage.
":OUTPut:TVOLtage?" on page 262	Returns the current programmed value for the external termination voltage for the connectors of the specified port/ terminal/ connector.
":OUTPut:POLarity" on page 260	Sets the output polarity of the specified data port/ terminal/ connector to either normal or inverted.
":OUTPut:POLarity?" on page 260	Returns the output polarity of the specified port/ terminal/ connector.
":OUTPut:IMPedance:EXTernal" on page 263	Sets the external termination impedance, the real load impedance of the DUT.
":OUTPut:IMPedance:EXTernal?" on page 263	Returns the currently set value for the termination (load) impedance.
":OUTPut:TCONfig" on page 264	Selects the termination model for output frontends.
":OUTPut:TCONfig?" on page 265	Returns the termination model for output frontends.
":OUTPut:DIMPedance:EXTernal" on page 266	Sets the impedance between the OUT and \ OUT\ outputs of an E4838A or E4843A frontend when the differential termination model is selected.
":OUTPut:DIMPedance:EXTernal?" on page 266	Returns the differential termination impedance.
":OUTPut:CAConfiguration[:MODE]" on page 267	Selects the channel-add mode of a port/ terminal/ connector.
":OUTPut:CAConfiguration[:MODE]?" on page 268	Returns the current channel addition mode of the specified port/ terminal/ connector.
":OUTPut:DISConnect:MODE?" on page 268	Returns how the specified connector is to be disconnected when the `SGENeral:GLOBal:CONNect OFF` command is executed.
":OUTPut:DISConnect:MODE" on page 269	Sets how the specified connector is to be disconnected when the `SGENeral:GLOBal:CONNect OFF` command is executed.

www.valuetronics.com

Optical Output Commands

The following table lists the commands available for pulse and data ports. The commands are available at port, terminal, and connector level.

Command (Link to detailed description)	Description
":OUTPut:OPTic[:STATe][:VALue]" on page 271	Enables/ disables the laser on the E/ O converter.
":OUTPut:OPTic[:STATe][:VALue]?" on page 272	Queries the state of the laser on the E/ O converter.
":OUTPut:OPTic:METHod[:VALue]" on page 273	Defines the method used for setting optical power values.
":OUTPut:OPTic:METHod[:VALue]?" on page 273	Queries the currently used method for setting optical power values.
":OUTPut:OPTic:POWer:UNIT[:VALue]" on page 274	Sets the unit for all power parameters of the selected optical output connector, port, or terminal.
":OUTPut:OPTic:POWer:UNIT[:VALue]?" on page 274	Queries the unit for all power parameters of the selected optical output connector, port, or terminal.
":OUTPut:OPTic:POWer:AVERage[:VALue]" on page 275	Sets the average power of the selected optical output connector, port, or terminal.
":OUTPut:OPTic:POWer:AVERage[:VALue]?" on page 275	Queries the average power of the selected optical output connector, port, or terminal.
":OUTPut:OPTic:POWer:MAMPlitude[:VALue]" on page 276	Sets the modulation of the selected optical output connector, port, or terminal.
":OUTPut:OPTic:POWer:MAMPlitude[:VALue]?" on page 277	Queries the modulation amplitude of the selected optical output connector, port, or terminal.
":OUTPut:OPTic:POWer:HIGH[:VALue]" on page 277	Sets the high power of the selected optical output connector, port, or terminal.
":OUTPut:OPTic:POWer:HIGH[:VALue]?" on page 278	Queries the high power of the selected optical output connector, port, or terminal.
":OUTPut:OPTic:POWer:LOW[:VALue]" on page 278	Sets the low power of the selected optical output connector, port, or terminal.
":OUTPut:OPTic:POWer:LOW[:VALue]?" on page 279	Queries the low power of the selected optical output connector, port, or terminal.
":OUTPut:OPTic:ERATio[:VALue]" on page 279	Sets the extinction ratio of the selected optical output connector, port, or terminal.
":OUTPut:OPTic:ERATio[:VALue]?" on page 280	Queries the extinction ratio of the selected optical output connector, port, or terminal.
":OUTPut:OPTic:ERATio:UNIT[:VALue]" on page 280	Sets the unit used for the extinction ratio of the selected optical output connector, port, or terminal.
":OUTPut:OPTic:ERATio:UNIT[:VALue]?" on page 281	Queries the unit used for the extinction ratio of the selected optical output connector, port, or terminal.

www.valuetronics.com

Performance Parameter Commands

The following table lists the commands available for pulse and data ports. The commands are available at port, terminal, and connector level.

Command (Link to detailed description)	Description
":SIGNal:MODE" on page 297	Sets the performance mode for the clock signal (not available for all clock modules).
":SIGNal:MODE?" on page 298	Returns the performance mode for the clock signal (not available for all clock modules).

Format Parameter Commands

The following table lists the commands available for pulse and data ports. The commands are available at port, terminal, and connector level.

Command (Link to detailed description)	Description
":FORMat" on page 299	Controls the output connectors data format of the specified port/ terminal/ connector.
":FORMat?" on page 300	Returns the current data format state for the output connectors of the specified port/ terminal/ connector.

www.valuetronics.com

www.valuetronics.com

Commands in the DVT Subsystem

The DVT subsystem is always available. The commands in this subsystem provide basic administration features.

This reference starts with an introductory example and then lists the available commands and subsystems:

- DVT
  - Top-level command :DVT:IDN?
  - :SYSTem Subsystem
  - :INSTrument Subsystem
  - :INSTrument:HANDle Subsystem
  - :MMEMory Subsystem

www.valuetronics.com

Example: Use of :DVT Subsystem Commands

```
:DVT:INST:LIST?              # check for instrument types
                             # available
                             # Returns: "DSR"

:DVT:INST:RES:LIST? "DSR"    # check for available virtual
                             # instruments
                             # Returns: "DSR1"

                             # query to get a handle for the
                             # virtual instrument. _TEST is a
                             # suggestion:
:DVT:INST:HAND:CRE? _TEST, "DSR", "DSR1"
                             # The suggested handle is accepted.
                             # Returns: _TEST

:DVT:INST:HAND:LIST?         # Lists the handles in use
                             # Returns: "DVT", "_TEST"

:_TEST:IDN?                  # Queries for the virtual instrument
                             # identification
                             # Returns:
                             # Agilent Technologies,E4875A,0,0

:DVT:instrument:handle:destroy _TEST
                             # destroy the handle _TEST
                             # (do not forget the underscore)
```


www.valuetronics.com

Top-Level Commands

On top level of the DVT subsystem there is only one command available:

- :DVT:IDN?

:DVT:IDN?

Command `:DVT:IDN?`

Syntax `:DVT:IDN?`

Description Identifies the device.

The response consists of the following four fields (fields are separated by commas):

- Manufacturer
- Model Number
- Serial Number (returns 0 if not available)
- Firmware Revision (returns 0 if not available)

Example

```
:dvt:idn?
```

might return

```
Agilent Technologies,Agilent 81200, 0, 0
```


www.valuetronics.com

:SYSTem Subsystem

The DVT:SYSTem subsystem supports the following command:

- :DVT:SYSTem:ERRor?

:DVT:SYSTem:ERRor?

Command `:DVT:SYSTem:ERRor?`

Syntax `:DVT:SYSTem:ERRor?`

Description The query returns the first entry of the error queue and removes it from the queue. As long as the command does not return `0, "No error"`, there are further errors in the queue.

Example

```
:dvt:syst:err?
```

might return

```
0,"No error"
```


www.valuetronics.com

:INSTrument Subsystem

The :INSTrument subsystem provides a mechanism to identify available applications and create handles for virtual instruments (see *"The Command Tree" on page 30*).

The :DVT:INSTrument subsystem supports the following commands:

- DVT:INSTrument
  - :LIST?
  - :RESource:LIST?
  - :HANDle Subsystem

:DVT:INSTrument:LIST?

Syntax `:DVT:INSTrument:LIST?`

Description The LIST? query returns a comma-separated list of <character response data> items containing the names of all applications.

Example `:DVT:INST:LIST?`

might return

`"DSR"`

:DVT:INSTrument:RESource:LIST?

Syntax `:DVT:INSTrument:RESource:LIST? <"Application">`

Parameters **<quoted string>** The name of the desired application

The response is a comma-separated list of <character response data> items.

Description The LIST? query returns a comma-separated list of hardware resources which may be connected to the specified application.

Example `:DVT:INSTrument:RESource:LIST? "DSR"`

might return

`"DSRA"`

www.valuetronics.com

:INSTrument:HANDle Subsystem

Syntax :DVT:INSTrument:HANDle

The HANDle subsystem collects the functions that are related to virtual instrument management.

:DVT:INSTrument:HANDle:CREate?

Syntax `:DVT:INSTrument:HANDle:CREate? <Handle>, <"Application">, <"Resource">`

Parameters **<Handle>** Suggested value for the Handle

<"Application"> The name of the desired application. The available applications may be listed using `:DVT:INST:LIST?`.

<"Resource"> The name of the hardware resource with which the new virtual instrument should be associated. Available resources may be listed using `:DVT:INST:RES:LIST?`.

Description Creates a new virtual instrument.

Return A <Character Response Data> element containing the name of the new handle is returned. <Handle> will be used if possible. If <Handle> is illegal for any reason, a legal and unique handle will be generated and returned.

A valid handle always starts with a "_" (it must match the following regular expression: _[a-zA-Z0-9]*[a-zA-Z]).

Example

```
:DVT:INST:HAND:CRE? _TEST, "DSR", "DSR1"
:DVT:INST:HAND:CRE? TEST, "DSR", "DSR1"
```

might return

```
_TEST
```


www.valuetronics.com

:DVT:INSTrument:HANDle:LIST?

Syntax `:DVT:INSTrument:HANDle:LIST?`

Returns a comma-separated list of all virtual instrument handles. The list will always contain the handle DVT.

Example `DVT:INST:HAND:LIST?`

might return “DVT”, “_TEST”

:DVT:INSTrument:HANDle:DESTroy

Syntax `DVT:INSTrument:HANDle:DESTroy <Handle>`

Parameters **<Handle>** The handle of the virtual instrument to be destroyed.

Destroys a handle created by CREate? and frees all associated resources.

Example `DVT:INST:HAND:DEST _TEST`

www.valuetronics.com

:MMEMory Subsystem

The DVT:MMEMory subsystem supports the following commands:

- :DVT:MMEMory
  - :SETTing:EXPort
  - :SETTing:EXPort?
  - :SETTing:IMPort:DVT:MMEMory:SETTing:EXPort

Syntax

```
:DVT:MMEMory:SETTing:EXPort <Handle>,
<"FileName">[,<YDELay | NDELay | ODELay>]
```

Parameters

<Handle> The handle of the virtual instrument.

<"FileName"> The name of the file that will be created.

YDELay Yes delay; export setting with cable delay parameters.

NDELay No delay; export setting without cable delay parameters.

ODELay Only delay; export only cable delay parameters.

Description

The system generates a file with the described setting. The optional parameter filters the cable delay parameters.

Example

```
:DVT:MMEM:SETT:EXP _TEST, "c:\ParBERT_Exchange\withdel.txt", YDEL
```

:DVT:MMEMory:SETTing:EXPort?

Syntax

```
:DVT:MMEMory:SETTing:EXPort? <Handle> [,<YDELay | NDELay |
ODELay>]
```

Parameters

<Handle> The handle of the virtual instrument.

YDELay Yes delay; export setting with cable delay parameters.

NDELay No delay; export setting without cable delay parameters.

ODELay Only delay; export only cable delay parameters.

Description

The query returns an expression describing the current setting. The optional parameter filters the cable delay parameters.

www.valuetronics.com

Example `:DVT:MMEM:SETT:EXP? _TEST, NDEL`

:DVT:MMEMory:SETTing:IMPort

Syntax
```
:DVT:MMEMory:SETTing:IMPort <Handle>,
<"Filename">|<(Expression)>
```

Parameters

<Handle> The handle of the virtual instrument for which the setting is supposed to be used.

<"Filename"> The name of a file that contains a series of SCPI commands describing the setting.

<Expression> A set of commands in form of an expression. The commands have to be listed without the <Handle> subnode.

TIP Use the :DVT:MMEMory:SETTing:EXPort? command to see the necessary format.

Description This command makes it possible to import a setting into the Agilent 81250 System.

Example `:DVT:MMEM:SETT:IMP _TEST, "c:\ParBERT_Exchange\setting1.txt"`

www.valuetronics.com

www.valuetronics.com

Commands in DSR Application Subsystems

This chapter describes the commands available for the DSR application of the Agilent 81250 system. The general commands always available for the system are described in *"Commands in the DVT Subsystem" on page 65*.

For basic information on the command syntax and the command tree, please refer to *"Overview of the SCPI Command Structure" on page 27*.

Each command and query entry has at least some of the following items:

- Full command syntax
- Brief description
- Parameters
- Return value
- Possible error conditions
- Cross-references to related commands
- Short example showing context in which the command is used, its effect and return value

www.valuetronics.com

Administration Commands

On top level, there are commands for managing the system.

:< Handle> :IDN?

Syntax :<Handle>:IDN?

Description Identifies the device.

The response consists of the following four fields (fields are separated by commas):

- Manufacturer
- Model Number
- Serial Number (returns 0 if not available)
- Firmware Revision (returns 0 if not available)

Example

```
:_test:IDN?
```

might return:

```
Agilent Technologies,E4875A,0,Rev. 1.01
```

:< Handle> :OPC?

Syntax :<Handle>:OPC?

Return Value The query returns when all modules report that they are ready.

Data is not available at the outputs immediately after an SGEN:GLOB:INIT:CONT ON command is sent, because of the pipelined system architecture.

Returns 1 when all pending operations are finished.

Example

```
:_test:SGEN:GLOB:PER 100e-9
:_test:OPC?
```


www.valuetronics.com

:SYSTem Subsystem

Syntax :<Handle>:SYSTem

The :SYSTem subsystem provides commands to check errors or to switch on/off the internal error check system.

:< Handle> :SYSTem:ERRor?

Syntax :<Handle>:SYSTem:ERRor?

Return Value When errors are detected, they are placed in a queue. This command returns the first entry of the error queue to the user.

Example

```
:_test:SYST:ERR?
```

might return for example:

```
0,"No error"
```

or, another example:

```
-300, "Device-specific error; 2001 Terminal suffix out of range"
```


www.valuetronics.com

:<Handle>:SYSTem:CINFormation?

Syntax :<Handle>:SYSTem:CINFormation? <SHORt | DETailed> [,<"FileName">]

Parameters **<"FileName">** (Optional) The settings file for the system.

Description This query returns the system configuration information. For each element, the subelements are listed (for example, for a mainframe, the modules in that mainframe are listed; for each module, the frontends are listed).

Return Value The response is an expression with the description of all elements in the following syntax:

<ModuleInfo>	::=	"("<ProductNr> "," <SpeedClass> ["," <SerNr> "," <IDNr>] "," <ConnInfo> ("," <ConnInfo>)* ")"
<ConnInfo>	::=	"("<ProductNr> "," <SpeedClass> "," "Type" ["," <SerNr> "," <IDNr> "," <ConnNr>] ")"
<ProductNr>	::=	for example, "E4838A"
<SpeedClass>	::=	for example, 660
<Type>	::=	"ANALYZER" | "GENERATOR"
<SerNr>	::=	serial number (0 if not available)
<IDNr>	::=	identification number (0 if not available)
<ConnNr>	::=	"S1" | "D1" | "D2"

S1: single frontend
D1: first connector of a dual frontend
D2: second connector of a dual frontend

The optional syntax elements only appear if DETailed is selected.

Example

```
:_test:SYST:CINF? SHOR
```

could return, for example (reformatted only for readability):

```
((E4805B,660),
  (E4861A,2640,
    (E4863A,2640,ANALYZER),
    (E4862A,2640,GENERATOR)),
  (E4861A,2640,
    (E4863A,2640,ANALYZER),
    (E4862A,2640,GENERATOR)))
```


www.valuetronics.com

:< Handle> :SYSTem:TEST:M ODule

Syntax :<Handle>:SYSTem:TEST:MODule DETailed | SHORt [,CGR#, MOD#]

Description A module selftest is started by the firmware on every module. The results of this test are stored in the message queue. The message queue can be read by with the command :SYST:MQU?.

Parameters The format of the messages can be specified by following keywords:

<DETailed> Generates a more detailed message.

<SHORt> Generates a less detailed message.

[,CGR#, MOD#] A single module can be specified by the optional parameters. The first parameter (CGR#) is the clock group the module belongs to, the second parameter (MOD#) is the number of the module itself.

Example For a SHORt message:

```
(1,3,1,4,'E4861A','Passed','E4861A Ver: 1.92 E4863 E4862 ','No
error')
```

For a DETailed message:

```
= (1,1,1,2,'E4805B','Failed','< AGILENT TECHNOLOGIES, E4805B, 01,
<>, "", Module Software Version = 1.81, Boot Loader Version = 1.00
>','-330, Self-test failed: Frequency generation PLL test failed:
high limit not reached!-330, Self-test failed: Frequency generation
PLL test failed: high limit not reached!')
```

A DETailed message for an E4861A could appear as:

```
(1,4,1,7,'E4861A','Failed','< AGILENT TECHNOLOGIES, E4861A, 00, <>,
"", Module Software Version = 0.32, Boot Loader Version = 0.01,
<E4863, 01, <>, "">, <E4862, 01, <>, "> >','-330, Self-test
failed: Voltage of +5VRef is out of valid range')
```


www.valuetronics.com

:< Handle> :SYSTem:TEST:GLOBal

Syntax :<Handle>:SYSTem:TEST:GLOBal DETailed | SHORt

Description This command initiates a system-wide selftest. The local bus and the SYSCLK signal distribution is tested. The selftest messages are placed in the message queue. First the message queue is cleared, and then the new results will be inserted.

Parameters The format of the messages can be specified by following keywords:

<DETailed> Generates a more detailed message.

<SHORt> Generates a less detailed message.

:< Handle> :M ODule:SREVisions

Syntax :<Handle>:MODule:SREVisions DETailed | SHORt

Description This command checks the actual BIOS Software REVisions against the expected revision located in the firmware. The results of this test are located in the message queue.

The message queue can be read with the command :SYST:MQU?

Parameters The format of the messages can be specified by following keywords:

<DETailed> Generates a more detailed message.

<SHORt> Generates a less detailed message.

Return Values The following values may be placed in the message queue:

- BIOS: Update is needed
- BIOS: Update is not needed

The module selftest message is replaced by the expected revision.

www.valuetronics.com

:<Handle>:SYSTem:MQUeue[:READ]?

Syntax :<Handle>:SYSTem:MQUeue[:READ]?

Return Value Returns and then deletes the first item contained in the message queue. The next execution of this command reports the next message contained in the queue.

If the queue is empty, "( )" is returned.

This command can be used, for example, to read the message queue after selftests have been executed for all modules. It has to be executed for each module.

NOTE The :READ option has no effect.

:<Handle>:SYSTem:MQUeue:LENGth?

Syntax :<Handle>:SYSTem:MQUeue:LENGth?

Return Value Returns the number of messages in the message queue.

:<Handle>:SYSTem:PON[:STATus]

Syntax :<Handle>:SYSTem:PON[:STATus] DETailed | SHORt

Description Moves the messages that are generated by the modules at system start up into the message queue. This command does *not* start a new selftest.

The message queue can be read with the command :SYST:MQU?.

Parameters A DETailed message is required for service purposes (in-depth message according to the Service selection on the GUI), SHORt is a summary status according to the user selection in the GUI.

www.valuetronics.com

:SYSTem:CLIent Subsystem

Syntax :<Handle>:SYSTem:CLIent

More than one client can simultaneously operate with the firmware server (for example, GUI-client, GPIB-client, VEE-client, ...).

To avoid conflicts, the commands in this subsystem control the firmware access. These commands allow using a system (virtual instrument) exclusively. There are two locking commands, which differ in the reaction of the locked application:

- If a system is locked by a client using the LOCK command, any access by another client returns an error code.
- If a system is locked by a client using the BLOCK command, any access by another client will simply wait until the system is unlocked, thus blocking the client. No error code is returned.

:< Handle> :SYSTem:CLIent[:HANDle]?

Syntax :<Handle>:SYSTem:CLIent[:HANDle]?

Return Value This query returns the name of the client that has currently locked the server. An empty string is returned if the server is not locked.

Example

```
:_test:SYST:CLI?
```

might return for example:

```
_GPIB
```

:< Handle> :SYSTem:CLIent:LOCK

Syntax :<Handle>:SYSTem:CLIent:LOCK

This command can be used to take over the control of the virtual instrument explicitly.

If another client tries to access a locked virtual instrument, the error code -19, “ANOTHER_CLIENT_USES_APPLICATION” is returned.

NOTE A locked virtual instrument can be blocked by calling: `SYSTem:CLIent:BLOCk`.

Example

```
:_test:SYST:CLI:LOCK
```


www.valuetronics.com

:< Handle> :SYSTem:CLIent:BLOCk

Syntax :<Handle>:SYSTem:CLIent:BLOCk

Description This command is used to take over the control of the virtual instrument explicitly.

If another client sends a command to a blocked virtual instrument, the command does not return until the instrument is unlocked (thus blocking the sender). No error message is returned.

The usage of this command should be restricted to *short* accesses.

NOTE A blocked virtual instrument can only be locked by calling: `SYSTem:CLIent:LOCk`.

Example

```
:_test:SYST:CLI:BLOC
```

:< Handle> :SYSTem:CLIent:UNLock

Syntax :<Handle>:SYSTem:CLIent:UNLock

Description This command is used to unlock (or "free") the virtual instrument. This command performs an undo of `SYSTem:CLIent:LOCk` and `SYSTem:CLIent:BLOCk`.

Example

```
:_test:SYST:CLI:UNL
```


www.valuetronics.com

:CONFiguration Subsystem

Syntax :<Handle>:CONFiguration

:<Handle>:CONFiguration:CGRoups(*)?

Syntax :<Handle>:CONFiguration:CGRoups(*)?

Return Value Returns the number of available clock groups. The (*) specifier is ignored.

Related Commands *"[:CGRoup(*)] Subsystem" on page 121*

Example

```
:_test:CONF:CGR?
```

might return:

```
1
```

:<Handle>:CONFiguration:CGRoups(*):MODules(*)?

Syntax :<Handle>:CONFiguration:CGRoups(*):MODules(*)?

Return Value Returns the number of modules contained in the specified clock group. The (*) specifier for modules is ignored.

Related Commands *"[:CGRoup(*)] Subsystem" on page 121*

Example

```
:_test:CONF:CGR1:MOD?
```

might return:

```
3
```


www.valuetronics.com

:<Handle>:CONFiguration:CGRoups(*):MODules(*):CONNectors(*)?

Syntax :<Handel>:CONFiguration:CGRoups(*):MODules(*):CONNectors(*)?

Return Value Returns the number of connectors of the specified module. The (*) specifier for connectors is ignored.

Related Commands *"[:CGRoup(*)] Subsystem" on page 121*

Example
```
:_test:CONF:CGR1:MOD2:CONN?
```
might return:
```
4
```

:<Handle>:CONFiguration:CGRoups(*):MODules(*):CONNectors(*):TYPE?

Syntax :<Handle>:CONFiguration:CGRoups(*):MODules(*):CONNectors(*):TYPE?

Return Value Returns the type of the specified connector.

Related Commands *"[:CGRoup(*)] Subsystem" on page 121*

Example
```
:_test:CONF:CGR1:MOD2:CONN1:TYPE?
```
might return:
```
ANALYZER
```

:<Handle>:CONFiguration:STYPes?

Syntax :<Handle>:CONFiguration:STYPes?

Return Value Returns a list of scheme types supported by the Agilent 81250 System. See also *":SGENeral Subsystem" on page 137*.

Example
```
:_test:CONF:STYP?
```
might return:
```
"SCHEME_GENERAL"
```


www.valuetronics.com

:<Handle>:CONFiguration:PROFile[:VALue]

Syntax :<Handle>:CONFiguration:PROFile[:VALue] <"ParameterName">, <(Expression)>

Parameters **<"ParameterName">** Name of parameter to be set.

<(Expression)> Specified value.

Description Sets the parameter <"ParameterName"> to <(Expression)>.

Return Value None

Example
```
:_test:CONF:PROF "PARAM1", (10d,10,"TEXT",10u,10f)
```

:<Handle>:CONFiguration:PROFile[:VALue]?

Syntax :<Handle>:CONFiguration:PROFile[:VALue]? <"ParameterName">

Parameters **<"ParameterName">** Name of parameter to be queried.

Return Value Returns the value of <"ParameterName">.

Example
```
:_test:CONF:PROF? "PARAM1"
```
might return:
```
(10d,10,"TEXT",10u,10f)
```

:<Handle>:CONFiguration:PROFile:REMove

Syntax :<Handle>:CONFiguration:PROFile:REMove <"ParameterName">

Parameters **<"ParameterName">** Parameter Name

Removes the specified parameter contained in the "profile".

Return Value None

Example
```
:_test:CONF:PROF:REM "PARAM1"
```


www.valuetronics.com

:< Handle> :CONFiguration:PROFile:LIST?

Syntax :<Handle>:CONFiguration:PROFile:LIST?

Description Lists the parameters contained in the “profile”.

Return Value None

Example

```
:_test:CONF:PROF:LIST?
```

might return:

```
"PARAM1"
```


www.valuetronics.com

:MMEMory Subsystem

Syntax :<Handle>:MMEMory

The :MMEMory subsystem provides mass storage capabilities. The mass storage may be either an internal or external drive, e.g.: hard disc drive, floppy disc drive, etc.

:<Handle>:MMEMory:LIST?

Syntax :<Handle>:MMEMory:LIST? <"Absolute Path">

Parameters **<"Absolute Path">** Path of directory to be checked using "/" as path separator.

Return Value Returns a comma-separated list of the files in the specified directory.

If <"Absolute Path"> is an empty string, the root is listed. On Windows NT, the root is a virtual point below the drives. In other words, all drives are listed.

If <"Absolute Path"> does not point to a valid directory, an error is returned and the response is empty.

Example

```
:_test:MMEM:LIST? "C:/"
```

Returns a list of the root directory of drive 'C:':

```
("file_a","file_b","file_c","file_d")
```


www.valuetronics.com

:<Handle>:MMEMory:INFormation?

Syntax :<Handle>:MMEMory:INFormation? <"Absolute Path">

Parameters **<"Absolute Path">** Path of file object to be checked using "/" as path separator.

Return Value Returns detailed information about the specified file object. The response consist of an expression containing 5 comma-separated values like: <object type>, <mtime>, <size>, <attribute>, <db class>

<object type>	type of the object (quoted string)
<mtime>	modification time (numerical value)
<size>	size of object in bytes (numerical value)
<attributes>	attributes of the object (quoted string)
<db class>	type of a database object e.g. Setting, Segment, etc. (quoted string)

Returns the object type of drive 'C:'. "" (empty string) or 0 means not applicable.

Example

```
:_test:MMEM:INF? "C:"
```

might return:

```
("DRIVE",0,0,"","")
```


www.valuetronics.com

:<Handle>:MMEMory:SEGMent:LOAD

Syntax

:<Handle>:MMEMory:SEGMent:LOAD
<"FileName">| <(Expression)>[,ON| OFF][,<"SettingName">]

Parameters

First parameter

<"FileName"> or <(Expression)> Either specifies a file name containing segments or creates the segments according to the segment import and export syntax for loading it into the Agilent 81250 System's database (see *"Segment Import and Export Language" on page 301*).

Optional second parameter

[ON or OFF] ON is the default. It means that if the specified file already exists, it will be overwritten (replaced) with the new segment properties and data pattern.

Optional third parameter

[<"SettingName">] If a setting name is specified, the new or updated segments will be added as local segments to the specified setting. If the setting name is omitted, then the segments are placed in the global segment pool.

Description

The command is used to import new or updated segments into the Agilent 81250 System's database.

Return Value

None

Example

```
:_test:MMEM:SEGM:LOAD "A:\TRAFFIC.txt", OFF,"TEST1"
```


www.valuetronics.com

:<Handle>:MMEMory:SEGMent:SAVE

Syntax

Syntax 1:

:<Handle>:MMEMory:SEGMent:SAVE <"NewFileName ">[,<"SettingName">]
[,<"SegmentName ">]

Syntax 2:

:<Handle>:MMEMory:SEGMent:SAVE <"NewFileName">,
<"AnalyzerSegmentName">

Parameters

First parameter

<"NewFileName"> Specifies the file name under which the segments will be stored. The file format is according to the segment import and export syntax (see *"Segment Import and Export Language" on page 301*).

Optional second parameter (Syntax 1)

[<"SettingName">] If a setting name is specified, the segment is a local segment. With "" (empty string, just two quotes), the segment is assumed to be in the global segment pool.

Optional second parameter (Syntax 2)

[<"AnalyzerSegmentName">] Specifies a special segment `Analyzer/Capture.p` or `Analyzer/ErrMem.p`, where `p` is the port number of an output port.

Optional third parameter

[<"SegmentName">] Specifies a special segment. Absolute paths are allowed ("`LocalSegments/segm`", "`GlobalSegments/segm`", where `segm` corresponds to valid segments stored in the database under those nodes). The path names "`Segments/segm`" and "`Analyzer/segm`" are not valid.

Description

The command is used to export segments to an external file.

Return Value

None

Example

```
:_test:MMEM:SEGM:SAVE "C:\TRAFFIC.TXT","TEST1","PAYLOAD"
```

In this example, the segment "`PAYLOAD`" located in the local setting "`TEST1`" is saved to the file "`C:\TRAFFIC.TXT`".

```
:_test:MMEM:SEGM:SAVE "C:\TRAFFIC2.TXT","Analyzer/Capture.1"
```

In this example, the captured segment at port 1 ("`Analyzer/Capture.1`") is directly exported to the file "`C:\TRAFFIC2.TXT`".

www.valuetronics.com

:<Handle>:MMEMory:SEGMent:GET?

Syntax :<Handle>:MMEMory:SEGMent:GET? [<"SettingName">][,<"SegmentName">]

Parameters Optional first parameter

[<"SettingName">] If not specified, the global segment pool is addressed.

Optional second parameter

[<"SegmentName">] If not specified, all segments in either the global or local segment pool (depending on [<"SettingName">]) are addressed.

Return Value Returns an expression that contains the contents of the first-found segment(s) (vector definition, see *"Segment Import and Export Language" on page 301*). If the "SettingName" and the "SegmentName" are omitted, all segments are addressed in the global segment pool. If either "SettingName" or "SegmentName" is specified, a specific local segment is addressed.

Example

```
:_test:MMEMory:SEGM:GET? "TEST1","PAYLOAD"
```

might return:

```
Vector definition for "PAYLOAD" (see "Segment Import and Export
Language" on page 301)
```

:<Handle>:MMEMory:SETTing:NAME?

Syntax :<Handle>:MMEMory:SETTing:NAME?

Return Value Returns the current setting name.

Example

```
:_test:MMEM:SETT:NAME?
```

might return the following current setting name:

```
"TEST1"
```


www.valuetronics.com

:<Handle>:MMEMory:SETTing:LOAD

Syntax :<Handle>:MMEMory:SETTing:LOAD [<"SettingName">] [,<YDELay | NDELay | ODELay>]

Parameters Optional first parameter

[<"SettingName">] If not specified, the current setting is used. The setting at system power-up is "Default". If changes are stored with 'save-as', the current setting could be "TEST1".

YDELay Yes delay; load setting with cable delay parameters. This is the default.

NDELay No delay; load setting without cable delay parameters.

ODELay Only delay; load only cable delay parameters.

Description The command is used to load the current setting or a specified setting into the Agilent 81250 System's database.

Example

```
:_test:MMEM:SETT:LOAD "TEST1"
```

:<Handle>:MMEMory:SETTing:SAVE

Syntax :<Handle>:MMEMory:SETTing:SAVE [<"SettingName">]

Parameters **[<"SettingName">]** If not specified, the changes are saved into the current setting. After system power-up, the name of the current setting is "Default".

Description The command is used to save into either the current setting or into a new setting.

Return Value None

Example

```
:_test:MMEM:SETT:SAVE "TEST"
```


www.valuetronics.com

:<Handle>:MMEMory:SETTing:NEW

Syntax :<Handle>:MMEMory:SETTing:NEW

Description Resets and creates an 'untitled' environment. Starting from default values, a new setting can be created. To validate the new setting, it has to be 'saved' or 'saved-as' a into the Agilent 81250 System's database.

Return Value None

Example
```
:_test:MMEM:SETT:NEW
```

:<Handle>:MMEMory:SETTing:DELete

Syntax :<Handle>:MMEMory:SETTing:DELete <"SettingName">

Parameters **<"SettingName">** Setting to be deleted.

Description The command is used to delete the specified setting from the Agilent 81250 System's database.

Return Value None

Example
```
:_test:MMEM:SETT:DEL "TEST1"
```


www.valuetronics.com

:EDIT:SEGMent(*) Subsystem

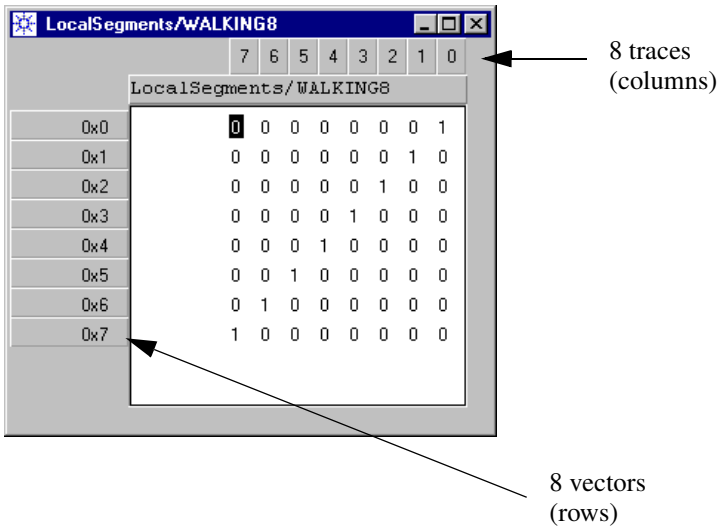
Syntax :<Handle>:EDIT:SEGMent(*)

The :EDIT:SEGMent(*) subsystem gives editing capabilities for creating new segments or updating existing segments. Data segments represent the pattern that the must be generate for output and the expected data or captured data segments for input. The generator and analyzers are grouped to ports.

As a convention, each row in the data segment is called a **vector**. Each vector represents the data pattern that is generated or received at the one clock cycle. Each column in the data segment is called a **trace**. This represents the data stream of one output or input connector. Therefore, the width of a vector corresponds to the number of traces in the current hardware configuration.

One **state** is the information stored in one cell. Every vector has as many states as traces in width is the segment. The valid states depend on the coding for the segment. For two-bit codings, up to four states are valid. For one-bit codings, two different states exist.

The following diagram shows the vector and trace representation in a data segment from the Graphical User Interface:



In order to work with one segment, you first have to open that segment. In return, you get an inspector number that is used for referencing that segment. This inspector number replaces the (*) symbol in the following calls. When no inspector number is given, the inspector number 1 is assumed by default.

www.valuetronics.com

Segments are stored in a segment pool, which is part of the system database. There is one segment pool with global scope and one segment pool with local scope.

- Segments in the local segment pools can only be accessed if the appropriate setting is loaded.
- Segments in the global segment pool can be accessed from any setting.

NOTE We recommend to set up the required data segments in the Segment Editor of the user interface and have these segments available in the system's database. In your program, you can open the segments.

Another possibility is to create the data segments using the Segment Import Language (see *"Segment Import and Export Language" on page 301*) and import the segments (see *":<Handle>:MMEMory:SEGMent:LOAD" on page 90*). This can ease your programming tasks and limits your time investment in coding.

:<Handle>:EDIT:SEGMent:OPEN?

Syntax :<Handle>:EDIT:SEGMent:OPEN? <"SegmentName">

Parameters **<"SegmentName">** The name of a segment that is stored in the Agilent 81250 System's database.

Description Opens the specified segment.

For input and output ports: If the segment name is prefixed with 'GlobalSegments/', the global pool is searched; if the segment name is prefixed with 'LocalSegments/', the current setting is searched. If only a segment name is supplied, first the current setting is searched. If no segment of the specified name is found, the global segment pool is searched.

For output ports: If the segment name is prefixed with 'Analyzer/', the instrument memory is searched for all 'Capture.p' and 'ErrMem.p' segments of all analyzer ports (p is the port number). In Compare and Acquire around Error mode 'Capture.p' and 'ErrMem.p', segments are available for each port. In Capture Data mode, only 'Capture.p' segments are available. In Error Rate Measurement, there are no segments.

www.valuetronics.com

Return Value If the segment was successfully opened, the query returns the index of the segment inspector allocated to the segment. If the query returns zero, the segment could not be opened.

NOTE At low system clock rates (low frequencies), it takes some time till the system is ready to give access to the data segments. So, it is recommended to check the system state by the command :SGEN:GLOB:SYST:STATe? and wait until FINished is returned (see *":<Handle>:SGENeral:GLOBal:SYSTem:STATe?" on page 151*). Then stop the system by the command :SGEN:GLOB:INIT:CONT OFF (see *":<Handle>:SGENeral:GLOBal:INITiate:CONTinuous" on page 150*). Now it is possible to open the data segments.

Example

```
:_test:EDIT:SEGM:OPEN? "PAYLOAD"
```

might return the following segment inspector index number:

```
3
```

As an example, the following command would save this segment under a new name:

```
:_test:EDIT:SEGM3:SAVE "PAYLOAD2"
```

:<Handle>:EDIT:SEGMent(*):SAVE

Syntax :<Handle>:EDIT:SEGMent(*):SAVE [<"SegmentName">]

Parameters **[<"SegmentName">]** Optional quoted string of the segment name with the target location for the new segment. The new segment can either be located in the "GlobalSegments/" pool or the "LocalSegments/" pool.

(*) Inspector number of the segment.

Description This command is used to save segments. When the "SegmentName" is given, it is a "save as"; if the "SegmentName" is omitted, then it is a "save" to the current segment name.

Example The following example saves the segment "test" that got the number "3" from the segment inspector (the number is the return value of the command :<hdl>:edit:segm:open?) as the new segment "payload" to the "GlobalSegments/" pool:

```
:_test:edit:segm3:save "GlobalSegments/payload"
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):DELete

Syntax :<Handle>:EDIT:SEGMent(*):DELete <"SegmentName">

Parameters **<"SegmentName">** The name of a segment stored in the Agilent 81250 System's database.

(*) The suffix (*) of SEGMent is ignored.

Description Deletes the specified segment. If the segment name is prefixed with 'GlobalSegments/' the global pool is searched; if the segment name is prefixed with 'LocalSegments/', then the current setting is searched. If only a segment name is supplied, the current setting is searched. If no segment of the specified name is found, the global segment pool is searched.

Example

```
:_test:EDIT:SEGM:DEL "PAYLOAD"  #Removes the segment "PAYLOAD"
                                #from the system's database
```

:<Handle>:EDIT:SEGMent(*):CREate?

Syntax :<Handle>:EDIT:SEGMent(*):CREate? <"SegmentName">, <SegmType>

Parameters **<"SegmentName">** The name of a new segment to be created.

<SegmType> Type of the segment to be created: MEMory | PRBS | PRWS | SFI5

(*) The suffix (*) of SEGMent is ignored.

Description Creates a new segment. The inspector number returned can then be used to refer to this segment. After being created, the segment is open and can be manipulated by means of the inspector number returned.

If the segment already exists, this segment is opened.

Only database segments can be created, not analyzer segments.

The path for the segment to be created must be absolute. The pool names "LocalSegments/" and "GlobalSegments/" can be used.

Segments that have been created during the current session but not yet saved in the database will not be listed when the MMEM subsystem is queried for them. However, the user can still work with them as normal segments.

www.valuetronics.com

Return Value Returns the inspector number of the segment just created.

Example
```
:_test:EDIT:SEGM:CRE? "LocalSegments/letstry", MEM
```
might return the following segment inspector index number:
```
3
```

:<Handle>:EDIT:SEGMent(*):EXISts?

Syntax :<Handle>:EDIT:SEGMent(*):EXISts? <"SegmentName">

Parameters **<"SegmentName">** The name of a segment.

(*) Inspector number of the segment.

Description Queries the existence of a segment. The segment might be an analyzer or a database segment.

Relative paths are only allowed for database segment. For other segments, the path must be absolute.

When a relative path is given, the LocalSegments pool is examined first. If the segment is not found there, the GlobalSegments pool is examined. If it is not found there, FALSE is returned

Return Value TRUE or FALSE

Example
```
:_test:EDIT:SEGM:EXIS? "walking8"
```
might return:
```
TRUE
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):RPATh?

Syntax :<Handle>:EDIT:SEGMent(*):RPATh? <"SegmentName">

Parameters **<"SegmentName">** The name of the segment (absolute or relative). This is an optional argument.

(*) Inspector number of the segment. This is an optional suffix.

Description Returns the absolute path for one segment.

To query for an open segment, you use the suffix for indicating its inspector number, but provide no arguments.

To query about a not open segment, use the argument for indicating the name of the segment. In this case, all suffixes provided will be ignored.

The name of the segment might be either an absolute or a relative path name. Of course, if an absolute path name is provided, the returned value will correspond to this argument.

When a relative path is given, the LocalSegments pool will be examined first. If the segment is not found there, the GlobalSegments pool will be examined before returning an error.

If neither argument nor suffix is provided, the inspector number 1 will be used by default.

Return Value **<"AbsolutePathName">** The absolute path name of the input segment.

Example

```
:_test:EDIT:SEGM:RPAT? "walking8"
```

might return:

```
"GlobalSegments/walking8"
```

```
:_test:EDIT:SEGM3:RPAT?
```

might return:

```
"LocalSegments/mysegment"
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):CLOSe

Syntax :<Handle>:EDIT:SEGMent(*):CLOSe

Parameters **(*)** Inspector number of the segment.

Description Closes the segment editor specified by the segment inspector index number without saving any changes made to the current segment name.

Example
```
:_test:edit:segm3:clos
```

:<Handle>:EDIT:SEGMent(*):PATTern:DATA

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:DATA
<trace1>,<vector1>,<trace2>,<vector2>,<stream>

Parameters

<trace1> Number of start trace.

<vector1> Number of start vector.

<trace2> Number of end trace.

<vector2> Number of end vector.

<stream> Data stream to be set in the segment (hexadecimal string).

(*) Inspector number of the segment.

Description Sets data in a bounded rectangle in the segment.

New data can only be set for database segments. Attempting to set new data in an analyzer segment will cause an error.

The input data stream (<stream>) is a string of hexadecimal characters. The length of this string must be at least as many hexadecimal characters as needed to provide all the information required for the bounded rectangle. The format of this string is the same as explained for *":<Handle>:EDIT:SEGMent(*):PATTern:DATA?" on page 102*.

If more characters are provided than required, they will be ignored. Also the trailing bits for the last character in every trace will be ignored.

Example
```
:_test:EDIT:SEGM3:PATT:DATA 0,0,3,2,"FFFF"
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PATTern:DATA?

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:DATA? <trace1>,<vector1>,<trace2>,<vector2>,<HEX|BIN>

Parameters

<trace1> Number of start trace.

<vector1> Number of start vector.

<trace2> Number of end trace.

<vector2> Number of end vector.

<HEX|BIN> Specifies the format of the data stream returned. This can be a bit stream (BIN) or its conversion into a hexadecimal string—as a sequence of ASCII characters—(HEX).

(*) Inspector number of the segment.

Description Retrieves data from a bounded rectangle in a segment.

The data is returned in a stream. The first bytes in the stream correspond to all the data belonging to the first trace, followed by the data corresponding to the next trace, and so on.

Data for every trace is packed in an exact number of bytes. For example, if 5 vectors for every trace have been queried and the coding is 2 bits long, 10 bits will be needed for every trace data. For these 10 bits, 2 bytes are needed, and the “empty” 6 bits will be returned as 0s. After that, the 2 bytes for the next trace will appear in the data stream.

Every byte is filled starting from the most significant bits, so when there are spare bits in one byte, the less-significant bits will not be used. For more information on the coding, please refer to *“:<Handle>:EDIT:SEGMent(*):PATTern:CODing?” on page 105*.

If the HEX format is selected, these bytes will be directly transformed into hexadecimal ASCII characters. Every byte needs two hexadecimal characters.

If the BIN format is selected, some additional information is required.

www.valuetronics.com

BIN Format

For BIN format, the returning stream will use the following format:

#abbbbcccccccccc where:

- a is the number of b's
- bbbb is the number of c's
- ccccccc is the raw data; in the GUI, this raw data is presented as ASCII characters.

Return Value

Returns the data stream corresponding to the region specified in the input.

Example

```
:_test:EDIT:SEGM3:PATT:DATA? 0,0,3,2,HEX
```

might return the following segment data:

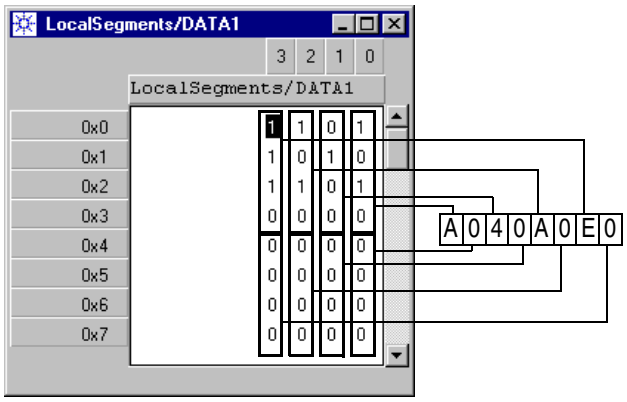
```
A040A0E0
```

```
:_test:EDIT:SEGM3:PATT:DATA? 0,0,3,2,BIN
```

might return the following segment data:

```
#9000000004 @ à
```

The following figure shows how the hex characters correspond to the segment data as shown in the user interface.



www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PATTern:CODing

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:CODing <"NewCoding">

Parameters **<"NewCoding">** A quoted string of the new segment coding.

(*) Inspector number of the segment.

Description Sets the symbol coding to be used in this segment. It is typically used for setting the coding of a newly created segment.

If you want to convert the coding of an existing segment, use the :EDIT:SEGM(*):MOD:CONV command instead. See *":<Handle>:EDIT:SEGMent(*):PATTern:MODify:CONVerse" on page 116* for more information.

New codings can only be set for database segments. Attempting to set a new coding in an analyzer segment will cause an error.

The only valid codings are "01" and "0 x1" (also "0 X1" is accepted).

- "01" results in a 1-bit coding for two states (low and high).
- "0 x1" results in a 2-bit coding for three states.

  The order of the characters "0 x1" is particularly important as it implies the underlying binary values. Starting with 0, the coding assumes increasing order from left to right. The following table shows the mapping of state characters to binary segment data:

State Character	Binary Segment Data
0	00
'blank'	n/ a
x	10
1	11

  For example, the sequence 001xx will be coded as 00 00 11 10 10.

Example

```
:_test:EDIT:SEGM3:PATT:CODing "01"
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PATTern:CODing?

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:CODing?

Return Value Returns the symbol coding used in this segment.

Example

```
:_test:EDIT:SEGM3:PATT:COD?
```

might return the following segment coding:

```
"01"
```

:<Handle>:EDIT:SEGMent(*):PATTern:ERRor:FIRSt?

Syntax :<Handle>:EDIT:SEGMent*:PATTern:ERRor:FIRSt? <VECT | TRAC>, [NORMal | RVS]

Parameters **[NORMal | RVS]** Defines the direction of the search. Default is NORMal.

- NORMal

  The search is started at trace 0, vector 0, and progresses in the incrementing trace, incrementing vector direction.

- RVS

  The search is started at trace n–1, vector 0, and progresses in the decrementing trace, incrementing vector direction.

Return Value **t, v** Returns the coordinates (trace, vector) of the first found error. Returns coordinates `-1,-1` if no error was found.

Example

```
_TEST:SEGM1:PATT:ERR:FIRS? VECT,NORM
```

might return:

```
2,7
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PATTern:ERRor:NEXT?

Syntax :<Handle>:EDIT:SEGMent*:PATTern:ERRor:NEXT? <trace>, <vector>, <VECT | TRAC>, [NORMal | RVS]

Parameters As in the ERRor:FIRSt? function, with the following additions:

<trace> Trace where the search starts.

<vector> Vector where the search starts.

Return Value See *":<Handle>:EDIT:SEGMent(*):PATTern:ERRor:FIRSt?" on page 105*.

Example

```
_TEST:SEGM1:PATT:ERR:NEXT? 2,4,VECT,NORM
```

might return:

```
2,7
```

:<Handle>:EDIT:SEGMent(*):PATTern:ERRor:PREVious?

Syntax :<Handle>:EDIT:SEGMent*:PATTern:ERRor:PREVious? <trace>, <vector>, <VECT | TRAC>, [NORMal | RVS]

Parameters As in the ERRor:FIRSt? function, with the following additions:

<trace> Trace where the search starts.

<vector> Vector where the search starts.

Return Value See *":<Handle>:EDIT:SEGMent(*):PATTern:ERRor:FIRSt?" on page 105*.

Example

```
_TEST:SEGM1:PATT:ERR:PREV? 2,63,VECT,NORM
```

might return:

```
4,23
```


www.valuetronics.com

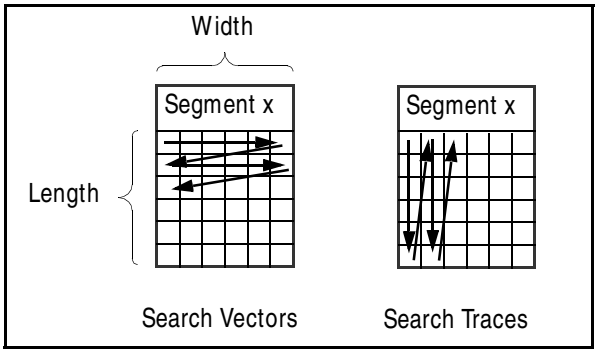
:<Handle>:EDIT:SEGMent(*):PATTern:FIND:FIRSt?

Syntax :<Handle>:EDIT:SEGMent*:PATTern:FIND:FIRSt? <data>, <VECT | TRAC>, [NORMal | RVS]

Parameters **<data>** Data sequence to be searched for. It can be one of the following:

- Binary sequence (default, with optional “B” or “b” after the sequence)
- Octal sequence (with mandatory “O” or “o” after the sequence)
- Hexadecimal sequence (with mandatory “H” or “h” after the sequence)

<VECT | TRAC> Defines if the data sequence is searched for in traces or vectors.



[NORMal | RVS] Defines if the direction of the search. Default is NORMal.

- NORMal

  The search is started at trace 0, vector 0, and progresses in the incrementing trace, incrementing vector direction.

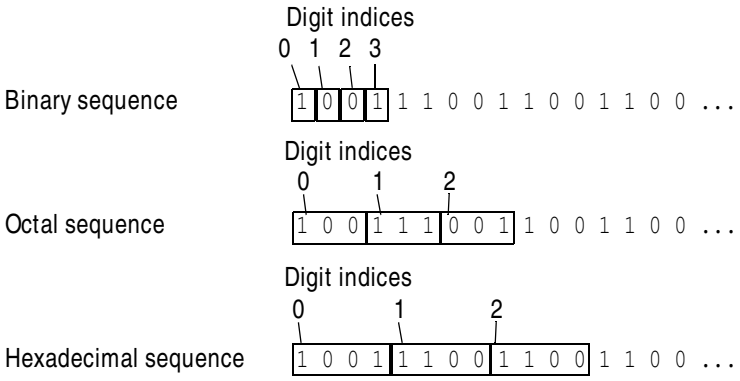
- RVS

  The search is started at trace n–1, vector 0, and progresses in the decrementing trace, incrementing vector direction.

Return Value **d, v** Returns the digit index and vector index (comma-separated) of the starting point of the first found instance of the data sequence. Returns coordinates `-1,-1` if the data sequence was not found.

www.valuetronics.com

The following graphic shows how the digit index and trace position are related.



The following can be used to calculate the trace position from the returned digit index (d):

- Binary sequence

  trace position = d
- Octal sequence

  trace position = 3 * d
- Hexadecimal sequence

  trace position = 4 * d

Example

```
:_test:EDIT:SEGM1:PATT:FIND:FIRS? "0101b", TRAC
:_test:EDIT:SEGM1:PATT:FIND:FIRS? "013h", TRAC
:_test:EDIT:SEGM1:PATT:FIND:FIRS? "037o", TRAC
```

might return:

```
1, 0
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PATTern:FIND:NEXT?

Syntax :<Handle>:EDIT:SEGMent*:PATTern:FIND:Next? <digit>, <vector>, <data>, <VECT | TRAC>, [NORMal | RVS]

Parameters As in the FIND:FIRSt? function, with the following additions:

<digit> Defines the starting trace point of the search.

- Binary sequence
  <digit> = trace point
- Octal sequence
  <digit>= trace point / 3
- Hexadecimal sequence
  <digit>= trace point / 4

<vector> Defines the vector where the search starts.

Return Value See *":<Handle>:EDIT:SEGMent(*):PATTern:FIND:FIRSt?" on page 107*.

:<Handle>:EDIT:SEGMent(*):PATTern:FIND:PREVious?

Syntax :<Handle>:EDIT:SEGMent*:PATTern:FIND:Prev? <digit>, <vector>, <data>, <VECT | TRAC>, [NORMal | RVS]

Parameters As in the FIND:FIRSt? function, with the following additions:

<digit> Defines the starting trace point of the search.

- Binary sequence
  <digit> = trace point
- Octal sequence
  <digit>= trace point / 3
- Hexadecimal sequence
  <digit>= trace point / 4

<vector> Defines the vector where the search starts.

Return Value See *":<Handle>:EDIT:SEGMent(*):PATTern:FIND:FIRSt?" on page 107*.

www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PATTern:LENGth?

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:LENGth?

Parameters (*) Inspector number of the segment

Return Value Returns the number of vectors of the segment.

Example

```
:_test:EDIT:SEGM3:PATT:LENG?
```

might return the following number of vectors of the specified segment:

```
80
```

:<Handle>:EDIT:SEGMent(*):PATTern:WIDTh?

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:WIDTh?

Parameters (*) Inspector number of the segment

Return Value Returns the numbers of traces of a segment.

Example

```
:_test:EDIT:SEGM3:PATT:WIDT?
```

might return the following number of traces of the specified segment:

```
8
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PATTern:MODify:COPY

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:MODify:COPY StartTrace, StartVector, EndTrace, EndVector

Parameters

StartTrace Number of start trace.

StartVector Number of start vector.

EndTrace Number of end trace.

EndVector Number of end vector.

(*) Inspector number of the segment.

Description Copies the bounded rectangle of data from the segment into the clipboard.

Example

```
:_test:edit:segm3:patt:mod:copy 0,2,10,15
```

:<Handle>:EDIT:SEGMent(*):PATTern:MODify:PASTE

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:MODify:PASTE StartTrace, StartVector

Parameters

StartTrace Number of start trace.

StartVector Number of start vector.

(*) Inspector number of the segment.

Description Pastes the data currently stored in the clipboard to the specified segment.

If the number of vectors or traces from the initial state (defined by the starting trace and vector) up to the segment borders is smaller than the size of the data stored in the clipboard, the exceeding data from the clipboard will be ignored.

Attempting to paste data in an analyzer segment will cause an error.

Example

```
:_test:EDIT:SEGM3:PATT:MOD:PASTE 0,76
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PATTern:MODify:FILL

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:MODify:FILL StartTrace, StartVector, EndTrace, EndVector, State

Parameters

StartTrace Number of start trace.

StartVector Number of start vector.

EndTrace Number of end trace.

EndVector Number of end vector.

State New value to be set in this bounded region.

(*) Inspector number of the segment.

Description Sets the bounded area of the segment with the same state value.

State is an unsigned integer used to index the segment's coding for getting the new state to be used.

For example, in coding "0 x1", the value of State might be an integer from 0 to 3. If a bigger integer is provided, the highest bits will be ignored and the least significant bits will be used to index the coding (for example, if "6" is provided, "2" will be used).

Attempting to fill a region in an analyzer segment will cause an error.

Example

```
:_test:EDIT:SEGM3:PATT:MOD:FILL 0,0,5,10,3
```


www.valuetronics.com

:< Handle> :EDIT:SEGMent(*):PATTern: MODify:INVert

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:MODify:INVert StartTrace, StartVector, EndTrace, EndVector

Parameters

StartTrace Number of start trace.

StartVector Number of start vector.

EndTrace Number of end trace.

EndVector Number of end vector.

(*) Inspector number of the segment.

Description Inverts the state information in the bounded area.

This command is only allowed in single-bit codings. Attempting to execute this command in multi-bit codings causes an error.

Attempting to invert data in a region in an analyzer segment causes an error.

For example, for the coding “01”, after this command execution, all 0’s will be changed into 1’s and all the 1’s will be changed into 0’s in the area.

Example

```
:_test:EDIT:SEGM3:PATT:MOD:INV 0,0,5,10
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PATTern:MODify:MIRRor

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:MODify:MIRRor StartTrace, StartVector, EndTrace, EndVector, <VECTor| TRACe>

Parameters

StartTrace Number of start trace.

StartVector Number of start vector.

EndTrace Number of end trace.

EndVector Number of end vector.

<VECTor|TRACe> Axis on which the mirror command will be performed.

(*) Inspector number of the segment.

Description Flips the specified block about horizontal (VECTor) or vertical axis (TRACe).

Attempting to flip data in one region in an analyzer segment will cause an error.

Example

```
:_test:EDIT:SEGM3:PATT:MOD:MIRR 0,0,5,10,VECT
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PATTern:MODify:INSert

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:MODify:INSert <Start>, <VECTor| TRACe>, <BEFore| AFTer>, HowMany

Parameters

<Start> Number of start trace or vector.

<VECTor|TRACe> What to insert, either traces or vectors.

<BEFore|AFTer> Insert before or after the start trace or vector.

HowMany Number of vectors or traces to be inserted.

(*) Inspector number of the segment.

Description Inserts new traces or vectors into a segment. The segment is resized according with the number of new traces or vectors added.

The new vectors or traces will be initialized to 0.

Attempting to insert new data in an analyzer segment will cause an error.

Example

```
_TEST:EDIT:SEGM3:PATT:MOD:INS 2,VECT,AFT,3
```

:<Handle>:EDIT:SEGMent(*):PATTern:MODify:DELete

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:MODify:DELete <Start>, <VECTor | TRACe>, <HowMany>

Parameters

<Start> Number of start trace or vector.

<VECTor | TRACe> What to delete, traces or vectors.

<HowMany> Number of vectors or traces to be deleted.

(*) Inspector number of the segment.

Description Deletes traces or vectors from a segment. The segment is resized according to the number of new traces or vectors deleted.

Attempting to delete data from an analyzer segment causes an error.

Example

```
:_test:EDIT:SEGM3:PATT:MOD:DEL 2,VECT,3
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PATTern:MODify:CONVerse

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:MODify:CONVerse NewCoding, Mapping

Parameters

NewCoding New coding for the segment.

Mapping Mapping between the old coding and the new coding.

(*) Inspector number of the segment.

Description This command changes the current coding for the segment and converts existing data in the segment according to the specified mapping.

The coding can be either “01” (binary) or “0x1” (binary with don’t care).

The mapping depends on the coding, and corresponds with the desired new values.

Attempting to convert data from an analyzer segment will cause an error.

Examples The following examples illustrate how the command works:

To change a “01”-coded segment to “0x1”-coded, where “0” -> “0”, “1” –> “x”:

```
EDIT:SEGM1:PATT:MOD:CONV "0x1", "0x"
```

To invert a “01”-coded segment:

```
EDIT:SEGM1:PATT:MOD:CONV "01", "10"
```

To change a “0x1”-coded segment to “01”-coded, where “0” -> “0”, “x” –> “1”, “1” –> “1”:

```
EDIT:SEGM1:PATT:MOD:CONV "01", "011"
```

To remap a “0x1”-coded segment, where “0” -> “x”, “x” –> “0”, “1” –> “1”:

```
EDIT:SEGM1:PATT:MOD:CONV "0x1", "x01"
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PATTern:MODify:SERialize

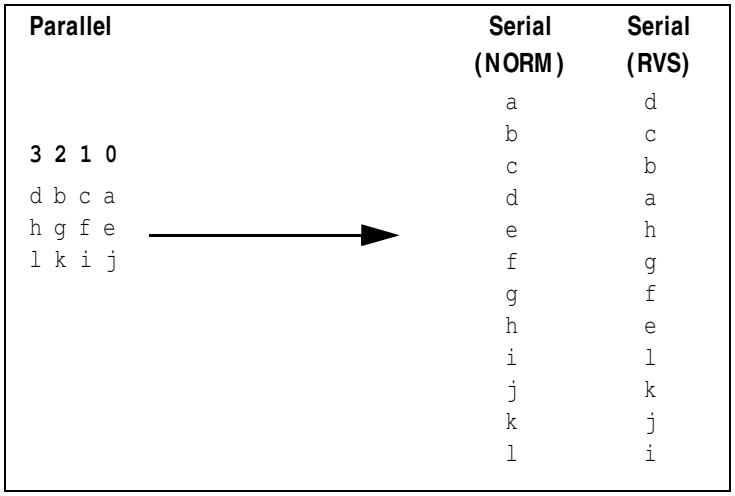
Syntax :<Handle>:EDIT:SEGMent(*):PATTern:MODify:SERialize [NORMal | RVS]

Parameters

NORMal Data is serialized in the order of the traces, from least significant bit to most significant bit (this is the default).

RVS Data is serialized in the reverse order (from most significant bit to least significant bit).

Description Serializes a parallel data stream. The optional direction parameter specifies the order in which the traces are to be considered. The following figure shows the differences between NORMal and ReVerSe serializing.



Example

```
:_test:EDIT:SEGM1:PATT:MOD:SER RVS
```


www.valuetronics.com

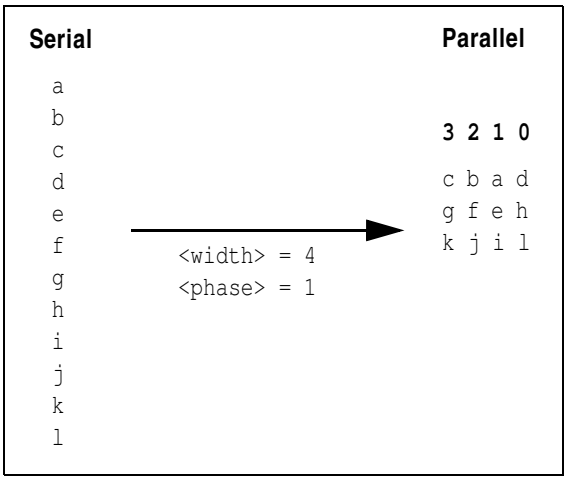
:<Handle>:EDIT:SEGMent(*):PATTern:MODify:DESerialize

Syntax :<Handle>:EDIT:SEGMent(*):PATTern:MODify:DESerialize <width> [,<phase>]

Parameters **<width>** Specifies the number of traces to be created for the parallel data stream.

[<phase>] (Optional) Specifies which trace is to be considered first (default is 0).

Description Deserializes a serial data stream. As shown below, <width> and <phase> determine how the serial data stream is deserialized to the parallel data stream.



Example

```
:_test:EDIT:SEGM1:PATT:MOD:DES 4, 1
```

:<Handle>:EDIT:SEGMent(*):PARAmeter:LENGth?

Syntax :<Handle>:EDIT:SEGMent(*):PARAmeter:LENGth?

Parameters **(*)** Inspector number of the segment.

Return Value Returns the length of the list of parameters available for this segment. The length of the list of parameters for analyzer segments is 0.

Example

```
:_test:EDIT:SEGM3:PARA:LENG?
= 1
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PARAmeter:LIST?

Syntax :<Handle>:EDIT:SEGMent(*):PARAmeter:LIST?

Parameters **(*)** Inspector number of the segment.

Return Value Returns the list of parameters available for this segment. The list of parameters for analyzer segments is always empty.

For the list of parameters, see *"Predefined Parameter Names" on page 311*.

Example

```
:_test:EDIT:SEGM3:PARA:LIST?
```

might return:

```
_Type
```

:<Handle>:EDIT:SEGMent(*):PARAmeter[:VALue]?

Syntax :<Handle>:EDIT:SEGMent(*):PARAmeter[:VALue]? <"ParamName">

Parameters **<"ParamName">** Name of the parameter.

For the list of parameters, see *"Predefined Parameter Names" on page 311*.

(*) Inspector number of the segment.

Return Value Returns the value for a specified parameter from the segment. Executing this command on analyzer segments causes an error.

Example

```
:_test:EDIT:SEGM3:PARA? "_Type"
```

might return:

```
= (MEMORY)
```


www.valuetronics.com

:<Handle>:EDIT:SEGMent(*):PARAmeter[:VALue]

Syntax :<Handle>:EDIT:SEGMent(*):PARAmeter[:VALue] <"ParamName">, <(Expression)>

Parameters **<"ParamName">** Name of the parameter. For the list of parameters, see *"Predefined Parameter Names" on page 311*.

<(Expression)> New value for the parameter.

(*) Inspector number of the segment.

Description Attempts to set the value for a specified parameter in the segment. Executing this command on analyzer segments causes an error.

Example
```
:_test:EDIT:SEGM3:PARA "_Type",(PRBS)
```

:<Handle>:EDIT:SEGMent(*):PARAmeter:REMove

Syntax :<Handle>:EDIT:SEGMent(*):PARAmeter:REMove <"ParamName">

Parameters **<"ParamName">** Name of the parameter. For the list of parameters, see *"Predefined Parameter Names" on page 311*.

(*) Inspector number of the segment.

Description Removes a parameter from the list of parameters in the segment. Executing this command on analyzer segments causes an error.

Example
```
:_test:EDIT:SEGM3:PARA:REM "_Type"
```

:<Handle>:EDIT:SEGMent(*):TYPE

Syntax :<Handle>:EDIT:SEGMent(*):TYPE <NewType>

Parameters **<NewType>** Available segment types: MEMory|PRBS|PRWS|SFI5.

(*) Inspector number of the segment.

Description Sets the segment type for the specified segment.

Example
```
:_test:EDIT:SEGM3:TYPE MEM
```


www.valuetronics.com

:< Handle> :EDIT:SEGM ent(*):TYPE?

Syntax :<Handle>:EDIT:SEGMent(*):TYPE?

Parameters **(*)** Inspector number of the segment.

Return Value Returns the segment type of the segment specified by the segment inspector index number (MEMory|PRBS|PRWS|SFI5).

Example

```
:_test:EDIT:SEGM3:TYPE?
```

might return the following segment type:

```
MEM
```

[:CGRoup(*)] Subsystem

:<Handle>[:CGRoup(*)]

This subsystem allows to control system parameters on clock group level. The following subsystems are available:

- *"[:CGRoup(*)]:MCLock Subsystem" on page 123*
- *"[:CGRoup(*)]:MODule(*) Subsystem" on page 124*
- *"[:CGRoup(*)]:MODule(*):CONNector(*) Subsystem" on page 127*
- *"[:CGRoup(*)][:SOURce]:TRIGger Subsystem" on page 132*

www.valuetronics.com

:<Handle>[:CGRoup(*)]:CINFormation?

Syntax :<Handle>[:CGRoup(*)]:CINFormation? <SHORt | DETailed>

Return Value This query returns the clock group configuration information. The response is an expression according to the following syntax:

<ClockGrpInfo>	::=	"(" <ModuleInfo> ("," <ModuleInfo>)* ")"
<ModuleInfo>	::=	"("<ProductNr> "," <SpeedClass> ["," <SerNr> "," <IDNr>] "," <ConnInfo> ("," <ConnInfo>)* ")"
<ConnInfo>	::=	"("<ProductNr> "," <SpeedClass> "," "Type" ["," <SerNr> "," <IDNr> "," <ConnNr>] ")"
<ProductNr>	::=	for example, "E4838A"
<SpeedClass>	::=	for example, 660
<Type>	::=	"ANALYZER" | "GENERATOR"
<SerNr>	::=	serial number (0 if not available)
<IDNr>	::=	identification number (0 if not available)
<ConnNr>	::=	"S1" | "D1" | "D2" S1: single frontend D1: first connector of a dual frontend D2: second connector of a dual frontend

The optional syntax elements only appear if DETailed is selected.

Example `:_test:CINF? SHOR`

might return, for example (formatted here only for readability):

```
((E4805B,660),
  (E4861A,2640,
    (E4863A,2640,ANALYZER),
    (E4862A,2640,GENERATOR)),
  (E4861A,2640,
    (E4863A,2640,ANALYZER),
    (E4862A,2640,GENERATOR)))
```


www.valuetronics.com

[:CGRoup(*)]:MCLock Subsystem

Syntax :<Handle>[:CGRoup(*)]:MCLock

This tree provides the commands which set or query the “master” system clock module in the Agilent 81250 System. It is possible to have more than one clock module installed, to have more than one “virtual” instrument. It is then possible to synchronize these “virtual” instruments, by defining one as the “master” clock generator.

:<Handle>[:CGRoup(*)]:MCLock:SOURce[:VALue]

Syntax :<Handle>[:CGRoup(*)]:MCLock:SOURce[:VALue] <ON | OFF>

Description Because it is possible to have more than one clock module in the Agilent 81250 System, this command controls which of the clock modules generates the “master” clock.

Example

```
:_test:CGR1:MCL:SOUR ON
```

:<Handle>[:CGRoup(*)]:MCLock:SOURce[:VALue]?

Syntax :<Handle>[:CGRoup(*)]:MCLock:SOURce[:VALue]?

Return Value State of the clock module of the specified clock group. When ON is returned, this is the “master” clock module in the Agilent 81250 System.

Example

```
:_test:cgroup1:mcl:sour?
```

might return:

```
ON
```


www.valuetronics.com

[:CGRoup(*)]:MODule(*) Subsystem

Syntax :<Handle>[:CGRoup(*)]:MODule(*)

This tree provides a set of commands which co-operate with a single module.

:<Handle>[:CGRoup(*)]:MODule(*):CINFormation?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):CINFormation? <SHORt | DETailed>

Return Value This query returns the module configuration information, including the frontends. The response is an expression according to the following syntax:

<ModuleInfo>	::=	"("<ProductNr> "," <SpeedClass> ["," <SerNr> "," <IDNr>] "," <ConnInfo> ("," <ConnInfo>)* ")"
<ConnInfo>	::=	"("<ProductNr> "," <SpeedClass> "," "Type" ["," <SerNr> "," <IDNr> "," <ConnNr>] ")"
<ProductNr>	::=	for example, "E4838A"
<SpeedClass>	::=	for example, 660
<Type>	::=	"ANALYZER" | "GENERATOR"
<SerNr>	::=	serial number (0 if not available)
<IDNr>	::=	identification number (0 if not available)
<ConnNr>	::=	"S1" | "D1" | "D2"

S1: single frontend
D1: first connector of a dual frontend
D2: second connector of a dual frontend

The optional syntax elements only appear if DETailed is selected.

Example `:_test:MOD2:CINF? DET`

might return for example (formatted here only for readability):

```
(E4861A,2640,"DE40701397","DE40701397/0",
  (E4863A,2640,ANALYZER,"DE40701316","DE40701316/0",S1),
  (E4862A,2640,GENERATOR,"DE40701738","DE40701738/0",S1))
```


www.valuetronics.com

:<Handle>[:CGRoup(*)]:MODule(*):TYPE?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):TYPE?

Returns the product number of the specified module in a quoted string.

Example
```
:_test:CGRoup1:MOD1:TYPE?
```
might return:
```
"4805B"
```

:<Handle>[:CGRoup(*)]:MODule(*):SLOT?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):SLOT?

Returns the slot number in which the specified module is located.

Example
```
:_test:CGR1:MOD1:SLOT?
```
might return:
```
3
```

:<Handle>[:CGRoup(*)]:MODule(*):FRAMe?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):FRAMe?

Returns the frame number to which the specified module belongs.

Example
```
:_test:CGR1:MOD1:FRAMe?
```
might return:
```
1
```

:<Handle>[:CGRoup(*)]:MODule(*):NAME?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):NAME?

Returns the name of the specified module.

Example
```
:_test:CGR1:MOD1:NAME?
```
might return:
```
"E4805B_01"
```


www.valuetronics.com

:<Handle>[:CGRoup(*)]:MODule(*):CNAMes?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):CNAMes?

Returns a quoted list of connector names.

The connector names are 7-digit quoted strings, defined according to connector, module and clock group. The following syntax is used:

`ggmmccc`

`gg` represents the clock group, `mm` represents the module, `ccc` represents the connector.

Example

```
:_test:CGR1:MOD1:CNAM?
```

might return:

```
"0102001","0102002","0102003","0102004"
```


www.valuetronics.com

[:CGRoup(*)]:MODule(*): CONNector(*) Subsystem

Syntax :<Handle>[:CGRoup(*)]:MODule(*):CONNector(*) subsystem

This tree provides administration and calibration commands available only at connector level.

Further commands are available at connector level (commands for timing, level, input and output parameters and for data formats). Because these commands are also available at port and terminal levels, they are described separately in sections for each parameter type.

:<Handle>[:CGRoup(*)]:MODule(*): CONNector(*):CINFormation?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CINFormation? <SHORt | DETailed>

Return Value This query returns the connector configuration information. The response is an expression according to the following syntax:

<ConnectorInfo>	::=	"("<ProductNr> "," <SpeedClass> "," "Type" ["," <SerNr> "," <IDNr> "," <ConnNr>] ")"
<ProductNr>	::=	for example, "E4838A"
<SpeedClass>	::=	for example, 660
<Type>	::=	"ANALYZER" | "GENERATOR"
<SerNr>	::=	serial number (0 if not available)
<IDNr>	::=	identification number (0 if not available)
<ConnNr>	::=	"S1" | "D1" | "D2"

S1: single frontend
D1: first connector of a dual frontend
D2: second connector of a dual frontend

The optional syntax elements only appear if DETailed is selected.

Example `:_test:MOD2:CONN1:CINF? DET`

might return for example:

```
(E4863A,2640,ANALYZER,"DE40701316","DE40701316/0",S1)
```


www.valuetronics.com

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):TYPE?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):TYPE?

Returns the product number of the frontend to which the connector belongs. This value is returned in a quoted string.

Example

```
:_test:CGR1:MOD1:CONN1:TYPE?
```

might return:

```
"E4863A"
```

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):NAME?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):NAME?

Returns the name of the specified connector as a 7-digit quoted string according to connector, module and clock group. The following syntax is used:

```
ggmmccc
```

`gg` represents the clock group, `mm` represents the module, `ccc` represents the connector.

Example

```
:_test:CGR1:MOD1:CONN1:NAME?
```

might return:

```
"0102001"
```

The above query returns the name of connector 1 in module 2 of clock group 1.

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):TNAMe?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):TNAMe?

Returns the terminal name to which this connector is connected. An empty “” string is returned if the connector is not connected.

Example

```
:_test:CGR1:MOD1:CONN1:TNAM?
```

might return:

```
"Data1"
```


www.valuetronics.com

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:CDELay

Syntax :<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:CDELay <Cable Delay>

Parameters **<NRf>** Cable Delay value.

Sets a cable delay for the specified connector to synchronize it with other signals applied to the DUT terminals.

NOTE Delay Auto Calibration must be performed before the cable delay can be set.

Example

```
:_test:MOD1:CONN1:CAL:CDEL 6.5e-9
```

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:CDELay?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:CDELay?

Return Value Returns the current cable delay for the specified connector of the specified module.

Example

```
:_test:MOD1:CONN1:CAL:CDEL?
```

might return:

```
6.500000E-009
```

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:ZDELay

Syntax :<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:ZDELay <Zero Delay>

Sets a zero delay for the specified connector of the specified module.

Parameters **<Zero Delay>** Zero delay value.

NOTE Delay Auto Calibration must be executed before zero delay can be set.

www.valuetronics.com

Related Commands
- *":<Handle>:SGENeral:GLOBal:CALibration:SELF DELay" on page 140*

Example
```
:_test:CGR1:MOD1:CONN:CAL:ZDEL 1.0e-9
```

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:ZDELay?

Syntax
:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:ZDELay?

Return Value
Returns the current zero delay for the specified connector.

Example
```
:_test:cgr1:mod1:conn1:cal:zdel?
```
might return:
```
1.000000E-009
```

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):SPECies[:VALue]?

Syntax
:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):SPECies?

Return Value
Returns a comma-separated list of the species states that are supported by the connector. This can be either ELECtric or OPTic.

Example
```
:_test:cgr1:mod1:conn1:spec?
```
might return:
```
ELEC,OPT
```

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):SPECies:STATe

Syntax
:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):SPECies:STATe <ELEC| OPT>

Parameters
<ELEC|OPT> Switches the connector into either electrical (ELEC) or optical (OPT) mode.

NOTE
This command can only be executed when the connector is not connected to any port.

Example
```
:_test:cgr1:mod1:conn1:spec:stat elec
```


www.valuetronics.com

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):SPECies:STATe?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):SPECies:STATe?

Return Value Returns the current species of the connector. Possible return values are electrical (ELEC) or optical (OPT).

Example

```
:_test:cgr1:mod1:conn1:spec:stat?
```

might return:

```
ELEC
```


www.valuetronics.com

[:CGRoup(*)][:SOURce]:TRIGger Subsystem

:<Handle>[:CGRoup(*)][:SOURce]:TRIGger

This command subsystem is used to set the trigger or strobe output parameters.

:< Handle> [:CGRoup(*)][:SOURce]:TRIGger:DELay

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:DELay <NRf>

The delay of the trigger output signal can be varied with this command. For frequencies <= 330 MHz in the range of 0 to 360 ns, for frequencies >330 MHz in the range of 0 to 3.3 ns

Example

```
:_test:CGR1:TRIG:DEL 2e-9
```


www.valuetronics.com

:<Handle>[:CGRoup(*)][:SOURce]:TRIGger:DELay?

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:DELay?

Return Value Returns the current delay of the trigger output signal.

Example
```
:_test:CGR1:TRIG:DEL?
```
might return:
```
2.000000E-009
```

:<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MUX

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MUX <NRf>

Parameters **<NRf>** MUX factor = frequency multiplier factor for the individual connector

In addition to the "system" MUX factor (Segment Resolution) it is possible to set the frequency multiply factor for the trigger output, as for individual connectors, to generate multiples or fractions of 2 of the frequency of the data streams, like f/2 or 2f etc. The default value is 1, or 1/4 if the system is equipped with E4861A modules only. Valid values are:

0.0625 (1/16), 0.125 (1/8), 0.25 (1/4), 0.5 (1/2), 1, 2, 4, 8, 16

The range of this parameter depends on the value of the "system" MUX factor (Segment Resolution). See *":<Handle>:SGENeral:GLOBal:MUX" on page 143*.

Example
```
:_test:CGR1:TRIG:MUX 2
```

:<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MUX?

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MUX?

Return Value The MUX factor associated with the trigger output.

www.valuetronics.com

Example `:_test:CGR1:TRIG:MUX?`

might return:

`2`

:< Handle> [:CGRoup(*)][:SOURce]:TRIGger: VOLTage[:LEVel][:IM M ediate]:HIGH

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]: HIGH <NRf>

Parameters **<NRf>** The upper voltage level of the trigger output signal.

Example `:_test:CGR1:TRIG:VOLT 2`

:< Handle> [:CGRoup(*)][:SOURce]:TRIGger: VOLTage[:LEVel][:IM M ediate]:HIGH?

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]: HIGH?

Returns the upper voltage level of the trigger output signal.

Example `:_test:CGR1:TRIG:VOLT?`

might return:

`2.000000E+000`

:< Handle> [:CGRoup(*)][:SOURce]:TRIGger: VOLTage[:LEVel][:IM M ediate]:LOW

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:LOW <NRf>

Parameter **<NRf>** The lower voltage level of the trigger output signal.

Example `:_test:CGR1:TRIG:VOLT:LOW -1`

www.valuetronics.com

:< Handle> [:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:LOW?

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:LOW?

Returns the lower voltage level of the trigger output signal.

Example

```
:_test:CGR1:TRIG:VOLT:LOW?
```

might return:

```
-1.000000E+000
```

:< Handle> [:CGRoup(*)][:SOURce]:TRIGger:TVOLtage

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:TVOLtage <NRf>

Specifies the termination voltage of a trigger output connector. The –2.1 V to +3.1 V.

Example

```
:_test:CGROUP1:SOURCE:TRIGGER:TVOLTAGE -2
```

:< Handle> [:CGRoup(*)][:SOURce]:TRIGger:TVOLtage?

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:TVOLtage?

Return Value Returns the current termination voltage of the trigger output connector.

Example

```
:_test:CGROUP1:SOURCE:TRIGGER:TVOLTAGE?
```

might return:

```
-2.000000E+000
```

:< Handle> [:CGRoup(*)][:SOURce]:TRIGger:IMPedance:EXTernal

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:IMPedance:EXTernal <NRf>

Description The external termination impedance can be specified by this command. Any negative value is interpreted as into “open”.

www.valuetronics.com

Example `:_test:CGR1:TRIG:IMP 500`

:< Handle> [:CGRoup(*)][:SOURce]:TRIGger:IMPedance:EXTernal?

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:IMPedance:EXTernal?

Return Value Returns the current programmed external termination impedance (load impedance).

Example `:_test:CGR1:TRIG:IMP?`

might return:

`5.000000E+002`

:< Handle> [:CGRoup(*)][:SOURce]:TRIGger:MODE

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MODE SEQuencer | CGENeration

Description This command is used to select the source of the trigger output. It is possible to derive the source of the trigger signal from the sequencer circuit (SEQuencer) or from the clock generation circuit (CGENeration).

Example `:_test:CGROUP1:TRIGGER:MODE SEQ`

:< Handle> [:CGRoup(*)][:SOURce]:TRIGger:MODE?

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MODE?

Return Value Returns the actual source used for the trigger signal.

Example `:_test:CGR1:TRIG:MODE?`

might return:

`SEQ`

www.valuetronics.com

:SGENeral Subsystem

Syntax :<Handle>:SGENeral

The :SGENeral subsystem represents the General Scheme. It represents a DUT with several ports. A port represents a collection of input or output connections (terminals). With these commands, you model your specific setup by defining data or pulse ports, naming the terminals and connecting the terminals of the DUT with the output and input resources of the Agilent 81250 System. The used virtual instrument of the Agilent 81250 System is represented by the *"[:CGRoup(*)] Subsystem" on page 121*.

The following subsystems are available:

- *":SGENeral:INFormation Subsystem" on page 138*
- *":SGENeral:GLOBal Subsystem" on page 139*
- The :SGENeral:PDATa(*) and :SGENeral:PPULse(*) subsystems allow to specify timing, level, input and output parameters and data formats on port and terminal levels. Furthermore, port and terminal administration commands are available for pulse and data ports.

  The :SGENeral:PDATa(*) subsystem also provides commands for error analysis.

  These commands are described separately in sections for each parameter type or command group.
- The :<Handle>:SGENeral:CONNect subsystem provides commands for connector administration for pulse and data ports. They are available at port and terminal level and are described in *"Connector Administration Commands" on page 290*.

www.valuetronics.com

:SGENeral:INFormation Subsystem

Syntax :<Handle>:SGENeral:INFormation

This subsystem provides information about the general scheme.

:< Handle> :SGENeral:INFormation: PCLasses?

Syntax :<Handle>:SGENeral:INFormation:PCLasses?

Return Value Returns the list of implemented port classes in a comma-separated quoted string list.

Example `:_test:SGENeral:INFormation:PCLasses?`

returns:

`"DATA","PULSE"`

www.valuetronics.com

:SGENeral:GLOBal Subsystem

Syntax :<Handle>:SGENeral:GLOBal

This subsystem provides commands to modify system parameters. These commands affect the entire system including any frames connected via MXI.

In addition to the top-level commands, this subsystem contains further subsystems:

- *":SGENeral:GLOBal:CONFigure Subsystem" on page 147*
- *":SGENeral:GLOBal:INITiate:CONTinuous Subsystem" on page 150*
- *":SGENeral:GLOBal:SYSTem Subsystem" on page 151*
- *":SGENeral:GLOBal:SEQuence Subsystem" on page 152*
- *":SGENeral:GLOBal:TRIGger Subsystem" on page 168*
- *":SGENeral:GLOBal:ARM Subsystem" on page 173*

:< Handle> :SGENeral:GLOBal:CONNect

Syntax :<Handle>:SGENeral:GLOBal:CONNect <ON | OFF>

Description Connects or disconnects all enabled connectors of the Agilent 81250 System at the same time.

Example

```
:_test:SGEN:GLOB:CONN OFF
```

:< Handle> :SGENeral:GLOBal:CONNect?

Syntax :<Handle>:SGENeral:GLOBal:CONNect?

Return Value Returns the actual connection status of the enabled connectors.

Example

```
:_test:SGEN:GLOB:CONN?
```

might return:

```
OFF
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal:CALibration: SELF DELay

Syntax :<Handle>:SGENeral:GLOBal:CALibration:SELF DELay

Description Any time frontends were exchanged in a module, the system must be autocalibrated after a 30-minute warmup period. Delay Auto Calibration must be performed before Zero Adjustment or Cable Adjustment.

Related Commands

- *":CALibration:CDELay" on page 285*
- *":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration: ZDELay" on page 129*

Example

```
:_test:SGEN:GLOB:CAL:SELF DEL
```

:< Handle> :SGENeral:GLOBal:DOFFset

Syntax :<Handle>:SGENeral:GLOBal:DOFFset <DelayOffset>

Parameters **<DelayOffset>** A numeric value out of the specified range, for example, 10e-9.

This command specifies an offset value to the fixed delay for all connectors. So, it is possible to set negative delays for individual connectors to achieve setup and hold times behavior.

Example

```
:_test:SGEN:GLOB:DOFF 10e-9
```

:< Handle> :SGENeral:GLOBal:DOFFset?

Syntax :<Handle>:SGENeral:GLOBal:DOFFset?

Return Value The current set delay offset.

Example

```
:_test:SGEN:GLOB:DOFF?
```

might return:

```
1.000000E-008
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal:FETCh:ERRor:ANY?

Syntax :<Handle>:SGENeral:GLOBal:FETCh:ERRor:ANY?

Description The query returns 0 for no errors, or 1 if an error was found in the system. This can be used to increase program speed. Uploading of memory segments can be completely avoided, when it is known that there are no errors at all.

Example
```
:_test:SGEN:GLOB:FETC:ERR:ANY?
```
might return:
```
0
```

:< Handle> :SGENeral:GLOBal:PERiod

Syntax :<Handle>:SGENeral:GLOBal:PERiod <Period>

Parameters **<Period>** A numeric value out of the specified range, e.g. 10e-9.

Sets the period of the Agilent 81250 System.

NOTE The period at an individual connector depends on the frequency multiply factor chosen for this connector.

Example
```
:_test:SGEN:GLOB:PER 10e-9
```

:< Handle> :SGENeral:GLOBal:PERiod?

Syntax :<Handle>:SGENeral:GLOBal:PERiod?

Return Value The currently set clock period.

Example
```
:_test:SGEN:GLOB:PER?
```
might return:
```
1.000000E-008
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal:FREQuency

Syntax :<Handle>:SGENeral:GLOBal:FREQuency <Frequency>

Parameters **<Frequency>** A numeric value out of the specified range, e.g. 100E6.

Sets the frequency of the Agilent 81250 System.

NOTE The frequency at an individual connector depends on the frequency multiply factor chosen for this connector.

Example

```
:_test:SGEN:GLOB:FREQ 100E6
```

:< Handle> :SGENeral:GLOBal:FREQuency?

Syntax :<Handle>:SGENeral:GLOBal:FREQuency?

Return Value The currently set clock frequency.

Example

```
:_test:SGEN:GLOB:FREQ?
```

might return:

```
1.000000E008
```


www.valuetronics.com

:<Handle>:SGENeral:GLOBal:MUX?

Syntax :<Handle>:SGENeral:GLOBal:MUX?

Return Value The current 'MUX' factor (Segment Resolution).

See ":<Handle>:SGENeral:GLOBal:MUX" on page 143 for details.

Example

```
:_test:SGEN:GLOB:MUX?
```

might return:

```
4
```

:<Handle>:SGENeral:GLOBal:MUX

Syntax :<Handle>:SGENeral:GLOBal:MUX <MUX factor>

Parameters **<MUX factor>** MUX factor equals the Segment Resolution. With this command, the global settings are defined. The Segment Resolution defines the Frequency Multiplier Range, the min/max System Clock Rate and the Memory Depth of the data stream.

For E4832A modules, valid values for the MUX Segment Resolution are 1, 2, 4, 8, and 16.

For E4861A modules, valid values are 16, 32, and 64.

For E4861B modules, valid values are 1, 2, 4, 8, 16, 32, 64, and 128.

Different values will be rounded off/on to the next valid value. Each connector can be set to an individual frequency multiply factor, see the timing parameter command *":MUX" on page 196*.

System Clock Frequency Mbit/s	Segment Resolution bits	Memory Depth bits	Possible Frequency Multipliers
≤ 42.1875	1	131,008	1, 2, 4, 8, 16
≤ 84.375	2	262,016	1/2, 1, 2, 4, 8
≤ 168.750	4	524,032	1/4, 1/2, 1, 2, 4
≤ 337.500	8	1,048,064	1/8, 1/4, 1/2, 1, 2
≤ 675.000	16	2,097,152	1/16, 1/8, 1/4, 1/2, 1

www.valuetronics.com

The E4861A module has a memory capacity of up to 8 MB per channel.

System Clock Frequency Mbit/ s	Segment Resolution bits	Memory Depth bits	Possible Frequency Multipliers
337.500 – 675.000	16	2,097,152	1, 2, 4
≤ 1,350.000	32	4,194,304	1/ 2, 1, 2
≤ 2,700.000	64	8,388,608	1/ 4, 1/ 2, 1

The E4861B module has a memory capacity of up to 16 MB per channel.

System Clock Frequency (Mbit/ s)	Segment Resolution (bits)	Memory Depth (bits)	Possible Frequency Multipliers
20.834 to 41.666	1	131,072	1, 2, 4, 8, 16, 32, 64, 128
≤ 82.333	2	262,144	1/ 2, 1, 2, 4, 8, 16, 32, 64
≤ 166.666	4	524,288	1/ 4, 1/ 2, 1, 2, 4, 8, 16, 32
≤ 333.333	8	1,048,576	1/ 8, 1/ 4, 1/ 2, 1, 2, 4, 8, 16
≤ 666.666	16	2,097,152	1/ 16, 1/ 8, 1/ 4, 1/ 2, 1, 2, 4, 8
≤ 1,333.333	32	4,194,304	1/ 32, 1/ 16, 1/ 8, 1/ 4, 1/ 2, 1, 2, 4
≤ 2,700.000	64	8,388,608	1/ 64, 1/ 32, 1/ 16, 1/ 8, 1/ 4, 1/ 2, 1, 2
≤ 3,350.000	128	16,777,216	1/ 128, 1/ 64, 1/ 32, 1/ 16, 1/ 8, 1/ 4, 1/ 2, 1

The E4866A and E4867A modules have a memory capacity of up to 32 MB per channel.

System Clock Frequency (Gbit/ s)	Segment Resolution (bits)	Memory Depth (bits)	Possible Frequency Multipliers
9.5 to 10.8	256	33,554,432	1

www.valuetronics.com

The desired system clock frequency determines the minimum segment resolution. As long as the segment resolution is less than 16 (or 64 for E4861A modules), it is possible to choose a higher general segment resolution. This increases the usable memory and simultaneously decreases the sequencer clock frequency.

NOTE For E4832A modules, the delay vernier and the functions for automatic analyzer sampling delay adjustment require that the minimum segment resolution of the desired system clock frequency is used.

For more details, see *"Frequency Multiplier and Segment Resolution"* in the *System User Guide*.

The relationships between the different values are:

Segment Resolution	Frequency Multiplier Range (FMR)[a]	Memory Depth[b]	System Clock Rates
1 bit (=1)	1, 2, 4, 8, 16	128 Kbit	≤ 42.1875 MHz
2 bits (=2)	1/ 2, 1, 2, 4, 8	256 Kbit	≤ 84.375 MHz
4 bits (=4)	1/ 4, 1/ 2, 1, 2, 4	512 Kbit	≤ 168.75 MHz
8 bits (=8)	1/ 8, 1/ 4, 1/ 2, 1, 2	1 Mbit	≤ 337.5 MHz
16 bits (=16)	1/ 16, 1/ 8, 1/ 4, 1/ 2, 1	2 Mbit	≤ 675 MHz

[a] This is the range of multiples and fractions that can be used at individual connectors. If you have most of your signals at 40 MHz and your pattern lengths are less then 64 Kbit, then you can choose segment resolution 1. You have the chance to set individual connectors to a multiple of this general setting. For example, selecting 16 as the multiply factor for a connector gives you 1 Mbit memory depth and 640 MHz with a segment resolution of 16.

[b] Subtract 32 x segment resolution, as this memory space is occupied by a 2^5-1 PRxS and the sequencing initialization.

Example

```
:_test:SGEN:GLOB:MUX 4
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal:PLL:STATe?

Syntax :<Handle>:SGENeral:GLOBal:PLL:STATe?

Description Determines the PLL (Phase Locked Loop) state. The PLL circuit locks on external clock signals. PLL:STATe indicates whether the internal and external signals are locked (that is, that they have the same period and phase).

Return Value **LOCKed** Indicates that the internal and external signals are locked.

UCDR (**U**nlocked **C**lock **D**ata **R**ecovery) If this value is returned, then the clock data recovery (CDR) circuit of the E4869 DEMUX module could not synchronize on the incoming data stream. As the generated clock signal is undefined, it is useless to continue the test. You may consider using an external clock.

Any other return value indicates that the internal and external signals are not locked. In this case, you can do the following to identify the problem:

- Check the physical connection to the module
- Check the level parameter or the frequency of the supplied signal

  The supplied signal should be roughly equal to the internal frequency of the module. You can check this in the GUI.

Example

```
:_test:SGEN:GLOB:PLL:STAT?
```

might return:

```
LOCK
```


www.valuetronics.com

:SGENeral:GLOBal:CONFigure Subsystem

Syntax :<Handle>:SGENeral:GLOBal:CONFigure

In accordance with the SCPI standard, this subsystem is used to perform measurements. The Agilent 81250 System offers three measurement modes:

- The Capture Data mode
- The Error Rate Measurement mode
- The Compare and Acquire around Error mode

:< Handle> :SGENeral:GLOBal:CONFigure?

Syntax :<Handle>:SGENeral:GLOBal:CONFigure?

Return Value Returns the last measurement mode setting in a quoted string (“ECO FAIL”, “ECO ONES”, “ECO ZER”, “ECAP <number of stop bits>”).

Example

```
:_test:SGEN:GLOB:CONF?
```

might return:

```
"ECO FAIL"
```

:< Handle> :SGENeral:GLOBal:CONFigure:CAPTure

Syntax :<Handle>:SGENeral:GLOBal:CONFigure:CAPTure

Description In the CAPTure measurement mode, it is possible to hook up analyzer channels at the output or input of the DUT (Device Under Test), capture (acquire) the data stream and then use the captured data stream as stimulus data or just to see what happens.

Example

```
:_test:SGEN:GLOB:CONF:CAPT
```


www.valuetronics.com

:<Handle>:SGENeral:GLOBal:CONFigure[:ECOunt]

Syntax :<Handle>:SGENeral:GLOBal:CONFigure[:ECOunt][FAILures | ONESfailed | ZEROsfailed]

In the Error Rate (Error COunt) measurement mode, an expected data segment (or more) is compared and the failed bits are counted. To read the results, see *":FETCh[:ECOunt]?" on page 293*.

If the argument is omitted, the default mode is count all FAILures.

Example

```
:_test:SGEN:GLOB:CONF ONES
```

:<Handle>:SGENeral:GLOBal:CONFigure:ECAPture

Syntax :<Handle>:SGENeral:GLOBal:CONFigure:ECAPture[<StopBits>]

Range/ Value Coupling On E4832A and E4861A modules, there is no restriction for the frequency multiplier.

The frequency multiplier can be queried via CGR:MOD:CONN:MUX?

The segment resolution can be queried via SGEN:GLOB:MUX?

Description In the Error CAPture mode, the incoming data stream is compared to a stream of expected data. After compare, the resulting data stream is held in the internal memory ready to be displayed or to be saved to the database. An additional parameter can be specified called <StopBits>. This is a number of cycles that can be specified to stop the analyzer after an error occurs. If the parameters are omitted following default values are assumed: StopBits= 32768.

For systems housing E4832A or E4861A modules, the minimum number of stop bits is 976.

Note that the value is always rounded up internally because of the resolution. Note that because of clock uncertainty, a few additional bits (depending on resolution) are recorded.

Example

```
:_test:SGEN:GLOB:CONF:ECAP 1024
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal:CONFigure:CCAPture

Syntax :<Handle>:SGENeral:GLOBal:CONFigure:CCAPture

Range/ Value Coupling On E4832A and E4861A modules there is no restriction for the frequency multiplier.

The frequency multiplier can be queried via CGR:MOD:CONN:MUX?

The segment resolution can be queried via SGEN:GLOB:MUX?

Reset Value After reset the system is in the ECO mode.

Description Sets the instrument to Compare and CAPture mode. In this mode, the incoming data stream is compared to a stream of expected data. Errors can be handled via event handling, for example, a deferred branch to the sequence end.

Example

```
:_test:SGEN:GLOB:CONF:CCAP
:_test:SGEN:GLOB:CONF?
= CCAP
```


www.valuetronics.com

:SGENeral:GLOBal:INITiate: CONTinuous Subsystem

Syntax :<Handle>:SGENeral:GLOBal:INITiate:CONTinuous

This subsystem controls the initialization of the trigger subsystem. In other words, these commands are used to start or stop the instrument (system).

:< Handle> :SGENeral:GLOBal:INITiate: CONTinuous

Syntax :<Handle>:SGENeral:GLOBal:INITiate:CONTinuous <ON | OFF>

Parameters **<ON | OFF>** ON starts the system.

This command starts/stops the system.

Example

```
:_test:SGEN:GLOB:INIT:CONT ON
```

:< Handle> :SGENeral:GLOBal:INITiate: CONTinuous?

Syntax :<Handle>:SGENeral:GLOBal:INITiate:CONTinuous?

Return Value The current state of the system.

Example

```
:_test:SGEN:GLOB:INIT:CONT ON
...
:_test:SGEN:GLOB:INIT:CONT?
ON
:_test:SGEN:GLOB:INIT:CONT OFF
...
:_test:SGEN:GLOB:INIT:CONT?
OFF
```


www.valuetronics.com

:SGENeral:GLOBal:SYSTem Subsystem

Syntax :<Handle>:SGENeral:GLOBal:SYSTem

The system is started or stopped by the previous command.

It is also possible that a system is stopped for example by the actual sequence, for example, if the sequence finishes before a stop was initiated by the user.

:< Handle> :SGENeral:GLOBal:SYSTem:STATe?

Syntax :<Handle>:SGENeral:GLOBal:SYSTem:STATe?

Return Value Returns the current state of the system.

The system is started or stopped by the previous command. It is also possible that a system is stopped for example by the actual sequence. for example, if the sequence finishes before a stop was initiated by the user.

The possible return values are:

FINished	The system has finished the sequence to generate
HALTed	The system is halted due to external gate or external stop
PROGram	The system is in the programming state (all parameters can be changed)
RUNNing	The system is in the running state (a data stream is generated)
SYNChronizing	The system is currently synchronizing on the data pattern

Example

```
:_test:SGEN:GLOB:SYST:STAT?
```

might return:

```
PROG
```


www.valuetronics.com

:SGENeral:GLOBal:SEQuence Subsystem

Syntax :<Handle>:SGENeral:GLOBal:SEQuence

This subsystem includes all commands related to sequencing. The functionality presented here is shown in the GUI as “Sequence Editor” and the “Prepare” Button.

It is recommended to use the GUI to set up and modify sequences. Afterwards, the sequence can be queried via the Command Line window. The resulting sequence or event expressions can then be integrated into remote programs via cut and paste.

Hierarchical Expressions

Sequences and Events use hierarchical expressions to encode information. The Syntax of an hierarchical expression is presented in EBNF notation (see ISO 14977):

- " or ' are quote characters, everything enclosed by quotes is treated “as is”.
- * is a repetition symbol, for example, 3 * “a” means aaa
- | separates alternative definitions
- ::= is the defining symbol
- ; ends a definition
- ( ) groups a definition
- [ ] encloses optional nodes
- { } encloses repetitive rules

www.valuetronics.com

:< Handle> :SGENeral:GLOBal:SEQuence[:VALue]

Syntax :<Handle>:SGENeral:GLOBal:SEQuence[:VALue] <(Expression)>

Parameters **<(Expression)>** hierarchical expression, syntax as described below

The sequence-expression is a simple programming language, defined by the following EBNF:

Sequence-Expression	::=	"(1.0," Label "," Block-Expression-1 ")" | "(2.0," Label "," Block-Expression-2 ")" | "(3.0," Label "," Block-Expression-3 ")" ;
Block-Expression-1	::=	Simple-Expression-1 | Sequential-Expression-1 | Loop-Expression-1 ;
Simple-Expression-1	::=	"(BLOCK," Trigger "," Vectors { "," Segment } ")" ;
Sequential-Expression-1	::=	"(SEQ," Block-Expression-1 { "," Label "," Block-Expression-1 } ")" ;
Loop-Expression-1	::=	"(LOOP" (1|2|3|4|5) "," Trigger "," Iterations "," Block-Expression-1 ")" ;
Block-Expression-2	::=	Simple-Expression-2 | Sequential-Expression-2 | Loop-Expression-2 ;
Simple-Expression-2	::=	"(BLOCK," Trigger "," Vectors ",0," VXI-Triggers-2 "," React-Expressions-2 { "," Segment ")" ;
Sequential-Expression-2	::=	"(SEQ," Block-Expression-2 { "," Label "," Block-Expression-2 } ")" ;
Loop-Expression-2	::=	"(LOOP" (1|2|3|4|5) "," Trigger "," Iterations "," VXI-Triggers-2 "," Block-Expression-2 ")" ;
React-Expressions-2	::=	"(React-Expression-2 { "," React-Expression-2 } ")" ;
React-Expression-2	::=	"(" Event-2 "," Goto-2 "," Trigger "," VXI-Triggers-2 ")" ;
Segment	::=	Segment-Name ",0,0" ;
Block-Expression-3	::=	Simple-Expression-3 | Sequential-Expression-3 | Loop-Expression-3 ;
Simple-Expression-3	::=	"(BLOCK," Trigger "," Vectors ",0," VXI-Triggers-2 "," React-Expressions-3 { "," Segment } ")" ;

www.valuetronics.com

```
Sequential-Expression-3  ::=  "(SEQ," Block-Expression-3 { "," Label ",
                              "Block-Expression-3 } ")" ;
Loop-Expression-3        ::=  "(LOOP" (1|2|3|4|5) "," Trigger ","
                              Iterations "," VXI-Triggers-2 ","
                              Block-Expression-3 ")" ;
React-Expressions-3      ::=  React-Expressions-2 | Sync-Expression-
                              3;
Sync-Expression-3        ::=  "SYNC";
Segment-Name             ::=  String | "PAUSE0" | "PAUSE1" |
                              "ACQUIRE" | "PAUSE" | "EXPECTED0" |
                              "EXPECTED1" | "PAUSE" |
                              "EXPECTED0" | "EXPECTED1" |
                              "DONTCARE" | "PAUSE" ;
Label                    ::=  String ;
Trigger                  ::=  "0" | "1" ;
Vectors                  ::=  Number ;
Iterations               ::=  Number | "INF" ;
VXI-Triggers-2           ::=  """ 2 * ( "0" | "1" )  """ |
                              "'" 2 * ( "0" | "1" ) "'" ;
Event-2                  ::=  String ;
Goto-2                   ::=  String ;
Number                   ::=  { Digit } ;
Digit                    ::=  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                              "8" | "9" ;
String                   ::=  "'" [ Letter { Letter | Digit } ] "'" |
                              """ [ Letter { Letter | Digit } ] """  ;
Letter                   ::=  "_" | "A" | "B" | "C" | "D" | "E" | "F" | "G"
                              | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O"
                              | "P" | "Q" | "R" | "S" | "T" | "U" | "V" |
                              "W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d" |
                              "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m"
                              | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" |
                              "v" | "w" | "x" | "y" | "z" ;
```

NOTE It is recommended to set up sequences in the graphical user interface. The resulting sequence expression can then be queried in the Command Line window. Afterwards, the sequence expression can easily be used in your program editor by copy and paste.

Range/ Value Coupling Event-2 names are defined by the events-expression described in the command. The expression is stored and checked against the number of ports and the maximum number of segments.

www.valuetronics.com

The expression is evaluated under two conditions:

- An SGEN:GLOB:SEQ:FORC command is executed.
- An SGEN:GLOB:INIT:CONT ON command is executed and the sequence expression is changed, or another parameter is changed that requires a new sequence / data download.

Therefore, syntax errors within the expression or range violations are detected only after these conditions.

Adding or removing ports changes the number of "Segment" entries needed. Therefore the sequence expression is automatically adjusted by inserting and deleting "Segment" entries. Inserted "Segment" entries depend on the port type:

- For INPUT_PORTs, an entry with a PAUSE0 segment is inserted.
- For OUTPUT_PORTs, an entry with a PAUSE segment is inserted.

The number of vectors depends on the system Segment Resolution as specified by the SGEN:GLOB:MUX command. The number of vectors must be a multiple of the Segment Resolution value. The minimum value depends on the number of looping levels started on that block: It must be (1 + "Number of Looping Levels Started") * "Segment Resolution".

The sequence will be downloaded with the next SGEN:GLOB:INIT:CONT ON command when one of the following happens:

- The channel add configuration changed (...:OUTP:CAC).
- A data port is added or deleted (SGEN:PDAT:APP, SGEN:PDAT(*):REM).
- A terminal within a data port is added or deleted (SGEN:CONN:PDAT(*):TERM, SGEN:CONN:PDAT(*):REM).
- The formatter mode is changed (...:DIG:STIM:SIGN:FORM).
- A frequency multiply factor is changed (...:MUX).
- The Segment Resolution is changed (SGEN:GLOB:MUX).
- A segment is changed (via the EDIT subsystem or via segment import MMEM:SEGM:LOAD).

Limits **Vectors** Just limited by the memory of the modules.

Iterations Up to 2^{20} or infinite on the highest available looping level (2 or 5 depending on the clock module used (see *":<Handle>:SGENeral:GLOBal:SEQuence:LLEVel?" on page 161*).

www.valuetronics.com

Segment-Name Valid keywords, depending on the port type and measurement mode:

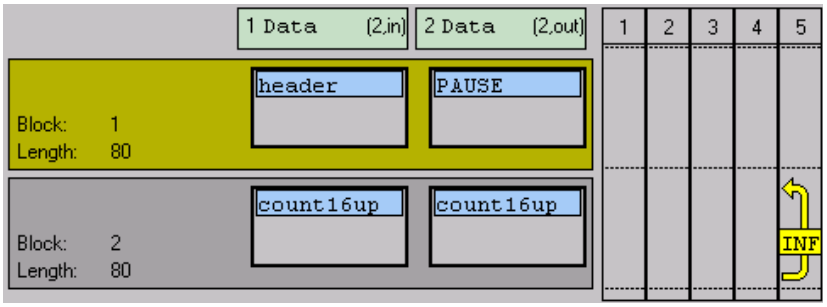
- Output Port: PAUSE0 and PAUSE1
- Input Port, Capture: ACQUIRE and PAUSE
- Input Port, Bit Error Rate: EXPECTED0, EXPECTED1, PAUSE
- Input Port, Compare and Acquire around Error: EXPECTED0, EXPECTED1, DONTCARE and PAUSE.

Description With this command, the data sequence is loaded into the system. The sequence expression corresponds to the Sequence Editor of the graphical user interface.

Example

```
:_test:SGEN:GLOB:SEQ (1.0,"",(SEQ, (BLOCK,0,80, "header",0,0,
PAUSE, 0, 0),"",
(LOOP5,0,INF,(BLOCK,0,80,"count16up",0,0,"count16up",0,0))))
```

results in the following sequence, as shown in the graphical user interface:



www.valuetronics.com

:< Handle> :SGENeral:GLOBal:SEQuence[:VALue]?

Syntax :<Handle>:SGENeral:GLOBal:SEQuence[:VALue]?

Return Value This query returns the current data sequence of the system.

Example `:_test:SGEN:GLOB:SEQ?`

might return:

```
(1.0,"",(SEQ,(BLOCK,0,80,"header",0,0,PAUSE,0,0),"",
(LOOP5,0,INF,(BLOCK,0,80,"count16up",0,0,"count16up",0,0))))
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal:SEQuence: EVENTs

Syntax :<Handle>:SGENeral:GLOBal:SEQuence:EVENTs <event-expression>

Parameters **<event-expression>** hierarchical expression, syntax described below

The event expression is a simple programming language, defined by the following EBNF:

Events-Expression	::=	"(2.0," 10 * Event-Expression ")" | "(3.0," 10 * Event-Expression ")" ;
Event-Expression	::=	"(" Name "," Enabled "," Or-Expressions ")" ;
Or-Expressions	::=	"(" Or-Expression { "," Or-Expressions } ")" ;
Or-Expression	::=	"(" CMD "," POD "," VXI-Triggers "," Errors ")" ;
Name	::=	String ;
Enabled	::=	"0" | "1" ;
CMD	::=	""" 1 * Bit-Mask """ | "'" 1 * Bit-Mask "'" ;
POD	::=	""" 8 * Bit-Mask """ | "'" 8 * Bit-Mask "'" ;
VXI-Triggers	::=	""" 2 * Bit-Mask """ | "'" 2 * Bit-Mask "'" ;
Errors	::=	"IGNORE" | "ERROR" | "NOERROR" | "ERROR" Number | "NOERROR" Number ;
Bit-Mask	::=	"0" | "1" | "x" | "X" ;
Number	::=	{ Digit } ;
Digit	::=	"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
String	::=	"'" Letter { Letter | Digit } "'" | """ Letter { Letter | Digit } """;
Letter	::=	"_" | "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" ;

www.valuetronics.com

Range/ Value Coupling

The events defined here are used by the sequence expression.

The expression is just stored and not checked. The expression is evaluated under two conditions:

- An SGEN:GLOB:SEQ:FORC command is executed.
- An SGEN:GLOB:INIT:CONT ON command is executed and the sequence expression is changed, or another parameter is changed that requires a new sequence / data download.

Therefore, syntax errors within the expression or range violations are detected only after these conditions.

The values possible for the Error events depend on the receive mode used. In Capture Mode the error events are simply ignored.

Deleting a port results in an error event (invalid port). Therefore, the event-expression is automatically adjusted to remove this error event.

Reset Value

As reset value all 10 events are disabled:

```
(3.0,(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))))
```

Limits

The specified events must be unique. The events to be used must be unambiguous.

Description:

With this command the events are defined for the system. The event expression corresponds to the Event Editor of the graphical user interface.

Example

```
sgen:glob:seq:even (2.0,(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))),
(,0,(('x','xxxxxxxx','xx',IGNORE))))
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal:SEQuence:EVENTs?

Syntax :<Handle>:SGENeral:GLOBal:SEQuence:EVENTs?

Range/ Value Coupling: This query is only available for E4805B and E4808A clock modules.

The events defined here are used by the sequence expression.

Reset Value

```
(2.0,(,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))))
```

Description This query returns the events defined by the system. The event-expression corresponds to the Event Editor of the graphical user interface.

Example

```
SGEN:GLOB:SEQ:EVEN?
```

might return:

```
(2.0,(,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))),
     (,0,(('x','xxxxxxxx','xx',IGNORE))))
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal:SEQuence: FORCe

Syntax :<Handle>:SGENeral:GLOBal:SEQuence:FORCe

Description Downloads the sequence (including events and data) into the hardware. The command corresponds to the *Prepare* button in the graphical user interface.

This command is useful when you want a fixed execution time for the SGEN:GLOB:INIT:CONT ON command. Otherwise, the SGEN:GLOB: INIT:CONT ON command downloads the sequence as needed and may therefore have different execution times depending on the history of commands that have been sent to the system.

Example

```
:_test:SGEN:GLOB:SEQ:FORC
```

:< Handle> :SGENeral:GLOBal:SEQuence: LLEVel?

Syntax :<Handle>:SGENeral:GLOBal:SEQuence:LLEVel?

Return Value Returns the actual number of available looping levels.

The graphical user interface uses this command to offer 5 looping levels in the Sequence Editor.

Example

```
:_test:SGEN:GLOB:SEQ:LLEV?
```

might return:

```
5
```


www.valuetronics.com

:<Handle>:SGENeral:GLOBal:SEQuence:PCONtrol

Syntax :<Handle>:SGENeral:GLOBal:SEQuence:PCONtrol <NRf>

Parameters Value in NRf format.

Range/ Value Coupling May influence the data sequence at runtime.

Reset Value 0

Limits Valid values are 0 and 1.

Description Sets the value of the program control event. With this event, which is part of the events defined by the events-expression, the data sequence can be influenced at runtime. This may be used for “Stop and Go” sequence model, etc.

Example

```
SGEN:GLOB:SEQ:PCON 1
```

:<Handle>:SGENeral:GLOBal:SEQuence:PCONtrol?

Syntax :<Handle>:SGENeral:GLOBal:SEQuence:PCONtrol?

Reset Value 0

Description Returns the state of the program control event line.

Example

```
SGEN:GLOB:SEQ:PCON?
```

might return:

```
1
```


www.valuetronics.com

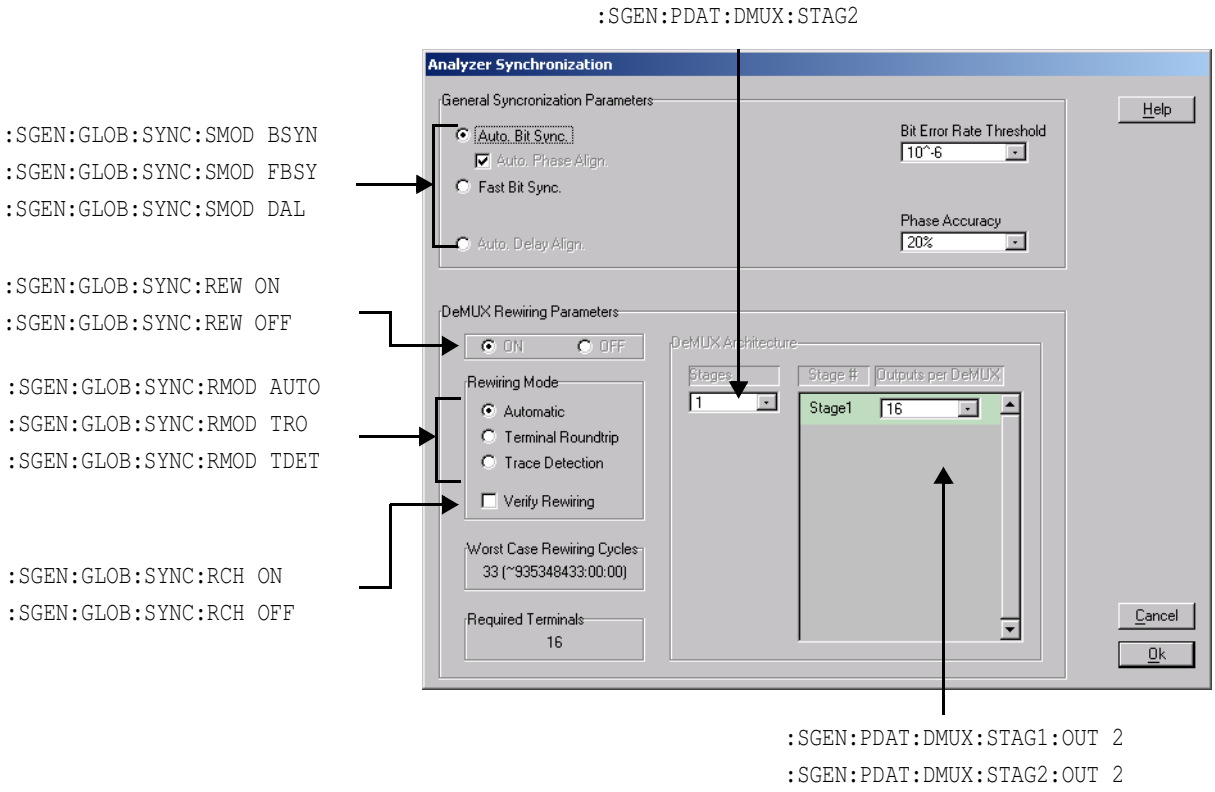
:SGENeral:GLOBal: SYNChronization Subsystem

Syntax :<Handle>:SGENeral:GLOBal:SYNChronization

This command system includes:

- Commands and queries for setting up the automatic alignment of incoming and expected data.
- DeMUX rewiring commands.

  The following figure shows the DeMUX rewiring commands in the Agilent 81250 user interface.



For the description of the DeMUX rewiring parameter commands, see *"DeMUX Rewiring Parameter Commands" on page 212*.

www.valuetronics.com

:<Handle>:SGENeral:GLOBal:SYNChronization:USED?

Syntax :<Handle>:SGENeral:GLOBal:SYNChronization:USED?

Return Value The query can only succeed (return TRUE) when there is at least one analyzer capable of synchronization installed.

Returns TRUE if the synchronization flag is found somewhere in the sequence and at least one analyzer capable of synchronization is installed.

Returns FALSE otherwise.

Example
```
:_test:SGEN:GLOB:SYNC:USED?
```
might return:
```
TRUE
```

:<Handle>:SGENeral:GLOBal:SYNChronization:BERThreshold

Syntax :<Handle>:SGENeral:GLOBal:SYNChronization:BERThreshold <Thresh>

Parameters **<Thresh>** Threshold for the bit error ratio to be accepted during synchronization. The range is from 1e-4 to 1e-9. The reset value is 1e-6.

Note that the time needed for synchronization increases with decreasing bit error ratio threshold. To assure that the bit error ratio is lower than the threshold, N * 1/<Thresh> bits must be measured (N>1).

<Thresh> is used during the synchronization process to determine whether the synchronization was successful. If after an internal synchronization attempt the bit error ratio is larger than this value, a new synchronization will be started. The synchronization block is only left when the bit error ratio is below the <Thresh> value.

This value is also used to calculate the optimum sampling point during sampling point optimization (phase adjust).

Note that this value will only be used if SGEN:GLOB:SYNC:USED? returns TRUE.

Example
```
SGEN:GLOB:SYNC:BERT 1e-9
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal: SYNChronization:BERThreshold?

Syntax :<Handle>:SGENeral:GLOBal:SYNChronization:BERThreshold?

Return Value Returns the programmed bit error ratio threshold. Note that this value will only be used if SGEN:GLOB:SYNC:USED? returns TRUE.

Example
```
SGEN:GLOB:SYNC:BERT?
```
might return:
```
1e-9
```

:< Handle> :SGENeral:GLOBal: SYNChronization:SMODe

Syntax :<Handle>:SGENeral:GLOBal:SYNChronization:SMODe <BSYNchronization | DALignment | FBSYnchronisation>

Parameters Specifies how the synchronization of the analyzers is achieved. Note that this value will only be used if SGEN:GLOB:SYNC:USED? returns TRUE.

BSYNchronization Neither the exact delay nor a delay range is known within which the incoming data will start. The data can come at any time after the system is started. The actual delay will be determined by automatic bit synchronization.

This is the reset value.

DALignment The exact sample delay is not known in advance, but a certain delay range the correct sample point will be inside. The actual delay will be determined by automatic delay adjustment.

FBSYnchronisation A method for aligning expected PRBS or PRWS data with incoming data very quickly. This method does not change the analyzer sampling delay.

Example
```
SGEN:GLOB:SYNC:SMOD BSYN
```


www.valuetronics.com

:<Handle>:SGENeral:GLOBal:SYNChronization:SMODe?

Syntax :<Handle>:SGENeral:GLOBal:SYNChronization:SMODe?

Return Value Returns the currently set synchronization mode.

Note that this value will only be used if SGEN:GLOB:SYNC:USED? returns TRUE.

Example
```
SGEN:GLOB:SYNC:SMOD?
```

might return:

```
BSYN
```

:<Handle>:SGENeral:GLOBal:SYNChronization:APAlignment

Syntax :<Handle>:SGENeral:GLOBal:SYNChronization:APAlignment <FALSe | TRUE>

Parameters Specifies whether a phase optimization is done after an automatic bit synchronization.

TRUE The phase is automatically optimized as specified by the SGEN:GLOB:SYNC:BERT and SGEN:GLOB:SYNC:PACC commands.

This is the reset value.

FALSe No automatic phase alignment.

Note that this value will only be used if SGEN:GLOB:SYNC:USED? returns TRUE and SGEN:GLOB:SYNC:SMOD? returns BSYN.

Example
```
SGEN:GLOB:SYNC:APA TRUE
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal: SYNChronization:APAlignment?

Syntax :<Handle>:SGENeral:GLOBal:SYNChronization:APAlignment?

Return Value Returns TRUE or FALSe depending on whether phase optimization will be performed after an automatic bit synchronization.

Note that this value will only be used if SGEN:GLOB:SYNC:USED? returns TRUE and SGEN:GLOB:SYNC:SMOD? returns BSYN.

Example `SGEN:GLOB:SYNC:APA?`

might return:

`TRUE`

:< Handle> :SGENeral:GLOBal: SYNChronization:PACCuracy

Syntax :<Handle>:SGENeral:GLOBal:SYNChronization:PACCuracy <accuracy>

Parameters **<accuracy>** Specifies in percent of the data period how accurate the phase optimization is done after a synchronization. Note that the time needed to achieve synchronization increases with the accuracy.

The range is from 0.01 to 0.2. The reset value is 0.2 (20 %).

Note that this value will only be used if SGEN:GLOB:SYNC:USED? returns TRUE.

Example `SGEN:GLOB:SYNC:PACC 0.01`

:< Handle> :SGENeral:GLOBal: SYNChronization:PACCuracy?

Syntax :<Handle>:SGENeral:GLOBal:SYNChronization:PACCuracy?

Return Value Returns the currently programmed phase accuracy to be achieved by automatic phase optimization after a synchronization.

This value will only be used if `SGEN:GLOB:SYNC:USED?` returns TRUE.

Example `SGEN:GLOB:SYNC:PACC?`

might return:

`2E-1`

www.valuetronics.com

:SGENeral:GLOBal:TRIGger Subsystem

:<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]

This command subsystem is used to set and measure the clock and clock reference.

:< Handle> :SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer][:SOURce]

Syntax :<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer][:SOURce] <INT10 | VXI10 | EXT1 | EXT2 | EXT5 | EXT10 | EXTernal>

Description This command sets the clock/reference or the external clock mode. The arguments available are:

INT10 reference	internal clock source, 10 MHz internal clock reference
VXI10 reference	internal clock source, 10 MHz VXI backplane reference
EXT1 reference	internal clock source, 1 MHz external clock reference
EXT2 reference	internal clock source, 2 MHz external clock reference
EXT5 reference	internal clock source, 5 MHz external clock reference
EXT10 reference	internal clock source, 10 MHz external clock reference
EXTernal	external clock source

For more information on clock sources, please refer to the *Agilent 81250 Parallel Bit Error Ratio Tester System User Guide*.

NOTE If the external clock mode is set and the external start mode is selected, the external clock must be in a stable state for 100 ms before the external start can take place.

Example

```
:_test:SGEN:GLOB:TRIG EXT5
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal:TRIGger?

Syntax :<Handle>:SGENeral:GLOBal:TRIGger?

Return Value This command returns the actual status of the clock/reference input of the central clock module. For possible values, see *":<Handle>: SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer][:SOURce]" on page 168*.

Example
```
:_test:SGEN:GLOB:TRIG?
```
might return:
```
EXT5
```

:< Handle> :SGENeral:GLOBal:TRIGger[: SEQuence][:LAYer]:CLOCk[:VALue]?

Syntax :<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLOCk[:VALue]?

Return Value Measures the supplied external clock at the clock/reference input of a clock module and returns the value without changing the mode of the machine.

The external clock can only be measured when the Agilent 81250 has been stopped.

Example
```
:_test:SGEN:GLOB:TRIG:SEQ:CLOC?
```

:< Handle> :SGENeral:GLOBal:TRIGger[: SEQuence][:LAYer]:CLOCk:M ULTiplier

Syntax :<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLOCk:MULTiplier <Multiplier>

Parameter **<Multiplier>** Sets the clock multiplication factor in a range of 1 - 32.

This command works only when the Agilent 81250 has been stopped.

Example
```
:_test:SGEN:GLOB:TRIG:CLOC:MULT 10
```


www.valuetronics.com

:<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLOCk:MULTiplier?

Syntax :<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLOCk:MULTiplier?

Return Value Returns the current clock multiplication factor.

This command works only when the Agilent 81250 has been stopped.

Example
```
:_test:SGEN:GLOB:TRIG:CLOC:MULT?
```
might return:
```
10
```

:<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLOCk:DIVider

Syntax :<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLOCk:DIVider <DIVider>

Parameters **<DIVider>** The clock divider.

Range/ Value Coupling Period, frequency, measured value.

Reset Value 1

Limits Works only on a stopped machine, range from 1 - 32.

Example
```
_TEST:SGEN:GLOB:TRIG:CLOC:DIV 2
```

:<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLOCk:DIVider?

Syntax :<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLOCk:DIVider?

Queries the current clock divider.

Example
```
:_test:sgen:glob:trig:cloc:div?
```
might return:
```
1
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:RCLock:DETect

Syntax :<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:RCLock:DETect [ONCE]

This command measures the external clock reference and sets the corresponding mode automatically. The automatically set modes are:

EXT1 reference	internal clock source with 1 MHz external clock
EXT2 reference	internal clock source with 2 MHz external clock
EXT5 reference	internal clock source with 5 MHz external clock
EXT10 reference	internal clock source with 10 MHz external clock

Example
```
:_test:SGEN:GLOB:TRIG:RCL:DET
```

:< Handle> :SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLock:MEASurement

Syntax :<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLock:MEASurement

This command measures the external clock and sets the corresponding mode EXTernal automatically.

Example
```
:_test:SGEN:GLOB:TRIG:CLOC:MEAS
```

NOTE The commands :RCL:DET and :CLOC:MEAS cannot be executed when the system is in the running state.

:< Handle> :SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:TVOLtage

Syntax :<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:TVOLtage <TermVoltage>

Parameters **<TermVoltage>** Specifies the termination voltage of the clock/ref input. The available termination voltage range is –2.10 V to +3.30 V. Default is 0 V.

Example
```
:_test:SGEN:GLOB:TRIG:TVOL 1
```


www.valuetronics.com

:<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:TVOLtage?

Syntax :<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:TVOLtage?

Return Value Returns the actual setting of the termination voltage of the clock/ref input.

Example

```
:_test:SGEN:GLOB:TRIG:TVOL?
```

might return:

```
1.000000E+000
```


www.valuetronics.com

:SGENeral:GLOBal:ARM Subsystem

:<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]

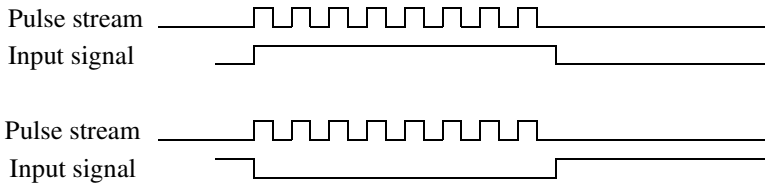
This command subsystem is used to control the system by an external signal.

The EXTernal :INPut connector at the front panel of the clock module is used to start the system by a so-called GATE or by a TRIGGER signal.

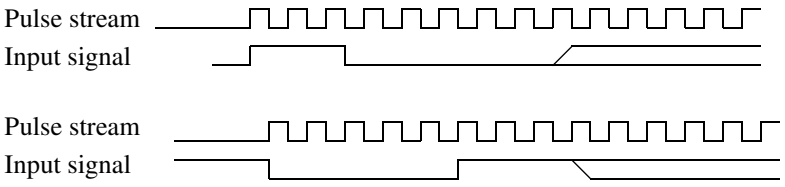
Following modes are available:

...:SOUR:IMM A pulse stream (sequence) is generated immediately as soon as the system is started by the start command.

...:SOUR:GATEd A pulse stream (sequence) is generated as soon as the level at the external input connector exceeds the input threshold. A sequence can be generated by a positive or a by a negative input signal depending on the parameter set.

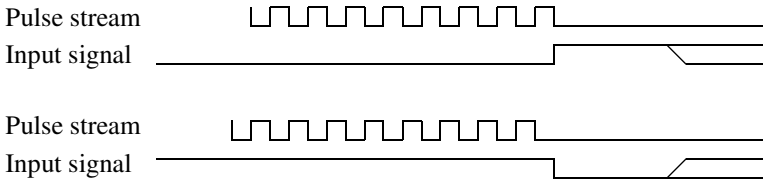


...:SOUR:STARtsignal A pulse stream (sequence) is generated as soon as a start signal exceeds the input threshold. A sequence can be started by a positive or a negative input signal depending on the parameter set.



www.valuetronics.com

...:SOUR:STOPsignal The pulse stream (sequence) is stopped as soon as a stop signal exceeds the input threshold. The sequence can be stooped by a positive or negative input signal depending on the parameter set.All these modes are active if the system is started manually or started by a remote command.



:< Handle> :SGENeral:GLOBal:ARM [:SEQuence][:LAYer][:SOURce]

Syntax :<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer][:SOURce] IMMediate | GATed | STARtsignal | STOPsignal

Description This command controls the start mode of the Agilent 81250 System. If IMMediate is selected, then the sequence can be started and stopped by the :SGEN:GLOB:INIT:CONT ON | OFF command, or it is possible to press the start or stop button in the graphical user interface.

NOTE If the external clock mode is set and the external start mode is selected, the external clock must be in a stable state for 100 ms before the external start can take place.

Example

```
:_test:SGEN:GLOB:ARM IMM
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal:ARM ?

Syntax :<Handle>:SGENeral:GLOBal:ARM?

Return Value This command returns the actual status of the External Input connector at the front of the central clock module. It shows how this input controls the system.

IMM Immediate

GAT Gated

STAR Started by a start signal

STOP Stopped by a stop signal

:< Handle> :SGENeral:GLOBal:ARM [:SEQuence][:LAYer]:SENSe

Syntax :<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:SENSe PLEVel | NLEVel

This command specifies the input level condition which is used in the GATed | STARtsignal | STOPsignal modes.

Example

```
:_test:SGEN:GLOB:ARM:SENS PLEV
```

:< Handle> :SGENeral:GLOBal:ARM [:SEQuence][:LAYer]:SENSe?

Syntax :<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:SENSe?

Returns the actual setting of the input level condition for the gate or start/stop signal (PLEVel or NLEVel).

Example

```
:_test:SGEN:GLOB:ARM:SENS?
```

might return:

```
PLEV
```


www.valuetronics.com

:<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:THReshold

Syntax :<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:THReshold <NRf>

This command specifies the threshold of the external input signal which is used in the `GATed | STARtsignal | STOPsignal` modes. The range is from -1.40 V to +3.70 V.

Example

```
:_test:SGEN:GLOB:ARM:THR 1
```

:<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:THReshold?

Syntax :<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:THReshold?

Returns the threshold for the external input signal used in the gate or start/stop mode.

Example

```
:_test:SGEN:GLOB:ARM:THR?
```

might return:

```
1.000000E+000
```

:<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:TVOLtage

Syntax :<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:TVOLtage <NRf>

This command specifies the termination voltage of the external input signal which is used in the GATed | STARtsignal | STOPsignal modes. The range is from -2.10 V to +3.30 V.

Example

```
:_test:SGEN:GLOB:ARM:TVOL -2
```


www.valuetronics.com

:< Handle> :SGENeral:GLOBal:ARM [:SEQuence][:LAYer]:TVOLtage?

Syntax :<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:TVOLtage?

Returns the actual termination voltage for the external input signal used in the gate or start/stop mode.

Example

```
:_test:SGEN:GLOB:ARM:TVOL?
```

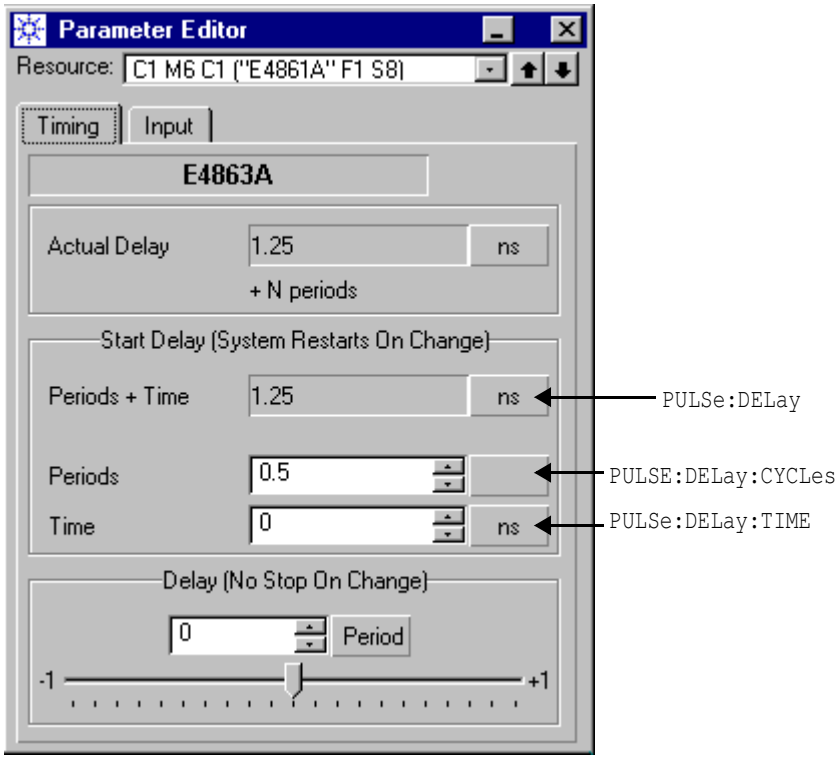
might return:

```
-2.000000E+000
```


www.valuetronics.com

Timing Parameter Commands

The commands for specifying the timing parameters are available on port, terminal and connector level. The parameters can be specified separately for pulse and data ports.



www.valuetronics.com

:PULSe:DELay

Syntax

:<Handle>:SGENeral:**PDATa(*)**:[:SOURce]:PULSe:DELay <Delay>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:[:SOURce]:PULSe:DELay <Delay>

:<Handle>:SGENeral:**PPULse(*)**[:SOURce]:PULSe:DELay <Delay>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:PULSe:DELay <Delay>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**:[:SOURce]:PULSe:DELay <Delay>

Parameters

<Delay> Delay value in seconds (<NRf>).

Description

Sets the time from the start of the period to the first edge of the pulse. Reference is the trigger output.

Valid range for E4832A: 0 ... max (3 µs, PERiod).

Valid range for E4861A, E4861B, E4866A: 0 ... 300 ns.

If this command is used, the value is interpreted as a timing value only. This sets `PULSe:DELay:CYCLes` to zero.

Otherwise, if the delay is specified by `PULSe:DELay:CYCLes` and `PULSe:DELay:TIME`, the corresponding delay is calculated. The number of cycles is converted into seconds and added to the specified `PULSe:DELay:TIME` value.

Example

```
:_test:SGEN:PDAT1:PULS:DEL 50e-9
:_test:SGEN1:PDAT1:TERM1:PULS:DEL 50e-9
:_test:SGEN:PDAT1:PULS:DEL 50e-9
:_test:SGEN1:PDAT1:TERM1:PULS:DEL 50e-9
:_test:CGROUP1:MODULE2:CONNECTOR4:PULS:DEL 50e-9
```


www.valuetronics.com

:PULSe:DELay?

Syntax

:<Handle>:SGENeral:**PDATa(*)**:[:SOURce]:PULSe:DELay?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:[:SOURce]:PULSe:DELay?

:<Handle>:SGENeral:**PPULs(*)**:[:SOURce]:PULSe:DELay?

:<Handle>:SGENeral:**PPULs (*):TERMinal(*)**:[:SOURce]:PULSe:DELay?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**:[:SOURce]:PULSe:DELay?

Return Value

Returns the current pulse delay value.

Example

```
:_test:SGEN:PDAT1:PULS:DEL?
:_test:SGEN1:PDAT1:TERM1:PULS:DEL?
:_test:SGEN:PPUL1:PULS:DEL?
:_test:SGEN1:PPUL1:TERM1:PULS:DEL?
:_test:CGROUP1:MODULE2:CONNECTOR4:PULS:DEL?
```

might return:

```
5.000000E-008
```

:PULSe:DELay:CYCLes

Syntax

:<Handle>:SGENeral:**PDATa(*)**:[:SOURce]:PULSe:DELay:CYCLes <Cycles>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:[:SOURce]:PULSe:DELay:CYCLes <Cycles>

:<Handle>:SGENeral:**PPULs(*)**:[:SOURce]:PULSe:DELay:CYCLes <Cycles>

:<Handle>:SGENeral:**PPULs (*):TERMinal(*)**:[:SOURce]:PULSe:DELay:CYCLes <Cycles>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**:[:SOURce]:PULSe:DELay:CYCLes <Cycles>

Parameters

<Cycles> One cycle corresponds to the actual period/frequency valid for the specified generator output connector.

If the delay is specified by `DELay:CYCLes` and `PULSe:DELay:TIME`, the corresponding delay will be calculated. The number of cycles is converted into seconds and added to the specified `PULSe:DELay:TIME` value.

If `PULSe:DELay` is used, the value is interpreted as a timing value only. This sets `PULSe:DELay:CYCLes` to zero.

www.valuetronics.com

Example

```
:_test:SGEN:PDAT1:PULS:DEL:CYCL 0.5
:_test:SGEN1:PDAT1:TERM1:PULS:DELAY:CYCL 0.5
:_test:SGEN:PPUL1:PULS:DEL:CYCL 0.5
:_test:SGEN1:PPUL1:TERM1:PULS:DELAY:CYCL 0.5
:_test:CGROUP1:MODULE2:CONNECTOR4:PULS:DELAY:CYCL 0.5
```

:PULSe:DELay:CYCLes?

Syntax

:<Handle>:SGENeral:**PDATa(*)**:[:SOURce]:PULSe:DELay:CYCLes?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:[:SOURce]:PULSe:DELay:CYCLes?

:<Handle>:SGENeral:**PPULs(*)**:[:SOURce]:PULSe:DELay:CYCLes?

:<Handle>:SGENeral:**PPULs (*):TERMinal(*)**:[:SOURce]:PULSe:DELay:CYCLes?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**:[:SOURce]:PULSe:DELay:CYCLes?

Return Value

Returns the current cycle value of the pulse delay for the specified generator output connector.

Example

```
:_test:SGEN:PDAT1:PULS:DEL:CYCL?
:_test:SGEN1:PDAT1:TERM1:PULS:DEL:CYCL?
:_test:SGEN:PPUL1:PULS:DEL:CYCL?
:_test:SGEN1:PPUL1:TERM1:PULS:DEL:CYCL?
:_test:CGROUP1:MODULE2:CONNECTOR4:PULSE:DELAY:CYCL?
```

might return:

```
5.000000E-001
```


www.valuetronics.com

:PULSe:DELay:TIME

Syntax

:<Handle>:SGENeral:**PDATa(*)**:[:SOURce]:PULSe:DELay:TIME <Time>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:[:SOURce]:PULSe:DELay:TIME <Time>

:<Handle>:SGENeral:**PPULs(*)**:[:SOURce]:PULSe:DELay:TIME <Time>

:<Handle>:SGENeral:**PPULs (*):TERMinal(*)**:[:SOURce]:PULSe:DELay:TIME <Time>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**:[:SOURce]:PULSe:DELay:TIME <Time>

Parameters

<Time> Delay in seconds.

If the delay is specified by `DELay:TIME` and `PULSe:DELay:TIME`, the corresponding delay is calculated. The number of cycles is converted into seconds and added to the specified `PULSe:DELay:TIME` value.

If `PULSe:DELay` is used, the value is interpreted as a timing value only. This sets `PULSe:DELay:TIME` to zero.

Example

```
:_test:SGEN:PDAT1:PULS:DEL:TIME 0.5
:_test:SGEN1:PDAT1:TERM1:PULS:DEL:TIME 0.5
:_test:SGEN:PPUL1:PULS:DELAY:TIME 0.5
:_test:CGROUP1:MODULE2:CONNECTOR4:PULS:DELAY:TIME 0.5
```

:PULSe:DELay:TIME?

Syntax

:<Handle>:SGENeral:**PDATa(*)**:[:SOURce]:PULSe:DELay:TIME?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:[:SOURce]:PULSe:DELay:TIME?

:<Handle>:SGENeral:**PPULs(*)**:[:SOURce]:PULSe:DELay:TIME?

:<Handle>:SGENeral:**PPULs (*):TERMinal(*)**:[:SOURce]:PULSe:DELay:TIME?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**:[:SOURce]:PULSe:DELay:TIME?

Return Value

Returns the current cycle value of the pulse delay for the specified generator output connector.

Example

```
:_test:SGEN:PDAT1:PULS:DELAY:TIME?
:_test:SGEN1:PDAT1:TERM1:PULS:DEL:TIME?
:_test:SGEN:PPUL1:PULS:DEL:TIME?
:_test:CGROUP1:MODULE2:CONNECTOR4:PULSE:DELAY:TIME?
```

might return:

```
5.000000E-001
```


www.valuetronics.com

:PULSe:CROSsing

Syntax

:<Handle>:SGENeral:PDATa(*):[:SOURce]:PULSe:CROSsing <CROSsing>

:<Handle>:SGENeral:PDATa(*):TERMinal(*)[:SOURce]:PULSe:CROSsing <CROSsing>

:<Handle>:SGENeral:PPULse(*):[:SOURce]:PULSe:CROSsing <CROSsing>

:<Handle>:SGENeral:PPULse(*):TERMinal(*)[:SOURce]:PULSe:CROSsing <CROSsing>

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*)[:SOURce]:PULSe:CROSsing <CROSsing>

Parameters

<CROSsing> Cross point of the signal in percentage of signal amplitude. The formula for the cross point is:

Cross Point = 100 * ((V_{Cross} – V_{High}) / (V_{High} – V_{Low}))

This parameter is valid only for NRZ format modes.

Cross point range is subject to change; depending on hardware capabilities.

Valid range for E4862B: 25 – 75 %

NOTE This parameter can also be used for the N4868A 10.8G Booster modules. The valid range is: 40 – 60 %

Example

```
:_test:SGEN:PDAT1:PULS:CROS 40.0
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:CROS 40.0
:_test:SGEN:PPUL1:PULS:CROS 40
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:CROS 40
:_test:MODULE2:CONNECTOR3:PULSE:CROSSING 40.0
```


www.valuetronics.com

:PULSe:CROSsing?

Syntax

:<Handle>:SGENeral:PDATa(*):[:SOURce]:PULSe:CROSsing?

:<Handle>:SGENeral:PDATa(*):TERMinal(*)[:SOURce]:PULSe:CROSsing?

:<Handle>:SGENeral:PPULse(*):[:SOURce]:PULSe:CROSsing?

:<Handle>:SGENeral:PPULse(*):TERMinal(*)[:SOURce]:PULSe:CROSsing?

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*)[:SOURce]:PULSe:CROSsing?

Return Value

Returns the current cross point. The unit for the cross point is in percentage of signal amplitude. The formula for the cross point is:

Cross Point = 100 * ((V_{Cross} – V_{High}) / (V_{High} – V_{Low}))

Example

```
:_test:SGEN:PDAT1:PULS:CROS?
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:CROS?
:_test:SGEN:PPUL1:PULS:CROS?
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:CROS?
:_test:MODULE2:CONNECTOR3:PULSE:CROSSING?
```

might return:

```
50.0
```


www.valuetronics.com

:PULSe:WIDTh

Syntax

:<Handle>:SGENeral:**PDATa(*)**[:SOURce]:PULSe:WIDTh <Width>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:PULSe:WIDTh <Width>

:<Handle>:SGENeral:**PPULse(*)**[:SOURce]:PULSe:WIDTh <Width>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:PULSe:WIDTh <Width>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:PULSe:WIDTh <Width>

Parameters

<Width> Pulse width in seconds (<NRf>).

Sets the width or duration of the pulse.

Valid range 0 ... PERiod
(on connector level: 0 ... PERiod / Connector:MUX).

Example

```
:_test:SGEN:PDAT1:PULS:WIDT 15e-9
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:WIDT 10e-9
:_test:SGEN:PPUL1:PULS:WIDT 15e-9
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:WIDT 10e-9
:_test:MOD2:CONN3:PULS:WIDT 20e-9
```

:PULSe:WIDTh?

Syntax

:<Handle>:SGENeral:**PDATa(*):**[:SOURce]:PULSe:WIDTh?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:PULSe:WIDTh?

:<Handle>:SGENeral:**PPULse(*):**[:SOURce]:PULSe:WIDTh?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:PULSe:WIDTh?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:PULSe:WIDTh?

Return Value

Returns the current pulse width value.

Example

```
:_test:SGEN:PDAT1:PULS:WIDT?
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:WIDT?
:_test:SGEN:PPUL1:PULS:WIDT?
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:WIDT?
:_test:MODULE2:CONNECTOR3:PULSE:WIDTH?
```

might return:

```
1.000000E-008
```


www.valuetronics.com

:PULSe:DCYCle

Syntax

:<Handle>:SGENeral:**PDATa(*)**[:SOURce]:PULSe:DCYCle <Duty Cycle>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:PULSe:DCYCle <Duty Cycle>

:<Handle>:SGENeral:**PPULse(*)**[:SOURce]:PULSe:DCYCle <Duty Cycle>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:PULSe:DCYCle <Duty Cycle>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:PULSe:DCYCle <Duty Cycle>

Parameters

<Duty Cycle> Duty Cycle value in % (<NRf>).

Sets the duty cycle of a repetitive pulse waveform.

Valid range 0 ... 100.

Example

```
:_test:SGEN:PDAT1:PULS:DCYC 25
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:DCYC 30
:_test:SGEN:PPUL1:PULS:DCYC 25
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:DCYC 30
:_test:mod2:conn3:pulse:DCYCLE 30
```

:PULSe:DCYCle?

Syntax

:<Handle>:SGENeral:**PDATa(*):**[:SOURce]:PULSe:DCYCle?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:PULSe:DCYCle?

:<Handle>:SGENeral:**PPULse(*):**[:SOURce]:PULSe:DCYCle?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:PULSe:DCYCle?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:PULSe:DCYCle?

Return Value

Returns the current duty cycle value.

Example

```
:_test:SGEN:PDAT1:PULS:DCYC?
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:DCYC?
:_test:SGEN:PPUL1:PULS:DCYC?
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:DCYC?
:_test:MOD2:CONNECTOR3:PULS:DCYCLE?
```

might return:

```
30
```


www.valuetronics.com

:PULSe:HOLD

Syntax

:<Handle>:SGENeral:**PDATa(*)**[:SOURce]:PULSe:HOLD <WIDTh | DCYCle>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:PULSe:HOLD <WIDTh | DCYCle>

:<Handle>:SGENeral:**PPULse(*)**[:SOURce]:PULSe:HOLD <WIDTh | DCYCle>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:PULSe:HOLD <WIDTh | DCYCle>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:PULSe:HOLD <WIDTh | DCYCle>

Parameters

<WIDTh | DCYCle> The value to hold constant.

Specifies which parameter should remain constant when the period changes.

Example

```
:_test:SGEN:PDAT1:PULS:HOLD DCYC
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:HOLD DCYC
:_test:SGEN:PPUL1:PULS:HOLD DCYC
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:HOLD WIDT
:_test:CGR:MOD2:CONN3:SOUR:PULS:HOLD WIDT
```

:PULSe:HOLD?

Syntax

:<Handle>:SGENeral:**PDATa(*):**[:SOURce]:PULSe:HOLD?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:PULSe:HOLD?

:<Handle>:SGENeral:**PPULse(*):**[:SOURce]:PULSe:HOLD?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:PULSe:HOLD?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:PULSe:HOLD?

Return Value

Returns the parameter which is held constant when the period changes.

Example

```
:_test:SGEN:PDAT1:PULS:HOLD?
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:HOLD?
:_test:SGEN:PPUL1:PULS:HOLD?
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:HOLD?
:_test:CGR:MOD2:CONN3:SOUR:PULS:HOLD?
```

might return:

```
DCYC
```


www.valuetronics.com

:PULSe:TRANsition[:LEADing]

Syntax

:<Handle>:SGENeral:**PDATa**(*)[:SOURce]:PULSe:TRANsition[:LEADing] <LeadingEdge>

:<Handle>:SGENeral:**PDATa**(*)**:TERMinal**(*)[:SOURce]:PULSe:TRANsition [:LEADing] <LeadingEdge>

:<Handle>:SGENeral:**PPULse**(*)[:SOURce]:PULSe:TRANsition[:LEADing] <LeadingEdge>

:<Handle>:SGENeral:**PPULse**(*)**:TERMinal**(*)[:SOURce]:PULSe:TRANsition [:LEADing] <LeadingEdge>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector**(*)[:SOURce]:PULSe:TRANsition
[:LEADing] <LeadingEdge>

Parameters

<LeadingEdge> Leading Edge value (<NRf>).

Sets the transition time for the LEADing edge. Only supported by E4842A and E4838A frontends. The valid ranges are

E4842A: 0.5ns ... 8.0ns
E4838A: 0.5ns ... 5.0ns

Consider the following restrictions:

- When the specified port/terminal/connector is operating in NRZ format-mode, the actual value range is restricted by the selected period/frequency:
  - *":<Handle>:SGENeral:GLOBal:PERiod" on page 141*
  - *":<Handle>:SGENeral:GLOBal:FREQuency" on page 142*
  - *":MUX" on page 196*
  - *":FORMat" on page 299*
- When the specified port/terminal/connector is operating in RZ or R1 format-mode, the actual value range is restricted by the selected width or the duty cycle:
  - *":PULSe:WIDTh" on page 185*
  - *":PULSe:DCYCle" on page 186*
  - *":MUX" on page 196*
  - *":FORMat" on page 299*

www.valuetronics.com

Example

```
:_test:SGEN:PDAT1:PULS:TRAN 2e-9
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:TRAN:LEAD
:_test:SGEN:PPUL1:PULS:TRAN 2e-9
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:TRAN:LEAD 5e-9
:_test:MOD2:CONN3:PULS:TRAN 5e-9
```

:PULSe:TRANsition[:LEADing]?

Syntax

:<HANDLE>:SGENeral:**PDATa(*):**[:SOURce]:PULSeTRANsition[:LEADing]?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:PULSe:TRANsition[:LEADing]?

:<Handle>:SGENeral:**PPULse(*):**[:SOURce]:PULSeTRANsition[:LEADing]?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:PULSe:TRANsition[:LEADing]?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:PULSe:TRANsition[:LEADing]?

Return Value

Returns the transition time for the leading edge.

Example

```
:_test:SGEN:PDAT1:PULS:TRAN?
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:LEAD?
:_test:SGEN:PDAT1:PULS:TRAN?
:_test:SGEN:PPUL:TERM1:SOUR:PULS:LEAD?
:_test:MOD2:CONN3:PULS:TRAN?
```

might return:

```
2.000000E-009
```


www.valuetronics.com

:PULSe:TRANsition:TRAiling

Syntax

:<Handle>:SGENeral:**PDATa**(*)[:SOURce]:PULSe:TRANsition:TRAiling <TrailingEdge>

:<Handle>:SGENeral:**PDATa**(*)**:TERMinal**(*)[:SOURce]:PULSe:TRANsition :TRAiling <TrailingEdge>

:<Handle>:SGENeral:**PPULse**(*)[:SOURce]:PULSe:TRANsition:TRAiling <TrailingEdge>

:<Handle>:SGENeral:**PPULse**(*)**:TERMinal**(*)[:SOURce]:PULSe:TRANsition :TRAiling <TrailingEdge>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector**(*)[:SOURce]:PULSe: TRANsition:TRAiling <TrailingEdge>

Parameters

<TrailingEdge> Trailing Edge value (<NRf>).

Sets the transition time for the trailing edge.

Only supported by E4842A and E4838A frontends. The valid ranges are

E4842A: 0.5ns ... 8.0ns
E4838A: 0.5ns ... 5.0ns

Consider the following restrictions:

- When the specified port/terminal/connector is operating in NRZ format-mode, the actual value range is restricted by the selected period/frequency:
  - *":<Handle>:SGENeral:GLOBal:PERiod" on page 141*
  - *":<Handle>:SGENeral:GLOBal:FREQuency" on page 142*
  - *":MUX" on page 196*
  - *":FORMat" on page 299*
- When the specified port/terminal/connector is operating in RZ or R1 format-mode, the actual value range is restricted by the selected width or the duty cycle:
  - *":PULSe:WIDTh" on page 185*
  - *":PULSe:DCYCle" on page 186*
  - *":MUX" on page 196*
  - *":FORMat" on page 299*

www.valuetronics.com

Example

```
:_test:SGEN:PDAT1:PULS:TRAN:TRA 2e-9
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:TRAN:TRA 2e-9
:_test:SGEN:PPUL1:PULS:TRAN:TRA 2e-9
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:TRAN:TRA 2e-9
:_test:CGR1:MOD2:CONN3:SOUR:PULS:TRAN:TRA 2e-9
```

:PULSe:TRANsition:TRAiling?

Syntax

:<Handle>:SGENeral:**PDATa(*):**[:SOURce]:PULSe:TRANsition:TRAiling?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:PULSe:TRANsition:TRAiling?

:<Handle>:SGENeral:**PPULse(*):**[:SOURce]:PULSe:TRANsition:TRAiling?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:PULSe:TRANsition:TRAiling?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:PULSe:TRANsition
:TRAiling?

Return Value

Returns the transition time for the TRAiling edge.

Example

```
:_test:SGEN:PDAT1:PULS:TRAN:TRA?
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:TRAN:TRA?
:_test:SGEN:PPUL1:PULS:TRAN:TRA?
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:TRAN:TRA?
:_test:CGR1:MOD2:CONN3:SOUR:PULS:TRAN:TRA?
```

might return:

```
2.000000E-009
```


www.valuetronics.com

:PULSe:TRANsition:CAConfiguration[:LEADing]

Syntax

:<Handle>:SGENeral:**PDATa(*)**[:SOURce]:PULSe:TRANsition:CAConfiguration [:LEADing] <ChannelAddLeadingEdge>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:PULSe:TRANsition : CAConfiguration [:LEADing] <ChannelAddLeadingEdge>

:<Handle>:SGENeral:**PPULse(*)**[:SOURce]:PULSe:TRANsition: CAConfiguration[:LEADing] <ChannelAddLeadingEdge>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:PULSe:TRANsition :CAConfiguration[:LEADing] <ChannelAddLeadingEdge>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:PULSe: TRANsition:CAConfiguration[:LEADing] <ChannelAddLeadingEdge>

Parameters

<ChannelAddLeadingEdge> Additional Leading Edge value (<NRf>).

Sets the additional leading edge if the Channel Add mode is activated. This command is only supported for E4838A frontends in A2 channel-add mode (see *":OUTPut:CAConfiguration[:MODE]" on page 267*).

The overall range is 0.5 ns to 5 ns. Consider the following restrictions:

- When the specified port/terminal/connector is operating in NRZ format-mode, the actual value range is restricted by the selected period/frequency:
  - *":<Handle>:SGENeral:GLOBal:PERiod" on page 141*
  - *":<Handle>:SGENeral:GLOBal:FREQuency" on page 142*
  - *":MUX" on page 196*
  - *":FORMat" on page 299*
- When the specified port/terminal/connector is operating in RZ or R1 format-mode, the actual value range is restricted by the selected width or the duty cycle:
  - *":PULSe:WIDTh" on page 185*
  - *":PULSe:DCYCle" on page 186*
  - *":MUX" on page 196*
  - *":FORMat" on page 299*

NOTE *":PULSe:TRANsition:CAConfiguration:TRAiling" on page 194* holds actually the same value. Both parameters are cross-updated.

www.valuetronics.com

Example

```
:_test:SGEN:PDAT1:PULS:TRAN:CAC 2e-9
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:TRAN:CAC:LEAD 3e-9
:_test:SGEN:PPUL1:PULS:TRAN:CAC 2e-9
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:TRAN:CAC:LEAD 3e-9
:_test:MODULE2:CONNECTOR4:PULSE:TRANSISITON:CACONFIGURATION 3e-9
```

:PULSe:TRANsition:CAConfiguration[:LEADing]?

Syntax

:<Handle>:SGENeral:**PDATa(*):**[:SOURce]:PULSe:TRANsition :CAConfiguration[:LEADing]?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:PULSe:TRANsition :CAConfiguration[:LEADing]?

:<Handle>:SGENeral:**PPULse(*):**[:SOURce]:PULSe:TRANsition :CAConfiguration[:LEADing]?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:PULSe:TRANsition :CAConfiguration[:LEADing]?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:PULSe :TRANsition:CAConfiguration[:LEADing]?

Return Value

Returns the additional transition-time of an E4838A frontend in A2 channel-add mode.

Example

```
:_test:SGEN:PDAT1:PULS:TRAN:CAC?
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:TRAN:CAC:LEAD?
:_test:SGEN:PPUL1:PULS:TRAN:CAC?
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:TRAN:CAC:LEAD?
:_test:MODULE2:CONNECTOR4:PULSE:TRANSISITON:CACONFIGURATION?
```

might return:

```
2.000000E-009
```


www.valuetronics.com

:PULSe:TRANsition:CAConfiguration:TRAiling

Syntax

:<Handle>:SGENeral:**PDATa(*)**[:SOURce]:PULSe:TRANsition:CAConfiguration:TRAiling <ChannelAddTrailingEdge>

:<Handle>:SGENeral:**PPULse(*)**[:SOURce]:PULSe:TRANsition:CAConfiguration:TRAiling <ChannelAddTrailingEdge>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:PULSe>:TRANsition:CAConfiguration:TRAiling <ChannelAddTrailingEdge>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:PULSe:TRANsition:CAConfiguration:TRAiling <ChannelAddTrailingEdge>

Parameters

<ChannelAddTrailingEdge> Additional Trailing Edge value (<NRf>).

Description

Sets the additional trailing edge, if the *Channel Add* mode is activated. This command is only supported for E4838A frontends in A2 channel-add mode (see *":OUTPut:CAConfiguration[:MODE]" on page 267*).

The overall range is 0.5 ns to 5 ns. Consider the following restrictions:

- When the specified port/terminal/connector is operating in NRZ format-mode, the actual value range is restricted by the selected period/frequency:
  - *":<Handle>:SGENeral:GLOBal:PERiod" on page 141*
  - *":<Handle>:SGENeral:GLOBal:FREQuency" on page 142*
  - *":MUX" on page 196*
  - *":FORMat" on page 299*
- When the specified port/terminal/connector is operating in RZ or R1 format-mode, the actual value range is restricted by the selected width or the duty cycle:
  - *":PULSe:WIDTh" on page 185*
  - *":PULSe:DCYCle" on page 186*
  - *":MUX" on page 196*
  - *":FORMat" on page 299*

NOTE *":PULSe:TRANsition:CAConfiguration[:LEADing]" on page 192* holds actually the same value. Both parameters are cross-updated.

www.valuetronics.com

Example

```
:_test:SGEN:PDAT1:PULS:TRAN:CAC:TRA 2e-9
:_test:SGEN:PPUL1:PULS:TRAN:CAC:TRA 2e-9
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:TRAN:CAC:TRA 3e-9
:_test:MOD2:CONN4:PULS:TRAN:CAC:TRA 4e-9
```

:PULSe:TRANsition:CAConfiguration:TRAiling?

Syntax

:<Handle>:SGENeral:**PDATa(*):**[:SOURce]:PULSe:TRANsition:CAConfiguration:TRAiling?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:PULSe:TRANsition:CAConfiguration:TRAiling?

:<Handle>:SGENeral:**PPULse(*):**[:SOURce]:PULSe:TRANsition:CAConfiguration:TRAiling?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:PULSe:TRANsition:CAConfiguration:TRAiling?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:PULSe:TRANsition:CAConfiguration:TRAiling?

Return Value

Returns the additional transition-time of an E4838A frontend in A2 channel-add mode.

Example

```
:_test:SGEN:PDAT1:PULS:TRAN:CAC:TRA?
:_test:SGEN:PDAT1:TERM1:SOUR:PULS:TRAN:CAC:TRA?
:_test:SGEN:PPUL1:PULS:TRAN:CAC:TRA?
:_test:SGEN:PPUL1:TERM1:SOUR:PULS:TRAN:CAC:TRA?
:_test:MOD2:CONN4:PULS:TRAN:CAC:TRA?
```

might return:

```
2.000000E-009
```


www.valuetronics.com

:MUX

Syntax

:<Handle>:SGENeral:**PDATa(*)**:MUX <MUX Factor>

:<Handle>:SGENeral:**PPULse(*)**:MUX <MUX Factor>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**:MUX <MuxFactor>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**:MUX <NRf>

Parameters

<NRf1> MUX factor = Multiply Factor for the individual port or connector.

In addition to the "system" MUX factor (segment resolution) it is possible to set the frequency multiply factor for individual ports, terminals or connectors to generate multiple or fraction of 2 data streams, like f/2 or 2f etc. The default value is 1. Valid values are:

0.0625 (1/16), 0.125 (1/8), 0.25 (1/4), 0.5 (1/2), 1, 2, 4, 8, 16

If an E4861A module is present, this range is expanded by 1/64 and 64.

The range of this parameter depends on the value of the system MUX factor (see *":<Handle>:SGENeral:GLOBal:MUX" on page 143*).

Example

```
:_test:SGEN:PDAT1:MUX 4
:_test:SGEN:PPUL1:MUX 4
:_test:SGEN:PPUL1:TERM1:MUX 0.5
:_test:CGR1:MOD1:CONN1:MUX 2
```

:MUX?

Syntax

:<Handle>:SGENeral:**PDATa(*)**:MUX?

:<Handle>:SGENeral:**PPULse(*)**:MUX?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**:MUX?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**:MUX?

Return Value

Returns the current frequency multiply factor (MUX factor) for the specified port, terminal or connector.

Example

```
:_test:SGEN:PDAT1:MUX?
:_test:SGEN:PPUL1:MUX?
:_test:SGEN:PPUL1:TERM1:MUX?
:_test:CGR1:MOD1:CONN1:MUX?
```

might return:

```
6.25E-2
```


www.valuetronics.com

Parameter Commands for MUX and DEMUX Modules

Available Parameter Commands

All new commands to set the properties of the MUX and DEMUX modules are found under:

```
:<Handle>[:CGRoup*]:MODule*:CONNector*
```

The following commands are available:

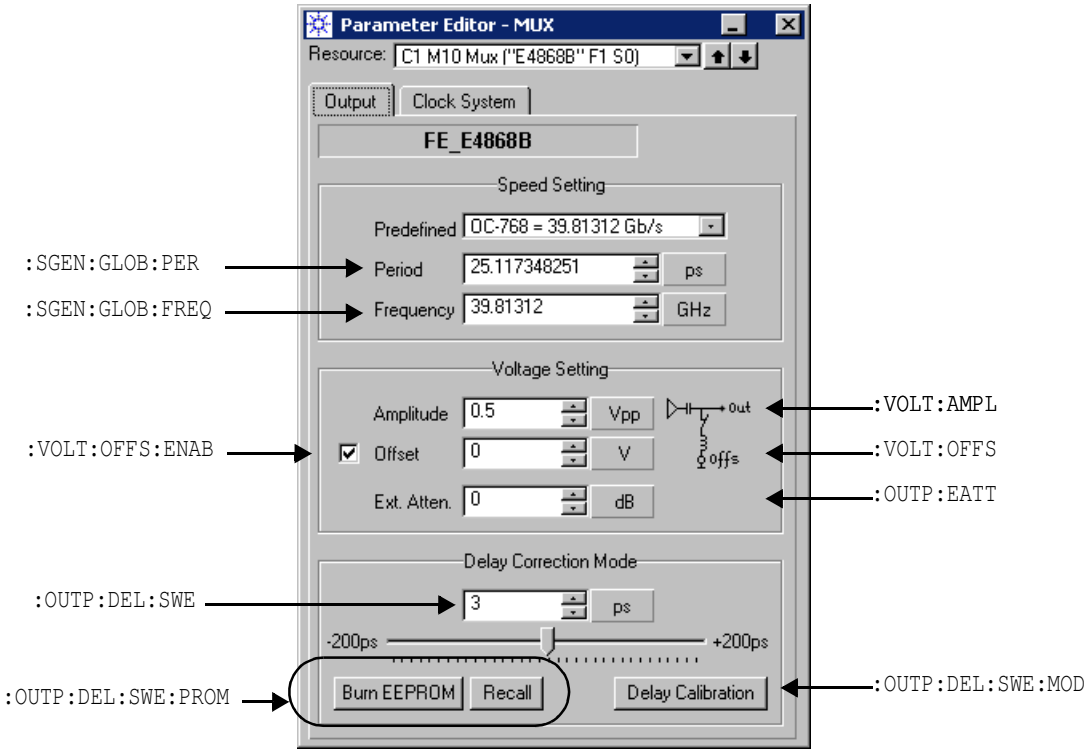
Parameter Group	Parameter	Command (on page)	Available on			
			E4868A	E4868B	E4869A	E4869B
Speed Parameters	Predefined Frequency		X	X	X	X
	Period	*":<Handle>:SGENeral:GLOBal:PERiod" on page 141*	X	X	X	X
	Frequency	*":<Handle>:SGENeral:GLOBal:FREQuency" on page 142*	X	X	X	X
Voltage Configuration	Amplitude	*":VOLTage:AMPLitude" on page 201*	X	X		
	Offset	*":VOLTage:OFFSet" on page 202*		X		
	Offset Enabling	*":VOLTage:OFFSet:ENABle" on page 203*		X		
	External Attenuator	*":OUTPut:EATTenuator" on page 204*	X	X		
Delay Correction and Calibration	Delay Value	*":OUTPut:DELay:SWEep" on page 205*	X	X		
	Delay Mode	*":OUTPut:DELay:SWEep:MODe" on page 206*	X	X		
	Delay Storage Handling	*":OUTPut:DELay:SWEep:PROM" on page 208*	X	X		
Offset Compensation		*":INPut:OCOMpensation" on page 209*			X	
Threshold		*":INPut:THReshold" on page 242*			X	X

www.valuetronics.com

Parameter Group	Parameter	Command (on page)	Available on			
			E4868A	E4868B	E4869A	E4869B
Sampling Phase Adjustment		*":INPut:DELay:SWEep" on page 249*				X
Clock System	External Clock Input	*":TRIGger" on page 210*	X	X	X	X
	Clock Output (Subrate)	*":TRIGger:MODE" on page 211*	X	X		

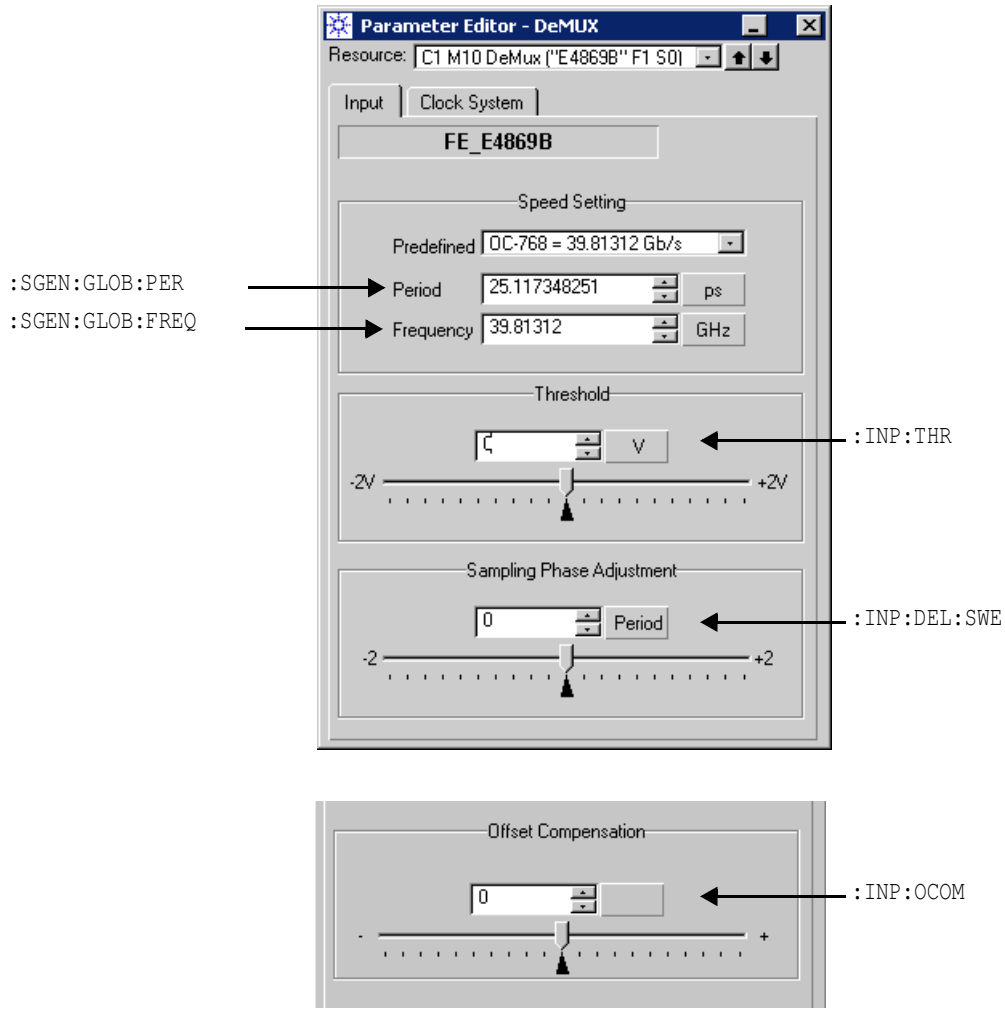
Commands in the User Interface

The following figure shows the output commands in the ParBERT user interface for a MUX module. Please note that not all commands are available for all modules. See the previous table for command availability.



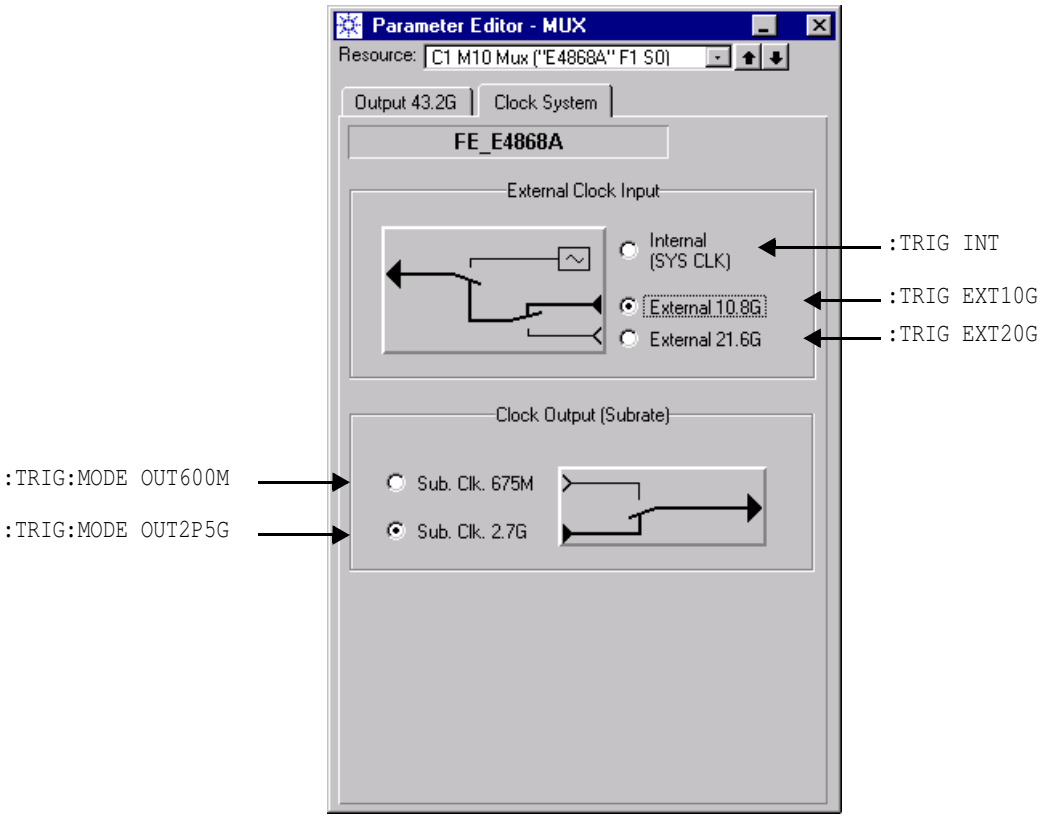
www.valuetronics.com

The following figures show the input commands in the ParBERT user interface for a DeMUX module. Please note that not all commands are available for all modules. See the previous table for command availability.

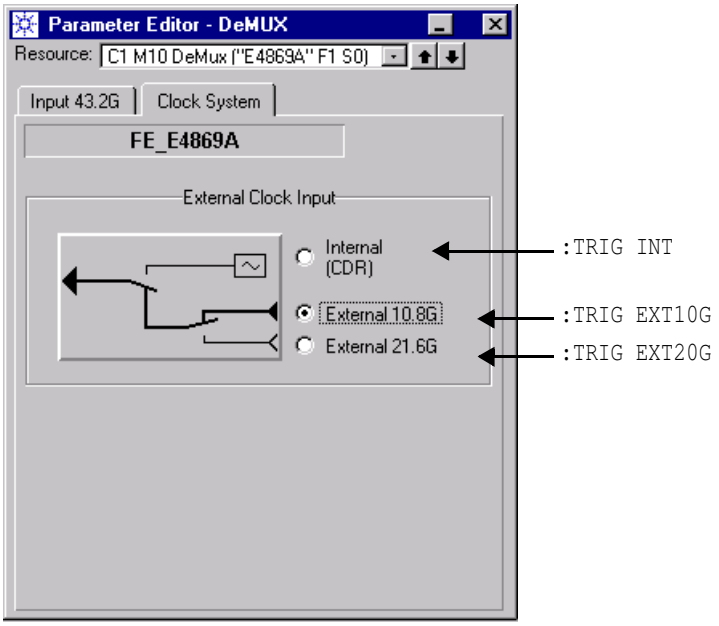


www.valuetronics.com

The following figure shows the clock system commands in the ParBERT user interface for a MUX module:



The following figure shows the clock system commands in the ParBERT user interface for a DeMUX module:



www.valuetronics.com

:VOLTage:AMPLitude

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector* [:SOURce]:VOLTage[:LEVel:IMMediate]:AMPLitude[:VALue]<value>

Description Sets the amplitude of the output signal of the MUX module.

NOTE It is possible to set the amplitude to values higher than are given in the product overview without generating warnings. However, it cannot be guaranteed that the values are still within the specifications. This feature has been added to provide more useability (for example, for additional stressing of the DUT).

Related Commands

- *":VOLTage:OFFSet" on page 202*
- *":OUTPut:EATTenuator" on page 204*

NOTE Because the output of the E4868A MUX module is AC coupled, only the amplitude (peak-to-peak voltage) can be specified.

For the E4868B MUX module, an offset can be enabled or disabled.

Parameters **<value>** Voltage value in Volts. For valid values, see *Agilent ParBERT 81250 43G Parallel Bit Error Ratio Tester Product Overview.*

Example

```
:_test:MOD10:CONN:VOLT:AMPL 1.0
```

:VOLTage:AMPLitude?

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector* [:SOURce]:VOLTage[:LEVel:IMMediate]:AMPLitude[:VALue]?

Description Returns the amplitude of the output signal.

Return Value Voltage (in Volts).

Example

```
:_test:MOD10:CONN:VOLT:AMPL?
```

might return:

```
1.0E+0
```


www.valuetronics.com

:VOLTage:OFFSet

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector* [:SOURce]:VOLTage[:LEVel:IMMediate]:OFFSet[:VALue] <value>

NOTE This command is only available for the E4868B modules.

Parameters **<value>** Offset voltage to be added to the generated signal, see the hardware specifications for the range.

Related Commands

- *":VOLTage:AMPLitude" on page 201*
- *":VOLTage:OFFSet:ENABle" on page 203*
- *":OUTPut:EATTenuator" on page 204*

Example

```
:_test:MOD10:CONN:VOLT:OFFS 0.5
```

:VOLTage:OFFSet?

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector* [:SOURce]:VOLTage[:LEVel:IMMediate]:OFFSet[:VALue]?

Return Value Returns the offset voltage (offset added to the generated signal).

Example

```
:_test:MOD10:CONN:VOLT:OFFS?
```

might return:

```
5.0E-1
```


www.valuetronics.com

:VOLTage:OFFSet:ENABle

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector* [:SOURce]:VOLTage[:LEVel:IMMediate]:OFFSet:ENABle <TRUE| FALSe>

NOTE This command is only available for the E4868B modules.

Description This command enables/disables voltage offset. If voltage offset is enabled and an external attenuator setting is specified, the offset voltage is checked and recalculated. If voltage offset is disabled, the offset voltage is ignored.

Related Commands

- *":VOLTage:AMPLitude" on page 201*
- *":VOLTage:OFFSet" on page 202*
- *":OUTPut:EATTenuator" on page 204*

Parameters **TRUE** Enables voltage offset.

FALSe Disables voltage offset.

Example

```
:_test:MOD10:CONN:VOLT:OFFS:ENAB TRUE
```

:VOLTage:OFFSet:ENABle?

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector* [:SOURce]:VOLTage[:LEVel:IMMediate]:OFFSet:ENABle?

Return Value Returns the current status of the voltage offset. See *":VOLTage:OFFSet:ENABle" on page 203* for a detailed description.

Example

```
:_test:MOD10:CONN:VOLT:OFFS:ENAB?
```

might return:

```
TRUE
```


www.valuetronics.com

:OUTPut:EATTenuator

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector*:OUTPut:EATTenuator <value>

Description This command allows you to enter the attenuation (in dB) of an external attenuator on the multiplexer's output. When this value is changed, the output values are also changed so that they reflect the values that actually reach the DUT (and not the values that reach the attenuator).

Related Commands

- *":VOLTage:AMPLitude" on page 201*
- *":VOLTage:OFFSet" on page 202*
- *":VOLTage:OFFSet:ENABle" on page 203*

Parameters **<value>** Attenuation value of the external attenuator in dB; the range is dependent on the offset voltage.

Example

```
:_test:_test::MOD10:CONN1:OUTP:EATT 20
```

:OUTPut:EATTenuator?

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector*:OUTPut:EATTenuator?

Return Value Returns the attenuation value that was entered (manually or by the program) for the external attenuator in dB; range: 0 – 120 dB.

Example

```
:_test:MOD10:CONN1:OUTP:EATT?
```

might return:

```
2.0E+1
```


www.valuetronics.com

:OUTPut:DELay:SWEep

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector*:OUTPut:DELay:SWEep <sweep>

Description Manipulates globally the delay for all generators bound to the inputs of the E4868A module. The values set with this command are added to the start delay (see *":PULSe:DELay" on page 179*) of those generators.

Parameters **<sweep>** The valid values depend on the current delay mode:

- Delay correction mode (`TCOM`): -200 ps to +200 ps.
- Delay calibration mode (`DCAL`): -2 ns to +2 ns

The current mode can be queried with *":OUTPut:DELay:SWEep:MODe?" on page 207*.

Example In delay correction mode, set the delay to 3 ps:

```
:_test:MOD10:CONN1:OUTP:DEL:SWE 3E-12
```

In delay calibration mode, set the delay to -100 ps:

```
:_test:MOD10:CONN1:OUTP:DEL:SWE -100E-12
```


www.valuetronics.com

:OUTPut:DELay:SWEep?

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector*:OUTPut:DELay:SWEep?

Description Returns the value for the delay.

Return Value The valid return values depend on the current delay mode:

- Delay correction mode: -200 ps to +200 ps.
- Delay calibration mode: -2ns to +2ns

The current mode can be queried with *":OUTPut:DELay:SWEep:MODe?" on page 207*.

Example For delay correction mode:

```
:_test:MOD10:CONN1:OUTP:DEL:SWE?
```

might return:

```
3.0E-12
```

For delay calibration mode:

```
:_test:MOD10:CONN1:OUTP:DEL:SWE?
```

might return:

```
-1.0E-10
```

:OUTPut:DELay:SWEep:MODe

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector*:OUTPut:DELay:SWEep:MODE TCOM| DCAL

Description This command is used to switch between the delay correction and delay calibration mode.

If you receive one of the ParBERT 43G bundles, all frontends and MUX/DEMUX modules are factory-calibrated and ready to use. These two modes allows you to add a common delay to all the generator frontends connected to the MUX module.

The delay calibration sets the base value for the delay correction. If the delay calibration value is changed, the delay correction value is reset to zero.

The system stores two delay calibration values —one for an external clock source, one for the internal clock source. To set/query the currently selected clock source, see *":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer][:SOURce]" on page 168*.

www.valuetronics.com

That means, the current delay value depends on the currently selected clock source and the current delay mode:

- If delay correction is set, the delay value depends on the delay calibration value (which depends on the selected clock source).
- If delay calibration (DCAL) is set, the delay value depends on the selected clock source.

Normally, the delay is automatically set with calibration tables, which enables the correct operation of the E4868A module.

Parameters **<TCOM|DCAL>** Mode to be set.

- TCOM

  (**T**emperature **COM**pensation: Delay Correction) Switches into delay correction mode. This mode allows you to correct delay fluctuations caused by variations in ambient temperature well above 20 °C, for instance.

- DCAL

  (**D**elay **CAL**ibration) Switches into delay calibration mode. This mode lets you recalibrate the delay if the clock module or the E4868A module was removed. This mode offers a wider timing range.

Example

```
:_test:MODule10:CONNector1:OUTPut:DELay:SWEeep:MODe TCOM
```

or

```
:_test:MODule10:CONNector1:OUTPut:DELay:SWEeep:MODe DCAL
```

:OUTPut:DELay:SWEep:MODe?

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector*:OUTPut:DELay:SWEep:MODe?

Description This command returns the current delay mode.

Return Value **TCOM** (**T**emperature **COM**pensation: Delay Correction) The module is in delay correction mode. This mode allows you to correct delay fluctuations caused by variations in ambient temperature well above 20 °C, for instance.

DCAL (**D**elay **CAL**ibration) The module is in delay calibration mode. This mode lets you recalibrate the delay if the clock module or the E4868A module was removed. This mode offers a wider timing range.

www.valuetronics.com

Example :_test:MODule10:CONNector1:OUTPut:DELay:SWEeep:MODe?

might return:

`TCOM` or `DCAL`

:OUTPut:DELay:SWEep:PROM

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector*:OUTPut:DELay:SWEep:PROM STORe | RECall

Description The delay values of an E4868 MUX module consist of three values: One delay calibration value if the built-in clock module generates the system clock, one delay calibration value if an external clock source is used, and one delay correction value. These values are stored in an EEPROM inside the module. They are hence non-volatile and independent of the ParBERT controller or the loaded setting.

If you change the delay values, your changes take immediate effect and remain effective until the ParBERT system is switched off. They are stored in the module. They remain effective, even if you terminate and restart the ParBERT user interface. But if you restart the ParBERT system, the values stored in the EEPROM become effective.

This command allows you to burn the currently active values to the EEPROM, or to restore the current values to the defaults from the EEPROM.

Related Commands

- *":OUTPut:DELay:SWEep" on page 205*
- *":OUTPut:DELay:SWEep:MODe" on page 206*

Parameters **STORe** Writes the currently used delay parameters to the EEPROM.

RECall Overwrites the currently used delay parameters with the last stored values from the EEPROM.

Example `:_test:MOD10:CONN1:OUTP:DEL:SWE:PROM REC`

www.valuetronics.com

:INPut:OCOMpensation

NOTE Valid only for E4869A/B (DEMUX) modules.

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector*:INPut:OCOMpensation <value>

Definition By default, the decision threshold for sampling the incoming signal is set according to the mean value of the data signal. This is adequate for pure PRBS data. In cases where the received pattern has an unbalanced ratio of zeros to ones, this command allows you to fine-tune the threshold to an appropriate value.

An inadequate offset can be suspected, if:

- None of the analyzer terminals could synchronize.
- All analyzer terminals did synchronize, but exhibit an abnormally high BER.
- Some analyzer terminals did synchronize but show a BER close to the synchronization threshold, while others did not synchronize at all.

See also "*How to Change the Input Parameters of a DEMUX Module*" in the *System User Guide* for further details.

Parameters **<value>** New value of offset compensation; range: ±1 (corresponds to roughly ±100 mV).

Example
```
:_test:MOD10:CONN1:INP:OCOM 0.9
```

:INPut:OCOMpensation?

NOTE Valid only for E4869A/B (DMUX) modules.

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector*:INPut:OCOMpensation?

Definition Returns value used for the offset compensation.

Return Value Current value for offset compensation; range: ±1 (corresponds to roughly ±100 mV).

Example
```
:_test:MOD10:CONN1:INP:OCOM?
```

might return:

```
9.0E-1
```


www.valuetronics.com

:TRIGger

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector* :TRIGger[:SEQuence:LAYer:SOURce] INT| EXT10G| EXT20G

Description Sets the source of the clock signal used for the timing of the E4868A MUX module.

Parameters **<INT | EXT10G | EXT20G>**

INT When this is selected, the module uses the clock signal provided by the central clock module.

NOTE The SYS CLK output from the central clock module must be connected to the SYS CLK input of the MUX module.

EXT10G The module uses an external signal in the range of 10.8 GHz provided by an external clock generator. The external clock must be connected to the Ext.Clk. input port.

EXT20G The module uses an external signal in the range of 21.6 GHz. provided by an external clock generator.

Example

```
DRSA:MOD10:CONN:TRIG INT
DRSA:MOD10:CONN:TRIG EXT10G
DRSA:MOD10:CONN:TRIG EXT20G
```

:TRIGger?

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector* :TRIGger[:SEQuence:LAYer:SOURce]?

Return Value Returns the source of the clock signal used for the timing of the E4868A multiplexer.

Example

```
DRSA:MOD10:CONN:TRIG?
```

might return:

```
INT or EXT10G or EXT20G
```


www.valuetronics.com

:TRIGger:MODE

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector*:TRIGger:MODE OUT600M| OUT2P5G

Parameters **<OUT600M | OUT2P5G>** Specifies the speed of the subrate clock output signal.

- OUT600M

  Provides a signal of up to 675 MHz at the subrate clock output connector.

- OUT2P5G

  Provides a signal of up to 2.7 GHz at the subrate clock output connector.

Example

```
DRSA:MOD10:CONN:TRIG:MODE OUT600M
DRSA:MOD10:CONN:TRIG:MODE OUT2P5G
```

:TRIGger:MODE?

Syntax :<Handle>[:CGRoup*]:MODule*:CONNector*:TRIGger:MODE?

Return Value Returns the speed of the subrate clock output signal. Can be either:

- OUT600M

  Provides a subrate clock output signal in the range of 675 MHz.

- OUT2P5G

  Provides a subrate clock output signal in the range of 2.7 GHz.

Example

```
DRSA:MOD10:CONN:TRIG:MODE?
```

might return:

```
OUT600M or OUT2P5G
```


www.valuetronics.com

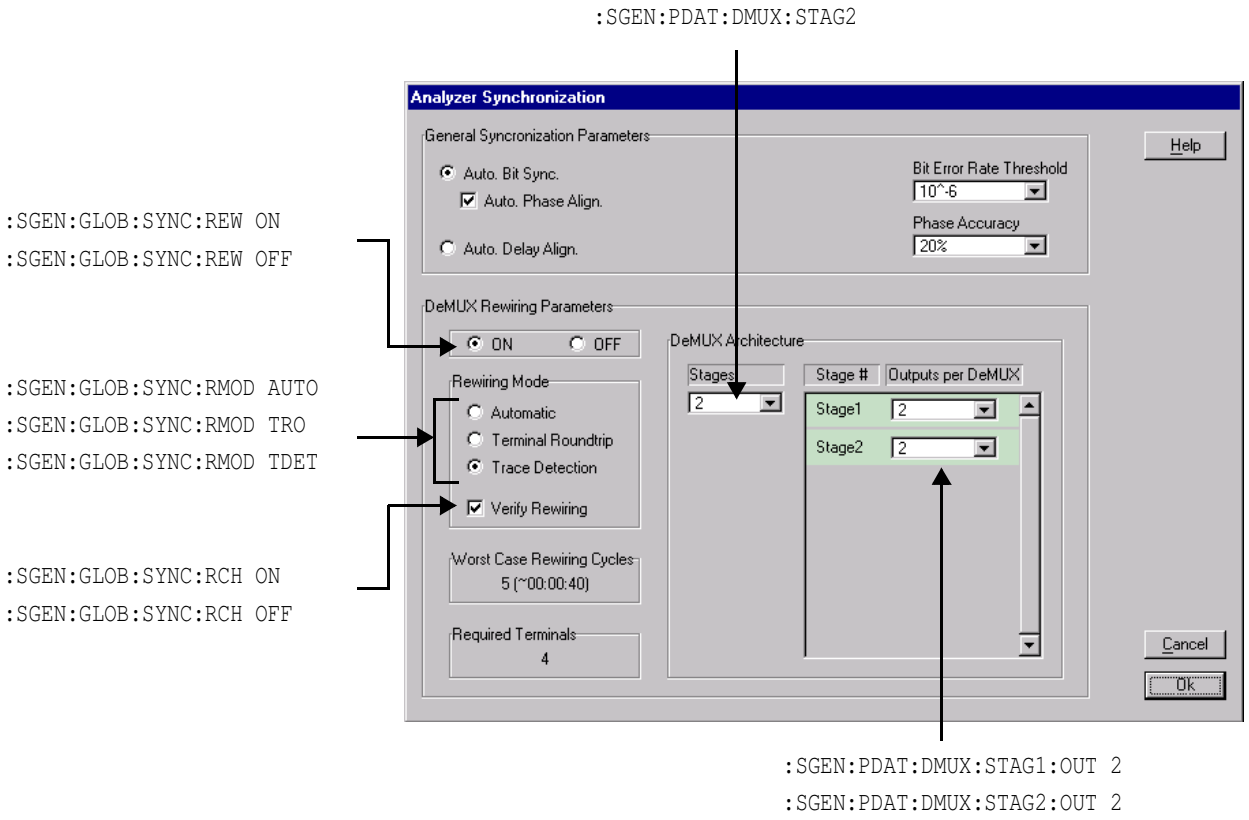
DeMUX Rewiring Parameter Commands

The following commands are available:

Parameter Group	Parameter	Command (on page)
Activation	Rewire	*":<Handle>:SGENeral:GLOBal:SYNChronization:REWire" on page 214*
Configuration	Rewiring Mode	*":<Handle>:SGENeral:GLOBal:SYNChronization:RMODe" on page 215*
Verification	Rewiring Check	*":<Handle>:SGENeral:GLOBal:SYNChronization:RCH" on page 216*
Multi-Stages DEMUX Settings	Number of Stages	*":<Handle>:SGENeral:PDATa:DMUX:STAGe" on page 217*
	Number of Outputs	*":<Handle>:SGENeral:PDATa:DMUX:STAGe:OUT" on page 218*

www.valuetronics.com

The following figure shows the parameter commands in the Agilent 81250 user interface.



www.valuetronics.com

:<Handle>:SGENeral:GLOBal:SYNChronization:REWire

Syntax :<Handle>:SGENeral:GLOBal:SYNChronizat:REWire ON| OFF

Description Switches the DeMUX rewiring on and off.

If DeMUX rewiring is switched on, DeMUX rewiring takes place when the measurement is started. There is no reaction of the system until rewiring is completed or failed. Because DeMUX rewiring can take a lot of time, it is recommended to set all other system parameters first.

For more information on DeMUX rewiring, please refer to *"Automatic Rewiring of Demultiplexer Terminals"* in the *Agilent 81250 Parallel Bit Error Ratio Tester System User Guide.*

Parameters **<ON|OFF>** Switches DeMUX rewiring on or off.

Example

```
:_test:SGEN:GLOB:SYNC:REW ON
```

or

```
:_test:SGEN:GLOB:SYNC:REW OFF
```

:<Handle>:SGENeral:GLOBal:SYNChronization:REWire?

Syntax :<Handle>:SGENeral:GLOBal:SYNChronizat:REWire?

Return Value Returns the current state of DeMUX rewiring.

Example

```
:_test:SGEN:GLOB:SYNC:REW?
```

might return:

```
ON or OFF
```


www.valuetronics.com

:<Handle>:SGENeral:GLOBal: SYNChronization:RMODe

Syntax :<Handle>:SGENeral:GLOBal:SYNChronizat:RMODe <AUTO| TRO| TDET>

Parameters **<AUTO | TRO | TDET>** Sets the modes for DeMUX rewiring.

- AUTO

  The automatic mode tries first to rewire with the trace detection method and, if that fails, with the terminal roundtrip method.

- TRO

  (**T**erminal **RO**undtrip) Rewire takes place with the terminal roundtrip.

- TDET

  (**T**race **DET**ection) Rewire takes place with the trace detection method.

For more information on DeMUX rewiring modes, please refer to *"DEMUX Rewiring Modes"* in the *Agilent 81250 Parallel Bit Error Ratio Tester System User Guide*.

Example

```
:_test:SGENeral:GLOBal:SYNCchronization:RMODe AUTO
```

or

```
:_test:SGENeral:GLOBal:SYNCchronization:RMODe TRO
```

or

```
:_test:SGENeral:GLOBal:SYNCchronization:RMODe TDET
```


www.valuetronics.com

:<Handle>:SGENeral:GLOBal:SYNChronization:RMODe?

Syntax :<Handle>:SGENeral:GLOBal:SYNChronizat:RMODe?

Return Value Returns the current mode for DeMUX rewiring. Can be:

- AUTO

  The automatic mode tries first to rewire with the trace detection method and, if that fails, with the terminal roundtrip method.

- TRO

  (**T**erminal **RO**undtrip) Rewire takes place with the terminal roundtrip.

- TDET

  (**T**race **DET**ection) Rewire takes place with the trace detection method.

Example

```
:_test:SGENeral:GLOBal:SYNCchronization:RMODe?
```

might return:

```
AUTO or TRO or TDET
```

:<Handle>:SGENeral:GLOBal:SYNChronization:RCH

Syntax :<Handle>:SGENeral:GLOBal:SYNChronizat:RCHeck ON| OFF

Parameters **<ON|OFF>** Switches the rewire verify mode on or off.

If verify rewiring is switched on, the system performs additional inspections and complains about possible errors.

If verify rewiring is switched off, the system tries to ignore noncritical problems to carry out a measurement.

Example

```
:_test:SGEN:GLOB:SYNC:RCH ON
```

or

```
:_test:SGEN:GLOB:SYNC:RCH OFF
```


www.valuetronics.com

:<Handle>:SGENeral:GLOBal:SYNChronization:RCH?

Syntax :<Handle>:SGENeral:GLOBal:SYNChronizat:RCHeck?

Return Value Returns the current rewire verify mode (on or off).

If verify rewiring is switched on, the system performs additional inspections and complains about possible errors.

If verify rewiring is switched off, the system tries to ignore noncritical problems to carry out a measurement.

Example

```
:_test:SGEN:GLOB:SYNC:RCH?
```

might return:

```
ON or OFF
```

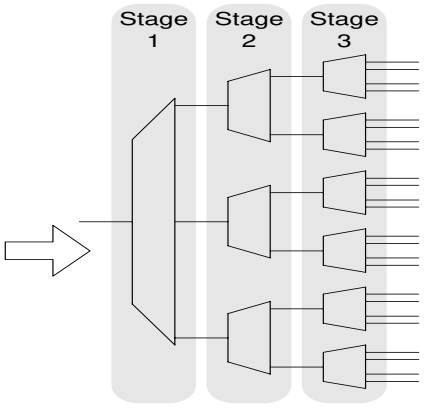
:< Handle> :SGENeral:PDATa:DMUX:STAGe

Syntax :<Handle>:SGENeral:PDATa*:DMUX:STAGe*[:VALUE]

Description Specifies the number of stages of the DeMUX module. To set this value, you need to know the architecture of the demultiplexer.

Parameters *. Number of stages of the DeMUX module.

Example For a demultiplexer with the following architecture:




```
:_test:SGEN:PDATA:DMUX:STAG 3
```


www.valuetronics.com

:<Handle>:SGENeral:PDATa:DMUX:STAGe?

Syntax :<Handle>:SGENeral:PDATa*:DMUX:STAGe*[:VALUE]?

Description Returns the number of stages of the DeMUX module.

Return Value Number of stages.

Example

```
:_test:SGEN:PDATA:DMUX:STAG?
```

might return:

```
3
```

:<Handle>:SGENeral:PDATa:DMUX:STAGe:OUT

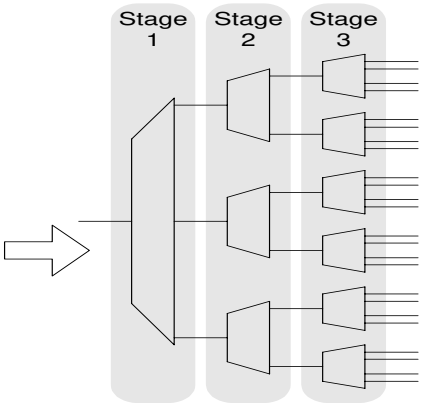
Syntax :<Handle>:SGENeral:PDATa*:DMUX:STAGe*:OUTputs[:VALUE]<value>

Description Sets the number of outputs per DeMUX module for a specified stage.

Parameters * Index of the stage.

<value> Number of outputs per DeMUX module for a specified stage.

Example For a demultiplexer with the following architecture:




```
:_test:SGEN:PDATA:DMUX:STAG1:OUT 3
```

or

```
:_test:SGEN:PDATA:DMUX:STAG2:OUT 2
```

or

```
:_test:SGEN:PDATA:DMUX:STAG3:OUT 4
```


www.valuetronics.com

:< Handle> :SGENeral:PDATa:DMUX:STAGe:OUT?

Syntax :<Handle>:SGENeral:PDATa*:DMUX:STAGe*:OUTputs[:VALUE]?

Description Returns the number of outputs per DeMUX module for a specified stage.

Return Value Number of outputs per DeMUX module for a specified stage.

Example

```
:_test:SGEN:PDATA:DMUX:STAG1:OUT?
```

might return:

```
3
```


www.valuetronics.com

Level Parameter Commands

The commands for specifying the level parameters are available on port, terminal and connector level. The parameters can be specified separately for pulse and data ports.

:VOLTage[:LEVel][:IMMediate]:HIGH

Syntax

:<Handle>:SGENeral:**PDATa**(*)[:SOURce]:VOLTage[:LEVel][:IMMediate]:HIGH <HighLevel>

:<Handle>:SGENeral:**PDATa**(*)**:TERMinal**(*)[:SOURce]:VOLTage[:LEVel][:IMMediate]:HIGH <HighLevel>

:<Handle>:SGENeral:**PPULse**(*)[:SOURce]:VOLTage[:LEVel][:IMMediate]:HIGH <HighLevel>

:<Handle>:SGENeral:**PPULse**(*)**:TERMinal**(*)[:SOURce]:VOLTage[:LEVel][:IMMediate]:HIGH <HighLevel>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector**(*)[:SOURce]:VOLTage[:LEVel][:IMMediate]:HIGH <HighLevel>

Parameters

<HighLevel> High Level value in Volts.

Sets the peak of a time varying signal.

Valid range depends on the frontend used:

E4838A: –2.15 to 4.40 V
E4843A: –1.75 to 3.10 V
These ranges are valid for a termination voltage of 0V (50 Ohm). With other termination voltages other values can be reached (for example, PECL Levels).

Example

```
:_test:SGEN:PDAT1:VOLT:HIGH 3

:_test:SGEN:PDAT1:TERM1:SOUR:VOLT:LEV:IMM:HIGH 1

:_test:SGEN:PPUL1:VOLT:HIGH 3

:_test:SGEN:PPUL1:TERM1:SOUR:VOLT:LEV:IMM:HIGH 1.0

:_test:mod2:conn4:volt:high 2.5
```


www.valuetronics.com

:VOLTage[:LEVel][:IMMediate]:HIGH?

Syntax

:<Handle>:SGENeral:**PDATa(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate]:HIGH?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate]:HIGH?

:<Handle>:SGENeral:**PPULse(*):**[:SOURce]:VOLTage[:LEVel][:IMMediate]:HIGH?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate]:HIGH?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate]:HIGH?

Return Value

Returns the high voltage level of the signal.

Example

```
:_test:SGEN:PDAT1:VOLT:HIGH?
:_test:SGEN:PDAT1:TERM1:SOUR:VOLT:LEV:IMM:HIGH?
:_test:SGEN:PPUL1:VOLT:HIGH?
:_test:SGEN:PPUL1:TERM1:SOUR:VOLT:LEV:IMM:HIGH?
:_test:MOD2:CONN4:VOLT:HIGH?
```

might return:

```
1.0000000E+000
```


www.valuetronics.com

:VOLTage[:LEVel][:IMMediate]:LOW

Syntax

:<Handle>:SGENeral:**PDATa**(*)[:SOURce]:VOLTage[:LEVel][:IMMediate]:LOW <LowLevel>

:<Handle>:SGENeral:**PDATa**(*)**:TERMinal**(*)[:SOURce]:VOLTage[:LEVel][:IMMediate]:LOW <LowLevel>

:<Handle>:SGENeral:**PPULse**(*)[:SOURce]:VOLTage[:LEVel][:IMMediate]:LOW <LowLevel>

:<Handle>:SGENeral:**PPULse**(*)**:TERMinal**(*)[:SOURce]:VOLTage[:LEVel][:IMMediate]:LOW <LowLevel>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector**(*)[:SOURce]:VOLTage[:LEVel][:IMMediate]:LOW <LowLevel>

Parameters

<LowLevel> Low Level value in Volts (<NRf>).

Sets the low level of the time varying signal.

Valid range depends on the frontend used:

E4838A: –2.2 to 4.35 V
E4843A: –2.20 to 2.95 V
These ranges are valid for a termination voltage of 0V (50 Ohm). With other termination voltages other values can be reached (for example, PECL Levels).

Example

```
:_test:SGEN:PDAT1:VOLT:LOW -1

:_test:SGEN:PDAT1:TERM1:SOUR:VOLT:LEV:IMM:LOW -1

:_test:SGEN:PPUL1:VOLT:LOW -1

:_test:SGEN:PPUL1:TERM1:SOUR:VOLT:LEV:IMM:LOW 0.1

:_test:MOD2:CONN4:VOLT:LOW 0.3
```


www.valuetronics.com

:VOLTage[:LEVel][:IMMediate]:LOW?

Syntax

:<Handle>:SGENeral:**PDATa(*):**[:SOURce]:VOLTage[:LEVel][:IMMediate]:LOW?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate]:LOW?

:<Handle>:SGENeral:**PPULse(*):**[:SOURce]:VOLTage[:LEVel][:IMMediate]:LOW?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate]:LOW?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate]:LOW?

Return Value

Returns the low voltage level.

Example

```
:_test:SGEN:PDAT1:VOLT:LOW?
:_test:SGEN:PDAT1:TERM1:SOUR:VOLT:LEV:IMM:LOW?
:_test:SGEN:PPUL1:VOLT:LOW?
:_test:SGEN:PPUL1:TERM1:SOUR:VOLT:LEV:IMM:LOW?
:_test:MOD2:CONN4:VOLT:LOW?
```

might return:

```
-1.000000E+000
```


www.valuetronics.com

:VOLTage[:LEVel][:IMMediate]: CAConfiguration:LOW

Syntax

:<Handle>:SGENeral:**PDATa(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate] :CAConfiguration:LOW <AdditionalLowLevel>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:VOLTage[:LEVel] [:IMMediate]:CAConfiguration:LOW <AdditionalLowLevel>

:<Handle>:SGENeral:**PPULse(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate] :CAConfiguration:LOW <AdditionalLowLevel>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:VOLTage[:LEVel] [:IMMediate]:CAConfiguration:LOW <AdditionalLowLevel>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:VOLTage[:LEVel] [:IMMediate]:CAConfiguration:LOW <AdditionalLowLevel>

Parameters

<AdditionalLowLevel> Additional low level value in Volts (<NRf>).

Sets the additional low level of the pulse, which is available only when the A2 channel-add mode is selected for the port/terminal/connector (see *":OUTPut:CAConfiguration[:MODE]" on page 267*).

This command is only supported for E4838A frontends.

For more details on the channel-add mode, see *"How to Add Channels in Analog Mode"* in the *Agilent 81250 System User Guide.*

Example

```
:_test:SGEN:PDAT1:VOLT:CAC:LOW -2
:_test:SGEN1:PDAT1:TERM1:SOUR:VOLT:LEV:IMM:CAC:LOW 0.4
:_test:SGEN:PPUL1:VOLT:CAC:LOW -2
:_test:SGEN1:PPUL1:TERM1:SOUR:VOLT:LEV:IMM:CAC:LOW 0.4
:_test:CGR1:MOD2:CONN3:SOUR:VOLT:LEV:IMM:CAC:LOW 1
```


www.valuetronics.com

:VOLTage[:LEVel][:IMMediate]:CAConfiguration:LOW?

Syntax

:<Handle>:SGENeral:**PDATa(*):**[:SOURce]:VOLTage[:LEVel][:IMMediate]:CAConfiguration:LOW?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate]:CAConfiguration:LOW?

:<Handle>:SGENeral:**PPULse(*):**[:SOURce]:VOLTage[:LEVel][:IMMediate]:CAConfiguration:LOW?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate]:CAConfiguration:LOW?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate]:CAConfiguration:LOW?

Return Value

Returns the additional low-level of an E4838A frontend in A2 channel-add mode.

Example

```
:_test:SGEN:PDAT1:VOLT:CAC:LOW?
```

```
:_test:SGEN1:PDAT1:TERM1:SOUR:VOLT:LEV:IMM:CAC:LOW?
```

```
:_test:SGEN:PPUL1:VOLT:CAC:LOW?
```

```
:_test:SGEN1:PPUL1:TERM1:SOUR:VOLT:LEV:IMM:CAC:LOW?
```

```
:_test:CGR1:MOD2:CONN3:SOUR:VOLT:LEV:IMM:CAC:LOW?
```

might return:

```
4.000000E-001
```


www.valuetronics.com

:VOLTage[:LEVel][:IMMediate]:SEMode

Syntax :<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate]:SEMode <New Value>

NOTE This command is only applicable to the N4868A 10.8G Booster modules. See the *System User Guide* for detailed information about how to work with the Booster modules.

Description Defines whether single-ended or differential channel operation mode should be used (TRUE = single-ended mode, FALSE = differential mode).

Parameter **<New Value>** New value used for the SEMode. When set to `TRUE`, single-ended mode is enabled. When set to `FALSE`, differential mode is enabled.

Example

```
:_test:CGR1:MOD2:CONN3:SOUR:VOLT:LEV:IMM:SEM TRUE
```


www.valuetronics.com

:VOLTage[:LEVel][:IMMediate]:SEMode?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate]:SEMode?

NOTE This command is only applicable to the N4868A 10.8G Booster modules. See the *System User Guide* for detailed information about how to work with the Booster modules.

Return Value Returns whether or not single-ended mode is currently enabled. When `TRUE`, single-ended mode is enabled. When `FALSE`, differential mode is enabled.

Example

```
:_test:CGR1:MOD2:CONN3:SOUR:VOLT:LEV:IMM:SEM?
```

might return:

```
TRUE
```

:VOLTage[:LEVel][:IMMediate]:SEMode:AVAilable?

Syntax :<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**[:SOURce]:VOLTage[:LEVel][:IMMediate]:SEMode:AVAilable?

NOTE This command is only applicable to the N4868A 10.8G Booster modules. See the *System User Guide* for detailed information about how to work with the Booster modules.

It is a good idea to implement this query in any SCPI scripts with the SEMode command to address future compatibility issues.

Return Value Returns whether or not single-ended mode is available.

Example

```
:_test:CGR1:MOD2:CONN3:SOUR:VOLT:LEV:IMM:SEM:AVA?
```

might return:

```
TRUE
```


www.valuetronics.com

:SENSe:VOLTage:RANGe

Syntax

:<Handle>:SGENeral:**PDATa(*)**:SENSe:VOLTage:RANGe <range>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:SENSe:VOLTage:RANGe <range>

:<Handle>:CGRoup(*):MODule(*):**CONNector(*)**:SENSe:VOLTage:RANGe <range>

Parameters

<range> Selects the allowed input voltage range. For E4835A differential input frontends two values are supported:

- 3 – allows input voltages –2 ... 3 V,
- 5 – allows input voltages 0 ... 5 V.

For E4863A/E4865A frontends the following values are supported:

- 0 – allows input voltages –2 ... 0 V,
- 1 – allows input voltages –1 ... 1 V,
- 2 – allows input voltages 0 ... 2 V,

Example

```
:_test:SGEN:PDAT1:INP:RANG 3
:_test:SGEN:PDAT1:TERM1:SENS:RANG 5
:_test:CGROUP1:MODULE2:CONNECTOR4:SENSE:VOLTAGE:RANGE 3
```


www.valuetronics.com

:SENSe:VOLTage:RANGe?

Syntax

:<Handle>:SGENeral:**PDATa(*)**:SENSe:VOLTage:RANGe?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:SENSe:VOLTage:RANGe?

:<Handle>:CGRoup(*):MODule(*):**CONNector(*)**:SENSe:VOLTage:RANGe?

Return Value

Returns the currently selected input voltage range. For E4835A differential input frontends, two values are possible:

- 3 – allows input voltages –2 ... 3 V,
- 5 – allows input voltages 0 ... 5 V.

For E4863A/E4865A frontends, the following values are possible:

- 0 – allows input voltages –2 ... 0 V,
- 1 – allows input voltages –1 ... 1 V,
- 2 – allows input voltages 0 ... 2 V.

Example

```
:_test:SGEN:PDAT1:SENS:VOLT:RANG?
:_test:SGEN:PDAT1:TERM1:SENS:VOLT:RANG?
:_test:CGROUP1:MODULE2:CONNECTOR4:SENSE:VOLTAGE:RANGE?
```

might return:

```
3.0E+0
```


www.valuetronics.com

Delay Control Parameter Commands

The delay control parameter commands are available on port, terminal and connector level. The parameters can be specified separately for pulse and data ports.

:DCONtrol[:DEViation]?

Syntax

:<Handle>:SGENeral:PDATa(*):[:SOURce]:DCONtrol:[DEViation]?

:<Handle>:SGENeral:PDATa(*):TERMinal(*)[:SOURce]:DCONtrol:[DEViation]?

:<Handle>:SGENeral:PPULse(*):[:SOURce]:DCONtrol:[DEViation]?

:<Handle>:SGENeral:PPULse(*):TERMinal(*)[:SOURce]:DCONtrol:[DEViation]?

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*)[:SOURce]:DCONtrol:[DEViation]?

Return Value

Returns the current maximum delay control deviation in seconds.

Example

```
:_test:SGEN:PDAT1:DCONT?

:_test:SGEN:PDAT1:TERM1:SOUR:DCONT:DEV?

:_test:SGEN:PPUL1:DCONT?

:_test:SGEN:PPUL1:TERM1:SOUR:DCONT?

:_test:MODULE2:CONNECTOR3:DCONT:DEV?
```

might return:

```
5.0E-11
```


www.valuetronics.com

:DCONtrol:SENSitivity

Syntax

:<Handle>:SGENeral:PDATa(*):[:SOURce]:DCONtrol:SENSitivity <Sensitivity>

:<Handle>:SGENeral:PDATa(*):TERMinal(*)[:SOURce]:DCONtrol:SENSitivity <Sensitivity>

:<Handle>:SGENeral:PPULse(*):[:SOURce]:DCONtrol:SENSitivity <Sensitivity>

:<Handle>:SGENeral:PPULse(*):TERMinal(*)[:SOURce]:DCONtrol:SENSitivity <Sensitivity>

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*)[:SOURce]:DCONtrol:SENSitivity <Sensitivity>

Description

Sets the delay control sensitivity of the external delay control modulation signal.

Parameters

<Sensitivity> Delay control sensitivity in seconds per volt (<NRf>).

Example

```
:_test:SGEN:PDAT1:DCONT:SENS 5.0E-10
:_test:SGEN:PDAT1:TERM1:SOUR:DCONT:SENS 5.0E-10
:_test:SGEN:PPUL1:DCONT:SENS 5.0E-11
:_test:SGEN:PPUL1:TERM1:SOUR:DCONT:SENS 5.0E-10
:_test:MODULE2:CONNECTOR3:DCONtrol:SENSITIVITY 5.0E-11
```

:DCONtrol:SENSitivity?

Syntax

:<Handle>:SGENeral:PDATa(*):[:SOURce]:DCONtrol:SENSitivity?

:<Handle>:SGENeral:PDATa(*):TERMinal(*)[:SOURce]:DCONtrol:SENSitivity?

:<Handle>:SGENeral:PPULse(*):[:SOURce]:DCONtrol:SENSitivity?

:<Handle>:SGENeral:PPULse(*):TERMinal(*)[:SOURce]:DCONtrol:SENSitivity?

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*)[:SOURce]:DCONtrol:SENSitivity?

Return Value

Returns the current delay control sensitivity in seconds per volt.

Example

```
:_test:SGEN:PDAT1:DCONT:SENS?
:_test:SGEN:PDAT1:TERM1:SOUR:DCONT:SENS?
:_test:SGEN:PPUL1:DCONT:SENS?
:_test:SGEN:PPUL1:TERM1:SOUR:DCONT:SENS?
:_test:MODULE2:CONNECTOR3:DCONTROL:SENSITIVITY?
```

might return:

```
5.0E-11
```


www.valuetronics.com

:DCONtrol:SOURce

Syntax

:<Handle>:SGENeral:PDATa(*):[:SOURce]:DCONtrol:SOURce <NAV| EXT>

:<Handle>:SGENeral:PDATa(*):TERMinal(*)[:SOURce]:DCONtrol:SOURce <NAV| EXT>

:<Handle>:SGENeral:PPULse(*):[:SOURce]:DCONtrol:SOURce <NAV| EXT>

:<Handle>:SGENeral:PPULse(*):TERMinal(*)[:SOURce]:DCONtrol:SOURce <NAV| EXT>

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*)[:SOURce]:DCONtrol:SOURce <NAV| EXT>

Parameters

<NAV | EXT> Sets the delay control source to either NAV (not available) or EXT (external).

The E4862B frontends support only EXT. All other frontends support only NAV.

Example

:_test:SGEN:PDAT1:DCONT:SOUR EXT

:_test:SGEN:PDAT1:TERM1:SOUR:DCONT:SOUR EXT

:_test:SGEN:PPUL1:DCONT:SOUR EXT

:_test:SGEN:PPUL1:TERM1:SOUR:DCONT:SOUR EXT

:_test:MODULE2:CONNECTOR3:DCONTROL:SOURCE EXT

www.valuetronics.com

:DCONtrol:SOURce?

Syntax

:<Handle>:SGENeral:PDATa(*):[:SOURce]:DCONtrol:SOURce?

:<Handle>:SGENeral:PDATa(*):TERMinal(*)[:SOURce]:DCONtrol:SOURce?

:<Handle>:SGENeral:PPULse(*):[:SOURce]:DCONtrol:SOURce?

:<Handle>:SGENeral:PPULse(*):TERMinal(*)[:SOURce]:DCONtrol:SOURce?

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*)[:SOURce]:DCONtrol:SOURce?

Return Value

Returns the current delay control source: either NAV (not available) or EXT (external).

The E4862B frontends support only EXT. All other frontends support only NAV.

Example

```
:_test:SGEN:PDAT1:DCONT:SOUR?
:_test:SGEN:PDAT1:TERM1:SOUR:DCONT:SOUR?
:_test:SGEN:PPUL1:DCONT:SOUR?
:_test:SGEN:PPUL1:TERM1:SOUR:DCONT:SOUR?
:_test:MODULE2:CONNECTOR3:DCONTROL:SOURCE?
```

might return:

```
EXT
```


www.valuetronics.com

:DCONtrol:STATe

NOTE Executing this command while the system is running causes the system to be restarted.

Syntax

:<Handle>:SGENeral:PDATa(*):[:SOURce]:DCONtrol:STATe <ON | OFF>

:<Handle>:SGENeral:PDATa(*):TERMinal(*)[:SOURce]:DCONtrol:STATe <ON | OFF>

:<Handle>:SGENeral:PPULse(*):[:SOURce]:DCONtrol:STATe <ON | OFF>

:<Handle>:SGENeral:PPULse(*):TERMinal(*)[:SOURce]:DCONtrol:STATe <ON | OFF>

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*)[:SOURce]:DCONtrol:STATe <ON | OFF>

Parameters

<ON | OFF> Switches the specified external delay control input connector on or off.

Example

```
:_test:SGEN:PDAT1:DCONT:STAT ON

:_test:SGEN:PPUL1:TERM1:SOUR:DCONT:STAT ON

:_test:MODULE2:CONNECTOR3:DCONTROL:STATE ON
```


www.valuetronics.com

:DCONtrol:STATe?

Syntax

:<Handle>:SGENeral:PDATa(*):[:SOURce]:DCONtrol:STATe?

:<Handle>:SGENeral:PDATa(*):TERMinal(*)[:SOURce]:DCONtrol:STATe?

:<Handle>:SGENeral:PPULse(*):[:SOURce]:DCONtrol:STATe?

:<Handle>:SGENeral:PPULse(*):TERMinal(*)[:SOURce]:DCONtrol:STATe?

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*)[:SOURce]:DCONtrol:STATe?

Return Value

Returns the current state of the specified external delay control input connector.

Example

```
:_test:SGEN:PDAT1:DCONT:STAT?
:_test:SGEN:PDAT1:TERM1:SOUR:DCONT:STAT?
:_test:SGEN:PPUL1:DCONT:STAT?
:_test:SGEN:PPUL1:TERM1:SOUR:DCONT:STAT?
:_test:MODULE2:CONNECTOR3:DCONTROL:STATE?
```

might return:

```
OFF
```


www.valuetronics.com

Input Parameter Commands

The input parameter commands are available on port, terminal and connector level.

:INPut[:STATe]

Syntax :<Handle>:SGENeral:**PDATa(*):INPut**[:STATe] <ON | OFF>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**[:STATe] <ON | OFF>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**[:STATe] <ON | OFF>

Parameters **<ON | OFF>** Switch the specified analyzer input connector ON or OFF.

Example

```
:_test:SGEN:PDAT1:INPUT ON
:_test:SGEN1:PDAT1:TERM1:INPUT ON
:_test:CGROUP1:MODULE2:CONNECTOR4:INPUT ON
```

:INPut[:STATe]?

Syntax :<Handle>:SGENeral:**PDATa(*):INPut**[:STATe]? <NFORce| FORCe>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**[:STATe]? <NFORce| FORCe>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**[:STATe]? <NFORce| FORCe>

Description Returns the current status of the specified analyzer input connector.

Parameters **<NFORce|FORCe>** Defines how the query is executed:

- NFORce

  Default value. Only the firmware is queried for the current state.

- FORCe

  When FORCe is entered, the firmware checks the current condition in the hardware. This is required to make sure that the input is really switched on, or to check if the protection circuit within the input has turned off the channel to protect it from damage.

NOTE The `FORCe` call is slower than `NFORce`.

www.valuetronics.com

Example

```
:_test:SGEN:PDAT1:INPUT?
:_test:SGEN1:PDAT1:TERM1:INPUT?
:_test:CGROUP1:MODULE2:CONNECTOR4:INPUT?
```

might return:

```
ON
```

:INPut:POLarity

Syntax

:<Handle>:SGENeral:**PDATa(*)**:INPut:POLarity <NORMal | INVerted>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:INPut:POLarity <NORMal | INVerted>

:<Handle>:CGRoup(*):MODule(*):**CONNector(*)**:INPut:POLarity <NORMal | INVerted>

Parameters

<NORMal | INVerted> Sets the input polarity of the specified *data* port/terminal/connector to either normal or inverted.

This command is only supported for E4835A frontends.

Example

```
:_test:SGEN:PDAT1:INP:POL INV
:_test:SGEN:PDAT1:TERM1:INP:POL INV
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:POLARITY NORM
```

:INPut:POLarity?

Syntax

:<Handle>:SGENeral:**PDATa(*)**:INPut:POLarity?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:INPut:POLarity?

:<Handle>:CGRoup(*):MODule(*):**CONNector(*)**:INPut:POLarity?

Return Value

Returns the input polarity of the specified port/terminal/connector. The return value is either NORMal or INVerted.

This command is only supported for E4835A frontends.

Example

```
:_test:SGEN:PDAT1:INP:POL?
:_test:SGEN:PDAT1:TERM1:INP:POL?
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:POLARITY?
```

might return:

```
NORM
```


www.valuetronics.com

:INPut:TYPE

Syntax

:<Handle>:SGENeral:**PDATa**(*):INPut:TYPE <BALanced | NORMal | INVerted>

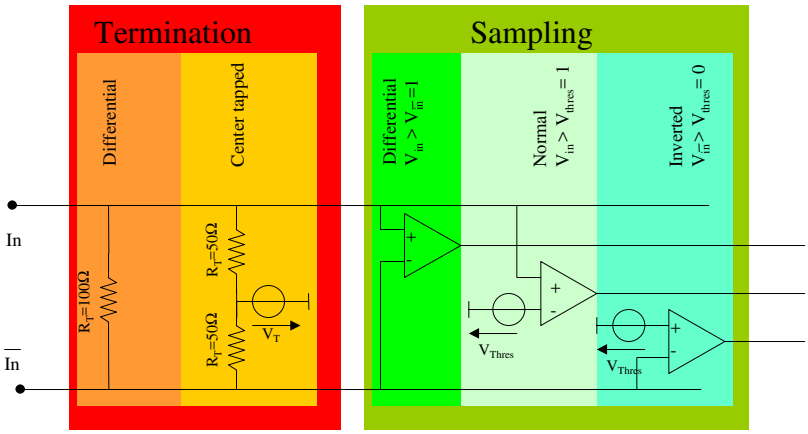
:<Handle>:SGENeral:**PDATa**(*)**:TERMinal**(*):INPut:TYPE <BALanced | NORMal | INVerted>

:<Handle>:CGRoup(*):MODule(*):**CONNector**(*):INPut:TYPE <BALanced | NORMal | INVerted>

Parameters

<BALanced | NORMal | INVerted> Selects how the E4835A, E4863A and E4865A differential input frontends are sampled.

- BALanced selects differential operation (this is the reset value).
- NORMal compares the normal input against a threshold value.
- INVerted compares the complement input against a threshold value.



In NORMal and INVerted operation (*":INPut:IMPedance[:INTernal]" on page 243* and *":INPut:TVOLtage" on page 241*) specify the termination of the input.

In BALanced operation the termination is not used, but is checked against absolute limits. *":INPut:SERial" on page 244* is also ignored in BALanced operation, but is checked against absolute limits.

Example

```
:_test:SGEN:PDAT1:INP:TYPE BAL

:_test:SGEN:PDAT1:TERM1:INP:TYPE INV

:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:TYPE NORM
```


www.valuetronics.com

:INPut:TYPE?

Syntax :<Handle>:SGENeral:**PDATa(*)**:INPut:TYPE?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:INPut:TYPE?

:<Handle>:CGRoup(*):MODule(*):**CONNector(*)**:INPut:TYPE?

Return Value Returns the operation mode of the E4835A, E4863A or E4865A differential input frontend.

Example

```
:_test:SGEN:PDAT1:INP:TYPE?
:_test:SGEN:PDAT1:TERM1:INP:TYPE?
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:TYPE?
```

might return:

```
BAL
```

:INPut:MODE

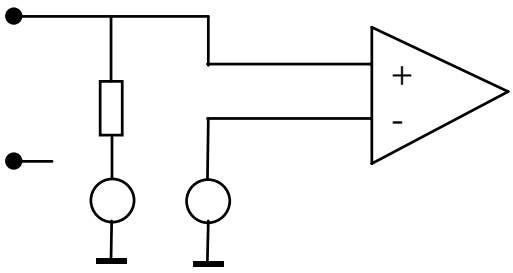
Syntax :<Handle>:SGENeral:**PDATa(*)**:INPut:MODE <DIFF | SNOR | SCOM>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:INPut:MODE <DIFF | SNOR | SCOM>

:<Handle>:CGRoup(*):MODule(*):**CONNector(*)**:INPut:MODE <DIFF | SNOR | SCOM>

Parameters **<DIFF | SNOR | SCOM>** Selects between single-ended operation and differential operation modes (if supported by the frontend).

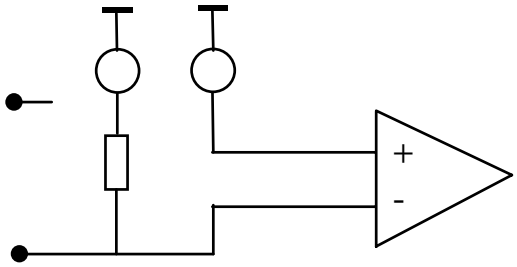
- DIFF selects differential frontend operation. Termination is done according to :INPut:TCONfig, comparator operation is selected according to :INPut:MODE (this is the reset value for E4835A, E4863A, and E4865A frontends).
- SNOR configures the frontend to single ended operation using the normal input as shown in the following picture:



Note that :INPut:TCONfig and :INPut:MODE is ignored in this case.

www.valuetronics.com

- SCOM configures the frontend to single-ended operation using the complement input according to the following picture:



Note that :INPut:TCONfig and :INPut:MODE is ignored in this case.

Example

```
:_test:SGEN:PDAT1:INP:MODE DIFF
:_test:SGEN:PDAT1:TERM1:INP:MODE SNOR
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:MODE SCOM
```

:INPut:MODE?

Syntax

:<Handle>:SGENeral:**PDATa(*)**:INPut:MODE?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:INPut:MODE?

:<Handle>:CGRoup(*):MODule(*):**CONNector(*)**:INPut:MODE?

Return Value

Returns the operation mode of an input frontend.

Example

```
:_test:SGEN:PDAT1:INP:MODE?
:_test:SGEN:PDAT1:TERM1:INP:MODE?
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:MODE?
```

might return:

```
SNOR
```


www.valuetronics.com

:INPut:TVOLtage

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:TVOLtage <TerminationVoltage>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:TVOLtage <TerminationVoltage>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:TVOLtage <TerminationVoltage>

Parameters

<TerminationVoltage> Termination voltage (<NRf>).

Specifies the termination voltage of an analyzer input connector. The termination voltage range of the available analyzer frontends is –2.1 V to +3.1 V.

Example

```
:_test:SGEN:PDAT1:INP:TVOL -2
:_test:SGEN1:PDAT1:TERM1:INP:TVOL -2
:_test:CGROUP1:MODULE2:CONNECTOR4:INP:TVOL -2
```

:INPut:TVOLtage?

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:TVOLtage?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:TVOLtage?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:TVOLtage?

Return Value

Returns the current termination voltage of the specified analyzer input connector.

Example

```
:_test:SGEN:PDAT1:INP:TVOL?
:_test:SGEN1:PDAT1:TERM1:INP:TVOL?
:_test:CGROUP1:MODULE2:CONNECTOR4:INP:TVOL?
```

might return:

```
-2.000000E+000
```


www.valuetronics.com

:INPut:THReshold

Syntax :<Handle>:SGENeral:**PDATa(*):INPut**:THReshold <Threshold>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:THReshold <Threshold>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:THReshold <Threshold>

Parameters **<Threshold>** Threshold value in volts (<NRf>).

Specifies the threshold voltage of an analyzer input connector. The threshold voltage range is:

- for E4835A differential analyzer frontend: –2.1 V to +4.1 V,
- for E4863A and E4865A differential analyzer frontend: –2 V to +3 V,
- for all other analyzer frontends: –2.1 V to +5.1 V.

Example

```
:_test:SGEN:PDAT1:INPut:THR 1
:_test:SGEN1:PDAT1:TERM1:INPut:THR 1
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:THR 1
```

:INPut:THReshold?

Syntax :<Handle>:SGENeral:**PDATa(*):INPut**:THReshold?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:THReshold?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:THReshold?

Return Value Returns the current threshold voltage of the specified analyzer input connector.

Example

```
:_test:SGEN:PDAT1:INPut:THR?
:_test:SGEN1:PDAT1:TERM1:INPut:THR?
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:THR?
```

might return.

```
1.000000E+000
```


www.valuetronics.com

:INPut:IMPedance[:INTernal]

Syntax :<Handle>:SGENeral:**PDATa(*):INPut**:IMPedance[:INTernal] <Impedance>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:IMPedance[:INTernal] <Impedance>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:IMPedance[:INTernal] <Impedance>

Parameters **<Impedance>** Internal termination impedance value.

Specifies the internal termination impedance of the analyzer input connector. To set 10 kOhm impedance, use -1 in the command. For 50 Ohm impedance, use 50 in the command.

Example

```
:_test:SGEN:PDAT1:INPut:IMP -1
:_test:SGEN1:PDAT1:TERM1:INPut:IMP -1
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:IMP -1
```

:INPut:IMPedance[:INTernal]?

Syntax :<Handle>:SGENeral:**PDATa(*):INPut**:IMPedance[:INTernal]?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:IMPedance[:INTernal]?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:IMPedance[:INTernal]?

Return Value Returns the current internal impedance of the specified analyzer input connector. A negative value relates to the HiZ internal impedance.

Example

```
:_test:SGEN:PDAT1:INPut:IMP?
:_test:SGEN1:PDAT1:TERM1:INPut:IMP?
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:IMP?
```

might return:

```
-1.000000E+000
```


www.valuetronics.com

:INPut:SERial

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:SERial <Impedance>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:SERial <Impedance>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:SERial <Impedance>

Parameters

<Impedance> Serial impedance value.

Specifies a serial impedance on the specified analyzer input connector. The reset value is 0.

For electronic devices that are not able to drive a 50-Ohm input, an additional serial resistor is used. The specified value is taken into account to calculate the corresponding threshold.

Example

```
:_test:SGEN:PDAT1:INPut:SER 50
:_test:SGEN1:PDAT1:TERM1:INPut:SER 50
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:SER 50
```

:INPut:SERial?

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:SERial?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:SERial?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:SERial?

Return Value

Returns the current serial impedance used for the specified analyzer input connector.

Example

```
:_test:SGEN:PDAT1:INPut:SER?
:_test:SGEN1:PDAT1:TERM1:INPut:SER?
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:SER?
```

might return:

```
5.000000E+001
```


www.valuetronics.com

:INPut:DELay

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:DELay <Delay>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:DELay <Delay>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:DELay <Delay>

Parameters

<Delay> Specifies the delay of the sampling edge (<NRf>). It can be specified in seconds or in cycles (take into account the MUX-factor or in other words the “oversampling factor”).

If this command is used, the value will be interpreted as a timing value only. Therefore, :INPut:DELay:CYCLes is set to zero.

Otherwise, if the delay is specified by :INPut:DELay:CYCLes and : INPut:DELay:TIME, the corresponding delay will be calculated. The number of cycles is converted into seconds and added to the specified :INPut:DELay:TIME value.

Example

```
:_test:SGEN:PDAT1:INPut:DEL 50e-9
:_test:SGEN1:PDAT1:TERM1:INPut:DEL 50e-9
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:DEL 50e-9
```

:INPut:DELay?

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:DELay?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:DELay?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:DELay?

Return Value

Returns the current delay of the sampling edge for the specified analyzer input connector.

Example

```
:_test:SGEN:PDAT1:INPut:DEL?
:_test:SGEN1:PDAT1:TERM1:INPut:DEL?
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:DEL?
```

might return:

```
5.000000E-008
```


www.valuetronics.com

:INPut:DELay:CYCLes

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:DELay:CYCLes <Cycles>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:DELay:CYCLes <Cycles>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:DELay:CYCLes <Cycles>

Parameters

<Cycles> 1 cycle corresponds to the actual period/frequency valid for the specified analyzer input connector.

If the delay is specified by DELay:CYCLes and :INPut:DELay:TIME, the corresponding delay will be calculated. The number of cycles is converted into seconds and added to the specified :INPut:DELay:TIME value.

If :INPut:DELay is used, the value will be interpreted as a timing value only. Therefore, :INPut:DELay:CYCLess is set to zero.

Example

```
:_test:SGEN:PDAT1:INPut:DELAY:CYCL 0.5
:_test:SGEN1:PDAT1:TERM1:INPut:DELAY:CYCL 0.5
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:DELAY:CYCL 0.5
```

:INPut:DELay:CYCLes?

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:DELay:CYCLes?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:DELay:CYCLes?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:DELay:CYCLes?

Return Value

Returns the current cycle value of the sampling edge for the specified analyzer input connector.

Example

```
:_test:SGEN:PDAT1:INPut:DELAY:CYCL?
:_test:SGEN1:PDAT1:TERM1:INPut:DELAY:CYCL?
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:DELAY:CYCL?
```

might return:

```
5.000000E-001
```


www.valuetronics.com

:INPut:DELay:TIME

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:DELay:TIME <Time>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:DELay:TIME <Time>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:DELay:TIME <Time>

Parameters

<Time> Delay in seconds.

If the delay is specified by :INPut:DELay:CYCLes and DELay:TIME, the corresponding delay will be calculated. The number of cycles is converted into seconds and added to the specified DELay:TIME value.

If :INPut:DELay is used, the value will be interpreted as a timing value only. Therefore, :INPut:DELay:CYCLes is set to zero.

Example

```
:_test:SGEN:PDAT1:INPut:DELAY:TIME 5e-9
:_test:SGEN1:PDAT1:TERM1:INPut:DELAY:TIME 5e-9
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:DELAY:TIME 5e-9
```

:INPut:DELay:TIME?

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:DELay:TIME?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:DELay:TIME?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:DELay:TIME?

Return Value

Returns the current additional delay value of the sampling edge for the specified analyzer input connector. When there is no CYCLe value specified, it corresponds to the DELay value.

Example

```
:_test:SGEN:PDAT1:INPut:DELAY:TIME?
:_test:SGEN1:PDAT1:TERM1:INPut:DELAY:TIME?
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:DELAY:TIME?
```

might return:

```
5.000000E-009
```


www.valuetronics.com

:INPut:DELay:ACTual?

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:DELay:ACTual?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:DELay:ACTual?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:DELay:ACTual?

Return Value

The Agilent 81250 has the following methods to get the expected data and the incoming data aligned to ensure proper real-time compare:

- Synchronous measurement

  The exact sample-point delay is well-known in advance and is programmed as the sample delay. In this case, the actual delay query returns:

  ACT DELAY= Delay(Time) + Delay(Cycles) + Delay(Sweep)

- Auto delay adjust

  The exact sample delay is not known in advance, but a certain delay range the correct sample point will be inside. The actual delay query returns:

  ACT DELAY= Delay(Time) + Delay(Cycles) + Delay(Sweep) + Delay(B-SYNC)

- Auto Bit Synchronization

  Not even a delay range is known within which the incoming data will start. The data can come at any time after the system is started. The actual delay query returns:

  ACT DELAY= Delay(Time) + Delay(Cycles) + Delay(Sweep) + Delay(B-SYNC)

  Note that the actual delay may exceed the returned value by a multiple the period.

Delay(Time), Delay(Cycles), and Delay(Sweep) are shown in property window of the user interface. To query the individual values, use *":INPut:DELay:TIME?" on page 247*, *":INPut:DELay:CYCLes?" on page 246* and *":INPut:DELay:SWEep?" on page 249*.

Delay(B-SYNC) is the bit sync delay. It is measured by every analyzer involved and is reported to the firmware and added to the actual delay after successful synchronization.

Example

```
:_test:SGEN:PDAT1:INPut:DELAY:ACT?
:_test:SGEN1:PDAT1:TERM1:INPut:DELAY:ACT?
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:DELAY:ACT?
```

might return:

```
1.25E-9
```


www.valuetronics.com

:INPut:DELay:SWEep

Syntax :<Handle>:SGENeral:**PDATa(*):INPut**:DELay:SWEep <Sweep>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:DELay:SWEep <Sweep>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:DELay:SWEep <Sweep>

Parameters **<Sweep>** Sets the delay sweep value in the range of –1.0 to +1.0.

The delay sweep allows to shift the analyzer sampling point by up to ±1 clock periods while a test is running. This makes it possible to measure the eye opening of a superimposed signal without interrupting the test.

The delay sweep is available on E4863A and E4865A frontends, and on E4835A frontends with a frequency higher than 20.8 MHz and highest possible segment resolution.

Example

```
:_test:SGEN:PDAT1:INPut:DELAY:SWE -0.4
:_test:SGEN:PDAT1:TERM1:INPut:DELAY:SWE 0.1
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:DELAY:SWE 0.75
```

:INPut:DELay:SWEep?

Syntax :<Handle>:SGENeral:**PDATa(*):INPut**:DELay:SWEep?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:DELay:SWEep?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:DELay:SWEep?

Return Value Returns the current delay sweep value in the range of –1.0 to +1.0.

Example

```
:_test:SGEN:PDAT1:INPut:DELAY:SWE?
:_test:SGEN:PDAT1:TERM1:INPut:DELAY:SWE?
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:DELAY:SWE?
```

might return:

```
-0.4
```


www.valuetronics.com

:INPut:TCONfig

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:TCONfig <VOLTage | DIFFerential>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:TCONfig <VOLTage | DIFFerential>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:TCONfig <VOLTage | DIFFerential>

Parameters

<VOLTage | DIFFerential> Selects the termination model for input frontends:

- VOLTage selects termination via resistor against a termination voltage.

  If VOLTage is selected, the termination resistor and termination voltage can be set with *":INPut:IMPedance[:INTernal]" on page 243* and *":INPut:TVOLtage" on page 241*.

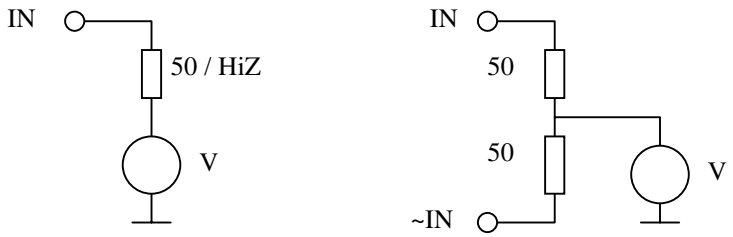
- DIFFerential selects termination via a resistor against the complement input (DIFFerential is only supported for E4835A, E4863A and E4865A differential input frontends).

  If DIFFerential is selected, the differential termination resistor can be set with *":INPut:DIMPedance" on page 251*.

Description

For input frontends two possible termination models are available:

- Via a resistor against a termination voltage. The resistor might be high impedance or 50 Ohm. In the following picture this is shown for single ended and differential inputs.



Example

```
:_test:SGEN:PDAT1:INP:TCON DIFF
:_test:SGEN:PDAT1:TERM1:INP:TCON DIFF
:_test:MOD1:CONN2:INP:TCON VOLT
```


www.valuetronics.com

:INPut:TCONfig?

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:TCONfig?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:TCONfig?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:TCONfig?

Return Value

Returns the termination model for input frontends (see *":INPut:TCONfig" on page 250*).

Example

```
:_test:SGEN:PDAT1:TERM1:INP:TCON?
:_test:SGEN1:PDAT1:TERM1:INPut:TCONFIG?
:_test:CGROUP1:MODULE2:CONNECTOR4:INP:TCON?
```

might return:

```
DIFF
```

:INPut:DIMPedance

Syntax

:<Handle>:SGENeral:**PDATa(*):INPut**:DIMPedance[:INTernal] <Impedance>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:DIMPedance[:INTernal] <Impedance>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:DIMPedance[:INTernal] <Impedance>

Parameters

<Impedance> Sets the impedance between the IN and \IN\ inputs of an E4835A, E4863A and E4865A frontend when the differential termination model is selected (see *":INPut:TCONfig" on page 250*).

The only possible value is 100 Ohm.

Example

```
:_test:SGEN:PDAT:INP:DIMP 100
:_test:SGEN:PDAT1:TERM1:INP:DIMP 100
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:DIMP 100
```


www.valuetronics.com

:INPut:DIMPedance?

Syntax :<Handle>:SGENeral:**PDATa(*):INPut**:DIMPedance[:INTernal]?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):INPut**:DIMPedance[:INTernal]?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):INPut**:DIMPedance[:INTernal]?

Return Value Returns the differential termination resistor (see *":INPut:TCONfig" on page 250*).

The only possible value is 100 Ohm.

Example

```
:_test:SGEN:PDAT:INP:DIMP?
:_test:SGEN:PDAT1:TERM1:INP:DIMP?
:_test:CGROUP1:MODULE2:CONNECTOR4:INPut:DIMP?
```

might return:

```
100
```

:INPut:DISConnect:MODE?

Syntax :<Handle>:CGRoup(*):MODule(*):CONNector(*):INPut:DISConnect:MODE?

:<Handle>:SGENeral:PDATa(*):INPut:DISConnect:MODE?

:<Handle>:SGENeral:PDATa(*):TERMinal:INPut:DISConnect:MODE?

Return Value Returns how the specified connector is to be disconnected when the `SGENeral:GLOBal:CONNect OFF` command is executed. Can be either RELay (disconnected by switching off the corresponding relay) or VOLtage (disconnected by reducing the voltage to 0).

Example

```
:_test:CGR1:MOD1:CONN1:INP:DISC:MODE?
```

might return:

```
REL
```


www.valuetronics.com

:INPut:DISConnect:MODE

Syntax

:<Handle>:CGRoup(*):MODule(*):CONNector(*):INPut:DISConnect:MODE RELay| VOLTage

:<Handle>:SGENeral:PDATa(*):INPut:DISConnect:MODE RELay| VOLTage

:<Handle>:SGENeral:PDATa(*):TERMinal:INPut:DISConnect:MODE RELay| VOLTage

Parameters

RELay Disconnects the specified connector by switching the corresponding relay when the `SGENeral:GLOBal:CONNect OFF` command is executed.

VOLTage Disconnects the specified connector by setting the voltage to 0 when the `SGENeral:GLOBal:CONNect OFF` command is executed.

Example

```
:_test:CGR1:MOD1:CONN1:INP:DISC:MODE REL
```

Optical Input Commands

The optical input commands are used for ParBERT optical analyzers, such as the E4811A. The following commands are available:

- STATe

  Sets/returns the state of the O/E converter.

- POWer:UNIT

  Sets/returns the unit used for the 0/1 decision threshold.

- POWer:THReshold

  Sets/returns the 0/1 decision threshold.

- MEASure

  Measures the average signal power and sets this as the 0/1 decision threshold.

- DLCalibration

  Performs Dark Level Calibration.

See also *"Optical Output Commands" on page 270*.

www.valuetronics.com

:INPut:OPTic[:STATe][:VALue]

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):INPut:OPTic:[STATe]:[VALue] <OFF| ON>

:<Handle>:SGENeral:PDATa(*):INPut:OPTic:[STATe]:[VALue] <OFF| ON>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):INPut:OPTic:[STATe]:[VALue] <OFF| ON>

Parameters

<OFF|ON> Sets the state of the optical channel(s) when in optical mode. The electrical analyzer is automatically enabled/disabled.

Examples

```
:_test:CGR1:MOD1:CONN1:INP:OPT:STAT:VAL OFF
```

```
:_test:CGR1:MOD1:CONN1:INP:OPT OFF
```

:INPut:OPTic[:STATe][:VALue]?

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):INPut:OPTic:[STATe]:[VALue]? <NFORce| FORCe>

:<Handle>:SGENeral:PDATa(*):INPut:OPTic:[STATe]:[VALue]? <NFORce| FORCe>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):INPut:OPTic:[STATe]:[VALue]? <NFORce| FORCe>

Return Value

The state of the optical channel(s) when in optical mode.

Parameters

<NFORce|FORCe> Defines how the query is executed:

- NFORce

  Default value. Only the firmware is queried for the current state.

- FORCe

  When FORCe is entered, the firmware checks the current condition in the hardware. This is required to make sure that the optical channel's electrical interface is really switched on.

NOTE The `FORCe` query is slower than `NFORce`.

Examples

```
:_test:CGR1:MOD1:CONN1:INP:OPT:STAT:VAL? FORC
```

```
:_test:CGR1:MOD1:CONN1:INP:OPT?
```

might return:

```
ON
```


www.valuetronics.com

:INPut:OPTic:POWer:UNIT[:VALue]

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):INPut:OPTic:POWer:UNIT[:VALue] <WATT| DBM>

:<Handle>:SGENeral:PDATa(*):INPut:OPTic:POWer:UNIT[:VALue] <WATT| DBM>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):INPut:OPTic:POWer:UNIT[:VALue] <WATT| DBM>

Parameters

<WATT|DBM> Selects the unit for the 0/1 decision threshold of the selected optical input port, terminal, or connector.

dBm is a unitless logarithmic alternative to Watt. The following calculations are used for transformations:

- $Power_{dBm} = 10*log(Power_{Watt}/0.001\ W)$
- $Power_{dBm} = 10*log(Power_{Watt}/1\ mW)$
- $Power_{Watt} = 1\ mW * 10\text{^}(Power_{dBm}/10)$

Examples

```
:_test:CGR1:MOD1:CONN1:INP:OPT:POW:UNIT:VAL WATT
```

```
:_test:CGR1:MOD1:CONN1:INP:OPT:POW:UNIT WATT
```

:INPut:OPTic:POWer:UNIT[:VALue]?

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):INPut:OPTic:POWer:UNIT[:VALue]?

:<Handle>:SGENeral:PDATa(*):INPut:OPTic:POWer:UNIT[:VALue]?

:<Handle>:SGENeral:PDATa(*):TERMinal(*):INPut:OPTic:POWer:UNIT[:VALue]?

Return Value

Used power unit for setting and querying the 0/1 decision threshold.

Examples

```
:_test:CGR1:MOD1:CONN1:INP:OPT:POW:UNIT:VAL?
```

```
:_test:CGR1:MOD1:CONN1:INP:OPT:POW:UNIT?
```

might return:

```
WATT
```


www.valuetronics.com

:INPut:OPTic:POWer:THReshold[:VALue]

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):INPut:OPTic:POWer:THReshold[:VALue] <DT>

:<Handle>:SGENeral:PDATa(*):INPut:OPTic:POWer:THReshold[:VALue] <DT>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):INPut:OPTic:POWer:THReshold[:VALue] <DT>

Parameters

<DT> 0/1 **D**ecision **T**hreshold for the comparator built into the O/E converter; signals above this value are treated as 1s, signals below this value are treated as 0s. The current unit can be queried with *":INPut:OPTic:POWer:UNIT[:VALue]?" on page 255*.

:INPut:OPTic:POWer:THReshold[:VALue]?

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):INPut:OPTic:POWer:THReshold[:VALue]?

:<Handle>:SGENeral:PDATa(*):INPut:OPTic:POWer:THReshold[:VALue]?

:<Handle>:SGENeral:PDATa(*):TERMinal(*):INPut:OPTic:POWer:THReshold[:VALue]?

Return Value

0/1 decision threshold of the connector, port or terminal. The unit of the returned value can be queried with *":INPut:OPTic:POWer:UNIT[:VALue]?" on page 255*

:INPut:OPTic:MEASure

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):INPut:OPTic:MEASure

:<Handle>:SGENeral:PDATa(*):INPut:OPTic:MEASure

:<Handle>:SGENeral:PDATa(*):TERMinal(*):INPut:OPTic:MEASure

Measures the average signal power and sets this as the 0/1 decision threshold.

Examples

```
:_test:CGR1:MOD1:CONN1:INP:OPT:MEAS
:_test:CGR1:MOD1:CONN1:INP:OPT:MEAS
```


www.valuetronics.com

:INPut:OPTic:DLCalibration

Syntax :<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):INPut:OPTic:DLCalibration

:<Handle>:SGENeral:PDATa(*):INPut:OPTic:DLCalibration

:<Handle>:SGENeral:PDATa(*):TERMinal(*):INPut:OPTic:DLCalibration

Performs **D**ark **L**evel **C**alibration on the port, terminal, or connector. For more information on Dark Level Calibration, see *How to Set Optical Analyzer Levels* in the *ParBERT System User Guide*.

It is recommended to perform Dark Level Calibration after the ParBERT has completely warmed up (~30 minutes), or after the temperature has changed by more than 10 °C (18 °F).

www.valuetronics.com

Output Parameter Commands

The commands for specifying the output parameters are available on port, terminal and connector level. The parameters can be specified separately for pulse and data ports.

For optical modules, see *"Optical Output Commands" on page 270* for the commands.

:OUTPut[:STATe]

Syntax

:<Handle>:SGENeral:**PDATa(*)**:OUTPut[:STATe] <ON | OFF>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:OUTPut[:STATe] <ON | OFF>

:<Handle>:SGENeral:**PPULse(*)**:OUTPut[:STATe] <ON | OFF>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*)**:OUTPut:[STATe] <ON | OFF>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**:OUTPut:[STATe] <ON | OFF>

Parameters

<ON | OFF> Switch the normal connector ON or OFF.

Controls whether the output connectors are open (ON) or closed (OFF).

Example

```
:_test:SGEN:PDAT1:OUTP ON
:_test:SGEN:PDAT1:TERM1:OUTP:STAT ON
:_test:SGEN:PPUL1:OUTP ON
:_test:SGEN:PPUL1:TERM1:OUTP:STAT OFF
:_test:MOD2:CONN1:OUTP OFF
```


www.valuetronics.com

:OUTPut[:STATe]?

Syntax

:<Handle>:SGENeral:**PDATa(*):OUTPut**[:STATe]? <NFORce| FORCe>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):OUTPut:**[STATe]? <NFORce| FORCe>

:<Handle>:SGENeral:**PPULse(*):OUTPut**[:STATe]? <NFORce| FORCe>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):OUTPut:**[STATe]? <NFORce| FORCe>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):OUTPut:**[STATe]? <NFORce| FORCe>

Parameters

<NFORce|FORCe> Defines how the query is executed:

- NFORce

  Default value. Only the firmware is queried for the current state.

- FORCe

  When FORCe is entered, the firmware checks the current condition in the hardware. This is required to make sure that the output is really switched on, or to check if the protection circuit within the output has turned off the channel to protect it from damage.

NOTE The `FORCe` call is slower than `NFORce`.

Return Value

Returns the current connection state for the specified output connectors.

```
:_test:SGEN:PDAT1:OUTP? NFOR
:_test:SGEN:PDAT1:TERM1:OUTP:STAT? FORC
:_test:SGEN:PPUL1:OUTP? NFOR
:_test:SGEN:PPUL1:TERM1:OUTP:STAT? FORC
:_test:MOD2:CONN1:OUTP?
```

might return:

```
ON
```


www.valuetronics.com

:OUTPut:POLarity

Syntax

:<Handle>:SGENeral:**PDATa**(*):OUTPut:POLarity <NORMal | INVerted>

:<Handle>:SGENeral:**PDATa**(*)**:TERMinal**(*):OUTPut:POLarity <NORMal | INVerted>

:<Handle>:SGENeral:**PPULse**(*):OUTPut:POLarity <NORMal | INVerted>

:<Handle>:SGENeral:**PPULse**(*)**:TERMinal**(*):OUTPut:POLarity <NORMal | INVerted>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector**(*):OUTPut:POLarity <NORMal | INVerted>

Parameters

<NORMal | INVerted> Sets the output polarity to either normal or inverted (NORMal is the reset value).

This command is only supported for E4838A frontends.

Example

```
:_test:SGEN:PDAT1:OUTP:POL NORM
:_test:SGEN:PDAT1:TERM1:OUTP:POL INV
:_test:SGEN:PPUL1:OUTP:POL NORM
:_test:SGEN:PPUL1:TERM1:OUTP:POL NORM
:_test:CGR1:MOD2:CONN4:OUTP:POL INV
```

:OUTPut:POLarity?

Syntax

:<Handle>:SGENeral:**PDATa**(*):OUTPut:POLarity?

:<Handle>:SGENeral:**PDATa**(*)**:TERMinal**(*):OUTPut:POLarity?

:<Handle>:SGENeral:**PPULse**(*):OUTPut:POLarity?

:<Handle>:SGENeral:**PPULse**(*)**:TERMinal**(*):OUTPut:POLarity?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*)**:OUTPut:POLarity?

Return Value

Returns the current output polarity (NORMal or INVerted).

This command is only supported for E4838A frontends.

Example

```
:_test:SGEN:PDAT1:OUTP:POL?
:_test:SGEN:PDAT1:TERM1:OUTP:POL?
:_test:SGEN:PPUL1:OUTP:POL?
:_test:SGEN:PPUL1:TERM1:OUTP:POL?
:_test:CGR1:MOD2:CONN4:OUTP:POL?
```

might return:

```
NORM
```


www.valuetronics.com

:OUTPut:CSTate

Syntax

:<Handle>:SGENeral:**PDATa(*):OUTPut:**CSTate <ON | OFF>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):OUTPut:**CSTate <ON | OFF>

:<Handle>:SGENeral:**PPULse(*):OUTPut:**CSTate <ON | OFF>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):OUTPut:**CSTate <ON | OFF>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):OUTPut:**CSTate <ON | OFF>

Parameters

<ON | OFF> Switches the complement connector ON or OFF.

Controls whether the complement output connectors are open (ON) or closed (OFF).

Example

```
:_test:SGEN:PDAT1:OUTP:CST ON
:_test:SGEN:PDAT1:TERM1:OUTP:CSTAT OFF
:_test:SGEN:PPUL1:OUTP:CST ON
:_test:SGEN:PPUL1:TERM1:OUTP:CSTAT OFF
:_test:CGR1:MOD2:CONN2:OUTP:CSTAT ON
```

:OUTPut:CSTate?

Syntax

:<Handle>:SGENeral:**PDATa(*):OUTPut:**CSTate?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):OUTPut:**CSTate?

:<Handle>:SGENeral:**PPULse(*):OUTPut:**CSTate?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):OUTPut:**CSTate?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):OUTPut:**CSTate?

Return Value

Returns the current connection state (enabled or disabled) for the complement connectors of the specified port, terminal, or connector.

Example

```
:_test:SGEN:PDAT1:OUTP:CST?
:_test:SGEN:PDAT1:TERM1:OUTP:CSTAT?
:_test:SGEN:PPUL1:OUTP:CST?
:_test:SGEN:PPUL1:TERM1:OUTP:CSTAT?
:_test:CGR1:MOD2:CONN2:OUTP:CSTAT?
```

might return:

```
OFF
```


www.valuetronics.com

:OUTPut:TVOLtage

Syntax

:<Handle>:SGENeral:**PDATa(*):OUTPut:**TVOLtage <TerminationVoltage>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):OUTPut:**TVOLtage <TerminationVoltage>

:<Handle>:SGENeral:**PPULse(*):OUTPut:**TVOLtage <TerminationVoltage>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):OUTPut:**TVOLtage <TerminationVoltage>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):OUTPut:**TVOLtage <TerminationVoltage>

Parameters

<TerminationVoltage> Termination voltage (<NRf>).

Sets the external termination voltage, default is 0 volts = ground.

Valid range is –2.0 to +3.0 volts.

Example

```
:_test:SGEN:PDAT1:OUTP:TVOLT -1
:_test:SGEN:PDAT1:TERM1:OUTP:TVOL 2.0
:_test:SGEN:PPUL1:OUTP:TVOLT -1
:_test:SGEN:PPUL1:TERM1:OUTP:TVOL 2.0
:_test:CGROUP1:MODULE2:CONNECTOR4:OUTPUT:TVOLTAGE -1
```

:OUTPut:TVOLtage?

Syntax

:<Handle>:SGENeral:**PDATa(*):OUTPut:**TVOLtage?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):OUTPut:**TVOLtage?

:<Handle>:SGENeral:**PPULse(*):OUTPut:**TVOLtage?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):OUTPut:**TVOLtage?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):OUTPut:**TVOLtage?

Return Value

Returns the currently set value for the external Termination VOLtage.

Example

```
:_test:SGEN:PDAT1:OUTP:TVOLT?
:_test:SGEN:PDAT1:TERM1:OUTP:TVOL?
:_test:SGEN:PPUL1:OUTP:TVOLT?
:_test:SGEN:PPUL1:TERM1:OUTP:TVOL?
:_test:CGROUP1:MODULE2:CONNECTOR4:OUTPUT:TVOLTAGE?
```

might return:

```
-1
```


www.valuetronics.com

:OUTPut:IM Pedance:EXTernal

Syntax

:<Handle>:SGENeral:**PDATa(*):OUTPut:**IMPedance:EXTernal <LoadImpedance>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):OUTPut:**IMPedance:EXTernal <LoadImpedance>

:<Handle>:SGENeral:**PPULse(*):OUTPut:**IMPedance:EXTernal <LoadImpedance>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):OUTPut:**IMPedance:EXTernal <LoadImpedance>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):OUTPut:**IMPedance: EXTernal <LoadImpedance>

Parameters

<LoadImpedance> Impedance of the DUT (load) in Ohms (<NRf>). Sets the external termination impedance, this is the real load impedance of the DUT.

Any negative value is interpreted as “into open”.

Example

```
:_test:SGEN:PDAT1:OUTP:IMP:EXT 500
:_test:SGEN:PDAT1:TERM1:OUTP:IMP:EXT 70
:_test:SGEN:PPUL1:OUTP:IMP:EXT 500
:_test:SGEN:PPUL1:TERM1:OUTP:IMP:EXT 70
:_test:MOD2:CONN3:OUTP:IMP:EXT 500
```

:OUTPut:IM Pedance:EXTernal?

Syntax

:<Handle>:SGENeral:**PDATa(*):OUTPut:**IMPedance:EXTernal?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):OUTPut:**IMPedance:EXTernal?

:<Handle>:SGENeral:**PPULse(*):OUTPut:**IMPedance:EXTernal?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):OUTPut:**IMPedance:EXTernal?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):OUTPut:**IMPedance: EXTernal?

Return Value

Returns the currently set value for the termination (load) impedance.

Example

```
:_test:SGEN:PDAT1:OUTP:IMP:EXT?
:_test:SGEN:PDAT1:TERM1:OUTP:IMP:EXT?
:_test:SGEN:PPUL1:OUTP:IMP:EXT?
:_test:MOD2:CONN3:OUTP:IMP:EXT?
```

might return:

```
500
```


www.valuetronics.com

:OUTPut:TCONfig

Syntax

:<Handle>:SGENeral:**PDATa(*):OUTPut:**TCONfig <VOLTage | DIFFerential>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):OUTPut:**TCONfig <VOLTage | DIFFerential>

:<Handle>:SGENeral:**PPULse(*):OUTPut:**TCONfig <VOLTage | DIFFerential>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):OUTPut:**TCONfig <VOLTage | DIFFerential>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):OUTPut:**TCONfig <VOLTage | DIFFerential>

Parameters

<VOLTage | DIFFerential> Selects the termination model for output frontends:

- VOLTage selects termination via resistor against a termination voltage (this is the reset value).

  If VOLTage is selected, the termination resistor and termination voltage can be set with *":OUTPut:IMPedance:EXTernal" on page 263* and *":OUTPut:TVOLtage" on page 262*.

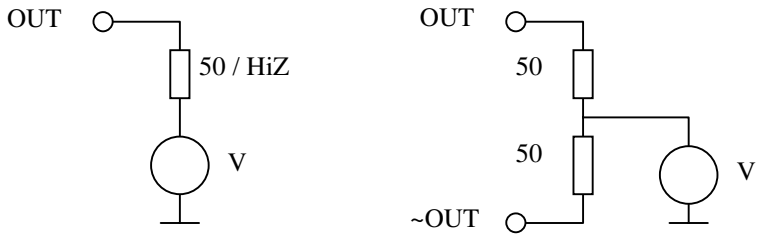
- DIFFerential selects termination via a resistor against the complement output. DIFFerential is only available on the differential output frontends E4838A, E4843A, E4862A and E4864A.

  If DIFFerential is selected, the differential termination resistor can be set with *":OUTPut:DIMPedance:EXTernal" on page 266*.

Description

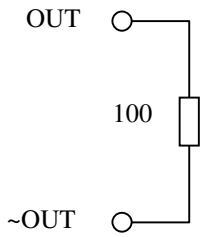
For output frontends it is necessary to specify how the signal will be terminated. There are two possible termination models available:

- Via a resistor against a termination voltage; the resistor might be high impedance. Optimum signal performance is achieved with 50 Ohm. In the following picture this is shown for single ended and differential outputs.



www.valuetronics.com

- Via a resistor to the complement output (only available for the differential output frontends E4838A and E4843A). Optimum signal performance is achieved with 100 Ohms.



Example

```
:_test:SGEN:PDAT1:OUTP:TCON DIFF
:_test:SGEN:PDAT1:TERM1:OUTP:TCON DIFF
:_test:SGEN:PPUL1:OUTP:TCON VOLT
:_test:SGEN:PPUL1:TERM1:OUTP:TCON DIFF
:_test:MOD2:CONN3:OUTP:TCON VOLT
```

:OUTPut:TCONfig?

Syntax

:<Handle>:SGENeral:**PDATa(*):OUTPut:**TCONfig?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):OUTPut:**TCONfig?

:<Handle>:SGENeral:**PPULse(*):OUTPut:**TCONfig?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):OUTPut:**TCONfig?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):OUTPut:**TCONfig?

Return Value

Returns the termination model for output frontends (see *":OUTPut:TCONfig" on page 264*).

Example

```
:_test:SGEN:PDAT1:OUTP:TCON?
:_test:SGEN:PDAT1:TERM1:OUTP:TCON?
:_test:SGEN:PPUL1:OUTP:TCON?
:_test:SGEN:PPUL1:TERM1:OUTP:TCON?
:_test:MOD2:CONN3:OUTP:TCON?
```

might return:

```
DIFF
```


www.valuetronics.com

:OUTPut:DIM Pedance:EXTernal

Syntax

SGENeral:**PDATa**(*):OUTPut:DIMPedance:EXTernal <Impedance>

SGENeral:**PDATa**(*)**:TERMinal**(*):OUTPut:DIMPedance:EXTernal <Impedance>

SGENeral:**PPULse**(*):OUTPut:DIMPedance:EXTernal <Impedance>

SGENeral:**PPULse**(*)**:TERMinal**(*):OUTPut:DIMPedance:EXTernal <Impedance>

CGR(*):MOD(*):**CONNector**(*):OUTP:DIMP:EXT <Impedance>

Parameters

<Impedance> Sets the impedance between the OUT and \OUT\ outputs of an E4838A, E4843A, E4862A or E4864A frontend when the differential termination model is selected (see *":OUTPut:TCONfig" on page 264*).

Usually 100 Ohms are used for termination. The E4843A supports any value exceeding 20 Ohms.

Example

```
:_test:SGEN:PDAT:OUTP:DIMP:EXT 100
:_test:SGEN:PDAT1:TERM2:OUTP:DIMP:EXT 50
:_test:SGEN:PPUL:OUTP:DIMP:EXT 100
:_test:SGEN:PPUL2:TERM3:OUTP:DIMP:EXT 100
```

:OUTPut:DIM Pedance:EXTernal?

Syntax

SGENeral:**PDATa**(*):OUTPut:DIMPedance:EXTernal?

SGENeral:**PDATa**(*)**:TERMinal**(*):OUTPut:DIMPedance:EXTernal?

SGENeral:**PPULse**(*):OUTPut:DIMPedance:EXTernal?

SGENeral:**PPULse**(*)**:TERMinal**(*):OUTPut:DIMPedance:EXTernal?

CGRoup(*):MODule(*):**CONNector**(*):OUTPut:DIMPedance:EXTernal?

Return Value

Returns the differential termination resistor (see *":OUTPut:TCONfig" on page 264*).

Example

```
:_test:SGEN:PDAT:OUTP:DIMP:EXT?
:_test:SGEN:PDAT1:TERM2:OUTP:DIMP:EXT?
:_test:SGEN:PPUL:OUTP:DIMP:EXT?
:_test:SGEN:PPUL2:TERM3:OUTP:DIMP:EXT?
:_test:MOD2:CONN3:OUTP:DIMP:EXT?
```

might return:

```
100
```


www.valuetronics.com

:OUTPut:CAConfiguration[:MODE]

Syntax

:<Handle>:SGENeral:**PDATa(*):OUTPut:**CAConfiguration[:MODE] <OFF | D2 | D4 | A2>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):OUTPut:**CAConfiguration[:MODE] <OFF | D2 | D4 | A2>

:<Handle>:SGENeral:**PPULse(*):OUTPut:**CAConfiguration[:MODE] <OFF | D2 | D4 | A2>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):OUTPut:**CAConfiguration[:MODE] <OFF | D2 | D4 | A2>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):OUTPut:**CAConfiguration[:MODE] <OFF | D2 | D4 | A2>

Parameters

<OFF | D2 | D4 | A2> Selects the channel-add mode of a port/terminal/ connector. The following modes are available:

- OFF (no channel addition). This is the reset value.
- D2 is a digital add of two channels (XOR). It can be used on the second and fourth frontend slot of an E4832A module if output frontends are plugged into this slot and into the one immediately above.
- D4 is a digital add of four channels (XOR). It can be used on the fourth frontend slot of an E4832A module if the whole module is equipped with output frontends.
- A2 is an analog add available only for E4838A frontends. It can be used on the second and fourth frontend slots of an E4832A module if an E4838A frontend is fitted there.

  If A2 Mode is selected, the parameters specified by the following commands are additionally available:

  – *":VOLTage[:LEVel][:IMMediate]:CAConfiguration:LOW" on page 224*

  – *":PULSe:TRANsition:CAConfiguration[:LEADing]" on page 192*

  – *":PULSe:TRANsition:CAConfiguration:TRAiling" on page 194*

Example

```
:_test:SGEN:PDAT1:OUTP:CAC D2

:_test:SGEN:PDAT1:TERM1:OUTP:CAC:MODE D4

:_test:SGEN:PPUL1:OUTP:CAC D2

:_test:SGEN:PPUL1:TERM1:OUTP:IMP:CAC:MODE D4

:_test:cgr1:mod2:conn4:outp:cac A2
```


www.valuetronics.com

:OUTPut:CACConfiguration[:MODE]?

Syntax

:<Handle>:SGENeral:**PDATa(*):OUTPut:**CAConfiguration[:MODE]?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):OUTPut:**CAConfiguration [:MODE]?

:<Handle>:SGENeral:**PPULse(*):OUTPut:**CAConfiguration[:MODE]?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):OUTPut:**CAConfiguration [:MODE]?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):OUTPut:**CAConfiguration [:MODE]?

Return Value

Returns the current channel addition mode of the specified port/terminal/connector.

Example

```
:_test:SGEN:PDAT1:OUTP:CAC?
:_test:SGEN:PDAT1:TERM1:OUTP:CAC:MODE?
:_test:SGEN:PPUL1:OUTP:CAC?
:_test:SGEN:PPUL1:TERM1:OUTP:CAC:MODE?
:_test:cgr1:mod2:conn4:outp:cac?
```

might return:

```
D2
```

:OUTPut:DISConnect:MODE?

Syntax

:<Handle>:CGRoup*:MODule*:CONNector*:OUTPut:DISConnect:MODE RELay| VOLTage

:<Handle>:SGENeral:PDATa*:OUTPut:DISConnect:MODE RELay| VOLTage

:<Handle>:SGENeral:PDATa*:TERMinal:OUTPut:DISConnect:MODE RELay| VOLTage

:<Handle>:SGENeral:PPULse*:OUTPut:DISConnect:MODE RELay| VOLTage

:<Handle>:SGENeral:PPULse*:TERMinal:OUTPut:DISConnect:MODE RELay| VOLTage

Return Value

Returns how the specified connector is to be disconnected when the `SGENeral:GLOBal:CONNect OFF` command is executed. Can be either RELay (disconnected by switching off the corresponding relay) or VOLtage (disconnected by reducing the voltage to 0).

Example

```
:_test:CGR1:MOD1:CONN1:OUTP:DISC:MODE?
```

might return:

```
REL
```


www.valuetronics.com

:OUTPut:DISConnect:MODE

Syntax

:<Handle>:CGRoup*:MODule*:CONNector*:OUTPut:DISConnect:MODE RELay| VOLTage

:<Handle>:SGENeral:PDATa*:OUTPut:DISConnect:MODE RELay| VOLTage

:<Handle>:SGENeral:PDATa*:TERMinal:OUTPut:DISConnect:MODE RELay| VOLTage

:<Handle>:SGENeral:PPULse*:OUTPut:DISConnect:MODE RELay| VOLTage

:<Handle>:SGENeral:PPULse*:TERMinal:OUTPut:DISConnect:MODE RELay| VOLTage

Parameters

<RELay|VOLTage> Sets how the connector, terminal or port is to be disconnected when the `SGENeral:GLOBal:CONNect OFF` command is executed:

- RELay

  Disconnects the specified connector by switching the corresponding relay.

- VOLTage

  Disconnects the specified connector by setting the voltage to 0.

Example

```
:_test:CGR1:MOD1:CONN1:OUTP:DISC:MODE VOLT
```


www.valuetronics.com

Optical Output Commands

The optical output commands are used for ParBERT optical generators, such as the E4810A. The following commands are available:

- STATe

  Sets/returns the state of the connector, terminal, or port.

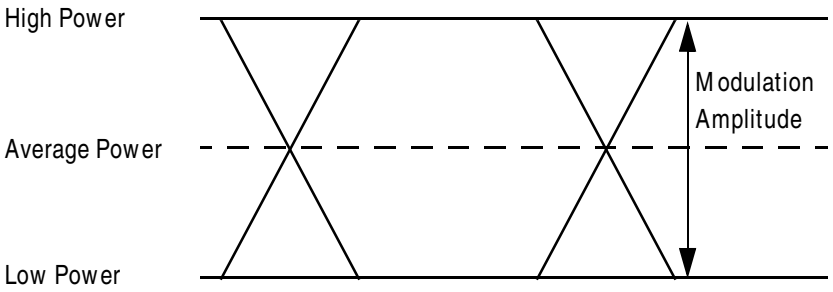
- METHod

  Sets/returns the state of the connector, terminal, or port.

- POWer:UNIT

- Optical Power Parameters

  The following parameters are used for the optical power:



Not shown: Extinction ratio (~ High/Low).

There are three possibilities of defining the optical waveform:

- Define the high and low power (`HIGH_LOW` method)
- Define the average power and the modulation amplitude (`AVER_MAMP` method)
- Define the average power and the extinction ratio (`AVER_ERAT` method)

www.valuetronics.com

The following table shows the relationship between the :OUTPut:OPTic:METHod parameter and the commands for setting the power levels:

Parameters	:OUTPut:OPTic:METHod		
	AVER_ERAT	AVER_MAMP	HIGH_LOW
High	Calculated	Calculated	Direct entry
Low	Calculated	Calculated	Direct entry
Average	Direct entry	Direct entry	Calculated
Modulation Amplitude	Calculated	Direct entry	Calculated
Extinction Ratio	Direct entry	Calculated	Calculated

The parameters are calculated as follows (based on the High and Low power values). Note that all powers are given in Watt.

- Average Power = (High + Low)/2
- Modulation Amplitude = High - Low
- Extinction $Ratio_{None}$ = High/Low
- Extinction $Ratio_{dB}$ = 10*log(High/Low)

See also *"Optical Input Commands" on page 253*.

:OUTPut:OPTic[:STATe][:VALue]

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:[STATe]:[VALue] <OFF| ON>

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:[STATe]:[VALue] <OFF| ON>

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:[STATe]:[VALue] <OFF| ON>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:[STATe]:[VALue] <OFF| ON>

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:[STATe]:[VALue] <OFF| ON>

Parameters

<OFF|ON> Enables/disables the laser on the E/O converter. The electrical board is enabled/disabled automatically if the connector species is set to OPT.

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:STAT:VAL OFF
:_test:CGR1:MOD1:CONN1:OUTP:OPT OFF
```


www.valuetronics.com

:OUTPut:OPTic[:STATe][:VALue]?

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:[STATe]:[VALue]? <NFORce| FORCe>

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:[STATe]:[VALue]? <NFORce| FORCe>

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:[STATe]:[VALue]? <NFORce| FORCe>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:[STATe]:[VALue]? <NFORce| FORCe>

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:[STATe]:[VALue]? <NFORce| FORCe>

Return Value

The state of the laser on the E/O converter.

Parameters

<NFORce|FORCe> Defines how the query is executed:

- NFORce

  Default value. Only the firmware is queried for the current state.

- FORCe

  When FORCe is entered, the firmware checks the current condition in the hardware. This is required to make sure that the output is really switched on, or to check if the protection circuit within the output has turned off the channel to protect it from damage.

NOTE The `FORCe` query is slower than `NFORce`.

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:STAT:VAL?
:_test:CGR1:MOD1:CONN1:OUTP:OPT?
```

might return:

```
ON
```


www.valuetronics.com

:OUTPut:OPTic:M ETHod[:VALue]

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:METHod:[VALue] <AVER_ERAT | AVER_MAMP | HIGH_LOW>

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:METHod:[VALue] <AVER_ERAT | AVER_MAMP | HIGH_LOW>

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:METHod:[VALue] <AVER_ERAT | AVER_MAMP | HIGH_LOW>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:METHod:[VALue] <AVER_ERAT | AVER_MAMP | HIGH_LOW>

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:METHod:[VALue] <AVER_ERAT | AVER_MAMP | HIGH_LOW>

Parameters

<AVER_ERAT | AVER_MAMP | HIGH_LOW> Sets the method used to define the optical levels:

- `AVER_ERAT` (Average Power / Extinction Ratio)
- `AVER_MAMP` (Average Power / Modulation Amplitude)
- `HIGH_LOW` (High Power / Low Power)

The optical parameters can always be queried, but only the parameters specified by `METHod` can be set. For example, if the `METHod` is `AVER_MAMP`, only average power and modulation amplitude can be directly set. Trying to set high power in this case results in an error.

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:METH HIGH_LOW
:_test:CGR1:MOD1:CONN1:OUTP:OPT:UNIT WATT
:_test:CGR1:MOD1:CONN1:OUTP:OPT:HIGH 0.865242735174E-3
```

:OUTPut:OPTic:M ETHod[:VALue]?

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:METHod:[VALue]?

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:METHod:[VALue]?

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:METHod:[VALue]?

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:METHod:[VALue]?

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:METHod:[VALue]?

Return Value

Returns the method used to set the optical levels:

- `AVER_ERAT` (Average Power / Extinction Ratio)
- `AVER_MAMP` (Average Power / Modulation Amplitude)
- `HIGH_LOW` (High Power / Low Power)

www.valuetronics.com

:OUTPut:OPTic:POWer:UNIT[:VALue]

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:POWer:UNIT:[VALue] <WATT| DBM>

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:POWer:UNIT:[VALue] <WATT| DBM>

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:POWer:UNIT:[VALue] <WATT| DBM>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:POWer:UNIT:[VALue] <WATT| DBM>

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:POWer:UNIT:[VALue] <WATT| DBM>

Parameters

<WATT|DBM> Selects the unit for all power parameters of the selected optical output port or terminal.

dBm is a unitless logarithmic alternative to Watt. The following calculations are used for transformations:

- $Power_{dBm} = 10*log(Power_{Watt}/0.001\ W)$
- $Power_{dBm} = 10*log(Power_{Watt}/1\ mW)$
- $Power_{Watt} = 1\ mW * 10\text{^}(Power_{dBm}/10)$

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:UNIT:VAL WATT
```

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:UNIT WATT
```

:OUTPut:OPTic:POWer:UNIT[:VALue]?

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:POWer:UNIT:[VALue]?

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:POWer:UNIT:[VALue]?

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:POWer:UNIT:[VALue]?

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:POWer:UNIT:[VALue]?

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:POWer:UNIT:[VALue]?

Return Value

Used power unit for setting and querying the optical power parameters.

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:UNIT:VAL?
```

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:UNIT?
```

might return:

```
WATT
```


www.valuetronics.com

:OUTPut:OPTic:POWer:AVERage[:VALue]

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:POWer:AVERage:[VALue] <avg. power>

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:POWer:AVERage:[VALue] <avg. power>

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:POWer:AVERage:[VALue] <avg. power>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:POWer:AVERage:[VALue] <avg. power>

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:POWer:AVERage:[VALue] <avg. power>

Parameters

<avg. power> Average power of the selected connector, terminal, or port.

NOTE

When setting this parameter, the parameter defined by `:OUTPut:OPTic:POW:UNIT` is used.

This command can also only be used when the `:OUTPut:OPTic:METHod` command is set appropriately.

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:METH:VAL AVER_ERAT
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:UNIT:VAL WATT
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:AVER:VAL 0.501187233627E-3

:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:METH AVER_MAMP
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:UNIT WATT
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:AVER 0.501187233627E-3
```

:OUTPut:OPTic:POWer:AVERage[:VALue]?

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:POWer:AVERage:[VALue] ?

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:POWer:AVERage:[VALue]?

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:POWer:AVERage:[VALue]?

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:POWer:AVERage:[VALue]?

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:POWer:AVERage:[VALue]?

Return Value

Average power of the selected connector, terminal, or port.

www.valuetronics.com

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:AVER:VAL?
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:AVER?
```

might return:

```
0.501187233627E-3
```

:OUTPut:OPTic:POWer:MAMPlitude[:VALue]

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:POWer:MAMPlitude[:VALue] <mod. ampl.>

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:POWer:MAMPlitude[:VALue] <mod. ampl.>

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:POWer:MAMPlitude[:VALue] <mod. ampl.>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:POWer:MAMPlitude[:VALue] <mod. ampl.>

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:POWer:MAMPlitude[:VALue] <mod. ampl.>

Parameters

<mod. ampl.> Modulation amplitude of the selected connector, terminal, or port.

NOTE

When setting this parameter, the parameter defined by `:OUTPut:OPTic:POW:UNIT` is used.

This command can also only be used when the `:OUTPut:OPTic:METHod` command is set appropriately.

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:METH DBM
:_test:CGR1:MOD1:CONN1:OUTP:OPT:MAMP -1.37802405912
```


www.valuetronics.com

:OUTPut:OPTic:POWer:M AM Plitude[: VALue]?

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:POWer: MAMPlitude:[VALue]?

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:POWer:MAMPlitude:[VALue]?

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:POWer:MAMPlitude:[VALue]?

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:POWer: MAMPlitude:[VALue]?

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:POWer: MAMPlitude:[VALue]?

Return Value

Modulation amplitude of the selected connector, terminal, or port.

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:MAMP?
```

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:MAMP:VAL?
```

might return:

```
-1.37802405912
```

:OUTPut:OPTic:POWer:HIGH[:VALue]

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:POWer:HIGH: [VALue] <high power>

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:POWer:HIGH:[VALue] <high power>

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:POWer:HIGH:[VALue] <high power>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:POWer:HIGH: [VALue] <high power>

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:POWer:HIGH: [VALue] <high power>

Parameters

<high power> High power of the selected connector, terminal, or port.

NOTE

When setting this parameter, the parameter defined by `:OUTPut:OPTic: POW:UNIT` is used.

This command can also only be used when the `:OUTPut:OPTic:METHod` command is set appropriately.

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:UNIT DBM
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:HIGH -0.628620384795
```


www.valuetronics.com

:OUTPut:OPTic:POWer:HIGH[:VALue]?

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:POWer:HIGH:[VALue]?

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:POWer:HIGH:[VALue]?

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:POWer:HIGH:[VALue]?

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:POWer:HIGH:[VALue]?

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:POWer:HIGH:[VALue]?

Return Value

High power of the selected connector, terminal, or port.

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:HIGH?
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:HIGH:VAL?
```

might return:

```
-0.628620384795
```

:OUTPut:OPTic:POWer:LOW[:VALue]

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:POWer:LOW:[VALue] <low power>

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:POWer:LOW:[VALue] <low power>

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:POWer:LOW:[VALue] <low power>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:POWer:LOW:[VALue] <low power>

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:POWer:LOW:[VALue] <low power>

Parameters

<low power> Low power of the selected connector, terminal, or port.

NOTE

When setting this parameter, the parameter defined by `:OUTPut:OPTic:POW:UNIT` is used.

This command can also only be used when the `:OUTPut:OPTic:METHod` command is set appropriately.

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:UNIT DBM
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:LOW -8.6286203848
```


www.valuetronics.com

:OUTPut:OPTic:POWer:LOW[:VALue]?

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:POWer:LOW:[VALue]?

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:POWer:LOW:[VALue]?

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:POWer:LOW:[VALue]?

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:POWer:LOW:[VALue] <low power>

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:POWer:LOW:[VALue] <low power>

Return Value

Low power of the selected connector, terminal, or port.

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:POW:LOW?
```

might return:

```
-8.6286203848
```

:OUTPut:OPTic:ERATio[:VALue]

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:ERATio:[VALue] <ext. ratio>

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:ERATio:[VALue] <ext. ratio>

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:ERATio:[VALue] <ext. ratio>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:ERATio:[VALue] <ext. ratio>

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:ERATio:[VALue] <ext. ratio>

Parameters

<ext. ratio> Extinction ratio of the selected connector, terminal, or port.

NOTE

When setting this parameter, the parameter defined by `:OUTPut:OPTic:POW:ERAT:UNIT` is used.

This command can also only be used when the `:OUTPut:OPTic:METHod` command is set appropriately.

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:ERAT 8
:_test:CGR1:MOD1:CONN1:OUTP:OPT:ERAT:VAL 8
```


www.valuetronics.com

:OUTPut:OPTic:ERATio[:VALue]?

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:ERATio:[VALue]?

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:ERATio:[VALue]?

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:ERATio:[VALue]?

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:ERATio:[VALue]?

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:ERATio:[VALue]?

Return Value

Extinction ratio of the selected connector, terminal, or port according to `:OUTPut:OPTic:ERATio:UNIT`.

Examples

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:ERAT?
:_test:CGR1:MOD1:CONN1:OUTP:OPT:ERAT:VAL?
```

might return:

```
8.0E+0
```

:OUTPut:OPTic:ERATio:UNIT[:VALue]

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:ERATio:UNIT:[VALue] <NONE| DB>

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:ERATio:UNIT:[VALue] <NONE| DB>

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:ERATio:UNIT:[VALue] <NONE| DB>

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:ERATio:[VALue] <NONE| DB>

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:ERATio:[VALue] <NONE| DB>

Parameters

<NONE|DB> Selects the unit for the extinction ratio of the selected optical connector, port, or terminal.

- NONE (linear value used)
- DB (logarithmic value used)

Example

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:UNIT DB
:_test:CGR1:MOD1:CONN1:OUTP:OPT:UNIT:VAL DB
```


www.valuetronics.com

:OUTPut:OPTic:ERATio:UNIT[:VALue]?

Syntax

:<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):OUTPut:OPTic:ERATio:UNIT:[VALue]?

:<Handle>:SGENeral:PDATa(*):OUTPut:OPTic:ERATio:UNIT:[VALue]?

:<Handle>:SGENeral:PPULse(*):OUTPut:OPTic:ERATio:UNIT:[VALue]?

:<Handle>:SGENeral:PDATa(*):TERMinal(*):OUTPut:OPTic:ERATio:UNIT:[VALue]?

:<Handle>:SGENeral:PPULse(*):TERMinal(*):OUTPut:OPTic:ERATio:UNIT:[VALue]?

Return Value

Unit used for the extinction ratio of the selected optical connector, port, or terminal. Can be:

- NONE (linear value used)
- DB (logarithmic value used)

Example

```
:_test:CGR1:MOD1:CONN1:OUTP:OPT:UNIT?
:_test:CGR1:MOD1:CONN1:OUTP:OPT:UNIT:VAL?
```

might return:

```
DB
```


www.valuetronics.com

Port Administration Commands

The following commands are available for managing pulse and data ports. The parameters can be specified separately for pulse and data ports.

The MUX commands—also available for pulse and data ports—are described in *"Timing Parameter Commands" on page 178*.

:APPend

Syntax

:<Handle>:SGENeral:**PDATa(*):**APPend <"Port Type"> [, <No. of Terminals>] [, <"Port Name">], [<Species>]

:<Handle>:SGENeral:**PPULse(*):**APPend <"Port Type"> [, <No. of Terminals>] [, <"Port Name">], [<Species>]

Parameters

<"Port Type"> A quoted string of the port type, either INPUT_PORT or OUTPUT_PORT, seen from DUT point of view.

[<No. of Terminals>] The integer number of terminals used in the new port (<NR1>). This is an optional parameter.

[<"Port Name">] A quoted string of the name used to identify the port. This is an optional parameter.

[<Species>] (Optional) Species of the port. Possible values are electrical (ELEC) or optical (OPT). All terminals of the port are assigned this species. Default is electrical.

Creates and appends a new port to the list of ports. The port type is required. The number of terminals, the port name, and species are optional. The suffix of PDATa and PPULse is ignored.

Example

```
:_test:SGEN:PDAT:APP "INPUT_PORT", 4, "DataBus", ELEC
:_test:SGEN:PPUL:APP "INPUT_PORT", 4, "DataBus", ELEC
```


www.valuetronics.com

:LIST?

Syntax :<Handle>:SGENeral:**PDATa(*):**LIST?

:<Handle>:SGENeral:**PPULse(*):**LIST?

Return Value Returns a comma-separated list of port names. If a port has no name specified, an empty quoted string will be returned. The suffix of PDATa or PPULse is ignored.

Example

```
:_test:sgen:pdat:list?
:_test:sgen:ppul:list?
```

might return:

```
"DataBus", "AddressBus", "","ControlBus"
```

:ATYPes?

Syntax :<Handle>:SGENeral:**PDATa(*):**ATYPes?

:<Handle>:SGENeral:**PPULse(*):**ATYPes?

Return Value Returns a comma-separated list of available predefined port TYPes. The suffix of PDATa or PPULse is ignored. Currently there are two types available, INPUT_PORT and OUTPUT_PORT (only for data ports).

Example

```
:_test:sgeneral:pdata:atypes?
```

might return:

```
"INPUT_PORT", "OUTPUT_PORT"
```

:DELete

Syntax :<Handle>:SGENeral:**PDATa(*):**DELete

:<Handle>:SGENeral:**PPULse(*):**DELete

Description Deletes the port specified by the suffix of PDATa or PPULse.

Example

```
:_test:SGEN:PDAT1:DEL
:_test:SGEN:PPUL1:DEL
```


www.valuetronics.com

:RENname

Syntax :<Handle>:SGENeral:**PDATa(*):**RENname <"New Name">

:<Handle>:SGENeral:**PPULse(*):**RENname <"New Name">

Parameters **<"New Name">** A quoted string of the new name of the port

Renames the port specified by the suffix of PDATa or PPULse.

Example

```
:_test:sgen:pdat1:ren "AddrBus"
:_test:sgen:pdat:list?
"AddrBus"
```

```
:_test:sgen:ppul1:ren "AddrBus"
:_test:sgen:ppul:list?
"AddrBus"
```

:NAME?

Syntax :<Handle>:SGENeral:**PDATa(*):**NAME?

:<Handle>:SGENeral:**PPULse(*):**NAME?

Return Value Returns the name of the specified port.

Example

```
:_test:SGEN:PDAT1:NAME?
```

```
:_test:SGEN:PPUL1:NAME?
```

might return:

```
"AddrBus"
```

:TYPE?

Syntax :<Handle>:SGENeral:**PDATa(*):**TYPE?

:<Handle>:SGENeral:**PPULse(*):**TYPE?

Return Value Returns the TYPE of this port.

Example

```
:_test:SGEN:PDAT1:TYPE?
```

might return:

```
"INPUT_PORT"
```

```
:_test:SGEN:PPUL1:TYPE?
```

might return:

```
"INPUT_PORT"
```


www.valuetronics.com

:CALibration:CDELay

Syntax :<Handle>:SGENeral:**PDATa(*)**:CALibration:CDELay <Cable Delay>

:<Handle>:SGENeral:**PPULse(*)**:CALibration:CDELay <Cable Delay>

Parameters **<Cable Delay>** Cable Delay value (<NRf>).

Sets a cable delay for the specified port to synchronize the signals at the DUT terminals.

NOTE Delay Auto Calibration must be performed before the cable delay can be set. See *":<Handle>:SGENeral:GLOBal:CALibration:SELF DELay" on page 140*.

Example

```
:_test:SGEN:PDAT1:CAL:CDEL 6.5e-9
```

:CALibration:CDELay?

Syntax :<Handle>:SGENeral:**PDATa(*):**CALibration:CDELay?

:<Handle>:SGENeral:**PPULse(*):**CALibration:CDELay?

Return Value Returns the current cable delay for the specified port.

Example

```
:_test:SGEN:PDAT1:CAL:CDEL?
:_test:SGEN:PPUL1:CAL:CDEL?
```

might return:

```
6.500000E-009
```

:SPECies?

Syntax :<Handle>:SGENeral:**PDATa(*):**SPECies?

:<Handle>:SGENeral:**PPULse(*):**SPECies?

Return Value Returns the port's species. This can be either electrical (`ELEC`) or optical (`OPT`).

Example

```
:_test:SGEN:PDAT1:SPEC?
:_test:SGEN:PPUL1:SPEC?
```

might return:

```
ELEC
```


www.valuetronics.com

Terminal Administration Commands

The following commands are available for administration of terminals. The parameters can be specified separately for pulse and data ports.

:APPend

Syntax :<Handle>:SGENeral:**PDATa(*):TERMinal(*):**APPend [<"Terminal Name">] [,<Position>]

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):**APPend [<"Terminal Name">] [,<Position>]

Parameters **[<"Terminal Name">]** Quoted String of the name of new terminal.

[<Position>] Integer number of the position where the new terminal will be added.

Creates and appends a new terminal to a port. The terminal name and the position of the new terminal are optional. If the specified position is zero, the new terminal is placed a the beginning of the list. If "Position" is omitted or greater than the number of terminals, the new terminal is appended to the list. Otherwise, the new terminal is inserted *after* the specified position.

Example

```
:_test:SGEN:PDAT1:TERM1:APP "Data5", 5
:_test:SGEN:PPUL1:TERM1:APP "Clock", 5
```

:LIST?

Syntax :<Handle>:SGENeral:**PDATa(*):TERMinal(*):**LIST?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):**LIST?

Return Value Returns a comma-separated list of all terminal names contained in the specified port.

Example

```
:_test:SGEN:PDAT1:TERM1:LIST?
"T1", "T2", "T3", "T4", "Data5"
:_test:SGEN:PPUL1:TERM1:LIST?
"T1", "T2", "T3", "T4", "Clock"
```


www.valuetronics.com

:DELete

Syntax :<Handle>:SGENeral:**PDATa(*):TERMinal(*):**DELete

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):**DELete

Description Deletes the specified terminal.

Example

```
:_test:SGEN:PDAT1:TERM1:DEL
:_test:SGEN:PPUL1:TERM1:DEL
```

:REName

Syntax :<Handle>:SGENeral:**PDATa(*):TERMinal(*):**REName <"New Name">

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):**REName <"New Name">

Parameters **<"New Name">** A quoted string of the new name for the terminal.

Renames the specified terminal.

Example

```
:_test:sgen:pdat1:term5:REN "Addr5"
:_test:SGEN:PDAT1:TERM1:LIST?
:_test:sgen:ppul1:term5:REN "Addr5"
:_test:SGEN:PPUL1:TERM1:LIST?
```

might return:

```
"T1", "T2", "T3", "T4", "Addr5"
```

:NAME?

Syntax :<Handle>:SGENeral:**PDATa(*):TERMinal(*):**NAME?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):**NAME?

Return Value Returns the name of the specified terminal.

Example

```
:_test:SGEN:PDAT1:TERM1:NAME?
:_test:SGEN:PPUL1:TERM1:NAME?
```

might return:

```
"T1"
```


www.valuetronics.com

:TYPE?

Syntax :<Handle>:SGENeral:**PDATa(*):**TERMinal(*):TYPE?

:<Handle>:SGENeral:**PPULse(*):**TERMinal(*):TYPE?

Return Value Returns the type of the specified terminal.

Example

```
:_test:SGEN:PDAT1:TERM1:TYPE?
```

might return:

```
"E4843A"
```

:M OVE

Syntax :<Handle>:SGENeral:**PDATa(*):TERMinal(*):**MOVE <Distance>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):**MOVE <Distance>

Parameters **<Distance>** Number of positions this terminal is to be moved by (<NR1>).

Moves the specified terminal to a new position.

Example

```
:_test:SGEN:PDAT1:TERM1:MOVE 1
:_test:SGEN:PDAT1:TERM1:LIST?
```

might return:

```
"T2", "T1", "T3", "T4", "Addr5"
```

```
:_test:SGEN:PPUL1:TERM1:MOVE 1
:_test:SGEN:PPUL1:TERM1:LIST?
```

might return:

```
"T2", "T1", "T3", "T4", "Addr5"
```


www.valuetronics.com

:CALibration:CDELay

Syntax :<Handle>:SGENeral:**PDATa(*):TERMinal(*):**CALibration:CDELay <Cable Delay>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):**CALibration:CDELay <Cable Delay>

Sets a cable delay for the specified terminal in the specific port to synchronize the signals at the DUT terminals.

NOTE Delay Auto Calibration must be performed before the cable delay can be set. See *":<Handle>:SGENeral:GLOBal:CALibration:SELF DELay" on page 140*.

Parameters **<Cable Delay>** Cable Delay value (<NRf>).

Example

```
:_test:SGEN:PDAT1:TERM1:CAL:CDEL 6.5e-9
:_test:SGEN:PPUL1:TERM1:CAL:CDEL 6.5e-9
```

:CALibration:CDELay?

Syntax :<Handle>:SGENeral:**PDATa(*):TERMinal(*):**CALibration:CDELay?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):**CALibration:CDELay?

Return Value Returns the current cable delay for the specified terminal of the specified terminal.

Example

```
:_test:SGEN:PDAT1:TERM1:CAL:CDEL?
:_test:SGEN:PPUL1:TERM1:CAL:CDEL?
```

might return:

```
6.500000E-009
```


www.valuetronics.com

Connector Administration Commands

The following commands are available to associate the connectors of the Agilent 81250 System to the terminals of the DUT. Other connector administration commands can be found in the *"[:CGRoup(*)]:MODule(*):CONNector(*) Subsystem" on page 127*.

The commands are available for pulse and data ports. They build the :<Handle>:SGENeral:CONNect subsystem.

:REMove

Syntax

:<Handle>:SGENeral:CONNect:**PDATa(*):**REMove

:<Handle>:SGENeral:CONNect:**PDATa(*):TERMinal(*):**REMove

:<Handle>:SGENeral:CONNect:**PPULse(*):**REMove

:<Handle>:SGENeral:CONNect:**PPULse(*):TERMinal(*)**:REMove

Description

Disconnects all terminals within the specified port, or the specified terminal.

Example

```
:_test:sgen:conn:pdat2:rem
:_test:sgeneral:connect:pdata1:terminal2:remove
:_test:sgen:conn:ppul2:rem
:_test:sgeneral:connect:ppul1:terminal2:remove
```


www.valuetronics.com

:TERMinal(*):[TO]

Syntax

:<Handle>:SGENeral:CONNect:**PDATa(*):TERMinal(*):**[TO] <@ChannelList>

:<Handle>:SGENeral:CONNect:**PPULse(*):TERMinal(*):**[TO] <@ChannelList>

Parameters

<@ChannelList> List of channels.

Connects the specified terminal to the specified channel or channels. If the number of connectors exceeds the number of terminals defined in the specified port, the connection will be continued to the next port. In other words, the suffix of the port and the suffix of the terminal are used as a start value. Only connectors of the same type can be combined to a port.

The syntax of the channel list is as follows:

<ChannelListExpr>	::=	"(@"<ChannelList>")"
<ChannelList>	::=	<ChannelListElem> [","<ChannelListElem>]
<ChannelListElem>	::=	<Connector> | (<Connector> ":" <Connector>)
<Connector>	::=	<ClockGroupNr><ModuleNr><ConnectorNr>
<ClockGroupNr>	::=	<Digit><Digit>
<ModuleNr>	::=	<Digit><Digit>
<ConnectorNr>	::=	<Digit><Digit><Digit>
<Digit>	::=	"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Syntax Examples:

- (@0102004) specifies connector 4 in clock group 1, module 2
- (@0102001:0102004,0103001) specifies connectors 1 to 4 of module 2 and connector 1 of module 3, both in clockgroup 1

Example

```
:_test:SGEN:CONN:PDAT1:TERM2:TO (@0102001)
:_test:SGEN:CONN:PPUL1:TERM2:TO (@0102001)
```


www.valuetronics.com

:TERMinal(*):[TO]?

Syntax

:<Handle>:SGENeral:CONNect:**PDATa(*):TERMinal(*):**[TO]? [ALL]

:<Handle>:SGENeral:CONNect:**PPULse(*):TERMinal(*):**[TO]? [ALL]

Parameters

ALL If specified, returns all channels connected to this terminal (only when channel addition is used).

This query returns the channel to which the specified terminal is connected. If the terminal is not connected an empty channel list is returned (for example, @). The optional parameter ALL returns a more detailed list of connectors if the channel add mode is active.

Example

```
:_test:sgen:conn:pdat1:term2:to? all
:_test:sgen:conn:ppul1:term2:to? all
```

might return:

```
(@0102001, 0102002)
```


www.valuetronics.com

Error Analysis Commands

The Error Analysis Commands are used to report and reset bit errors on specific ports or terminals.

:FETCh:ERRor:ANY?

Syntax :<Handle>:SGENeral:**PDATa(*):**FETCh:ERRor:ANY?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):**FETCh:ERRor:ANY?

Return Value Returns 0 for no errors, 1 if an error was found for the specified data port or terminal.

This can be used to increase program speed. Uploading of memory segments can be completely avoided, when it is known, that there are no errors at all.

Example

```
:_test:SGEN:PDAT1:FETC:ERR:ANY?
:_test:SGEN:PDAT1:TERM2:FETC:ERR:ANY?
```

might return:

```
0
```

:FETCh[:ECOunt]?

Syntax :<Handle>:SGENeral:PDATa(*):FETCh [:ECOunt]? [(<TermChannelList >) | <PSUM>]

Parameters **<PSUM>** Generates an accumulation of the received bits count (Accumulated Number of Bits) and the error bits count (Accumulated Number of Errors) for all terminals of the whole port.

<TermChannelList> Selects specific terminals of a data port. If the argument is omitted, all measured values within the specified port are reported.

In the error count mode this query returns the number of received bits and the number of erroneous bits. The counters are set to zero after a stop/start request or reset. The results are contained in a comma-separated list.

www.valuetronics.com

For example, the channel list (@1:3) addresses 3 channels within a data port and reports 6 values in a comma-separated list, for example, 2.345e5,0,2.33e5,0,2.32e5,5. Each pair of values relates to one terminal. The first value is the number of received bits, the second value is the number of erroneous bits (failed bits). One pair is sampled at the same time, the subsequent pair is sampled some cycles later. In the example, there are 5 errors counted at terminal 3.

The syntax of the terminal channel list is as follows:

<TermChannelListExpr>	::=	“(@”<TermChannelList>“)”
<TermChannelList>	::=	<TermChannelListElem> [“,” <TermChannelListElem>]
<TermChannelListElem>	::=	<TermNr> | (<TermNr> “:” <TermNr>)
<TermNr>	::=	<Digit>[<Digit>]
<Digit>	::=	“0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

Return Value

On parameter <PSUM>:

Comma-separated list of two exponential float values.

- The first value represents the accumulation of the received bits count (Accumulated Number of Bits).
- The second value is the accumulation of the the error bits count (Accumulated Number of Errors) for all terminals of the whole port.

On parameter <TermChannelList>:

Comma-separated list of tuple exponential float values. There are as many tuples as addressed by the <TermChannelList>. The values of the tuple are likewise comma-separated.

- The first value of the tuple represents the accumulation of the received bits count (Accumulated Number of Bits).
- The second value of the tuple represents the accumulation of the error bits count (Accumulated Number of Errors) for all terminals of the whole port.

When no parameter provided:

Comma-separated list of tuple exponential float values. There are as many tuples as terminals contained in the specified port. The values of the tuple are likewise comma-separated.

- The first value of the tuple represents the accumulation of the received bits count (Accumulated Number of Bits).

www.valuetronics.com

- The second value of the tuple is the accumulation of the the error bits count (Accumulated Number of Errors) for all terminals of the whole port.

Syntax Examples: (@4) specifies terminal 4.

(@3:5,9) specifies terminals 3, 4, 5 and 9.

Example

```
:_test:SGEN:PDAT1:FETC? PSUM
```

might return:

```
5.1967356791200e+012, 4.1612064440000e+009
```

```
:_test:SGEN:PDAT1:FETC? (@1:3)
```

might return:

```
3.2480134390400e+011, 1.4717679000000e+008,
3.2458887171200e+011, 1.6075325500000e+008,
3.2459149132800e+011, 1.7538254400000e+008
```

```
:_test:SGEN:PDAT1:FETC?
```

might return:

```
3.2480134390400e+011, 1.4717679000000e+008,
3.2458887171200e+011, 1.6075325500000e+008,
3.2459149132800e+011, 1.7538254400000e+008,
3.2502299556800e+011, 1.9273679100000e+008
```

NOTE If you have a finite sequence and are interested in the final result, you should query the system state before trying to fetch the error rate results. With the command :sgen:glob:syst:stat? you can check when the system has FINished the sequence generation. Then stop the system by :sgen:glob:init:cont OFF. Now it is save to fetch the error rate results.

If you have an infinite sequence or want intermediate results anyway, you can query the error results an any time. You always get the number of received bits and the number or erroneous bits. If you query soon after start, it can happen that the number of received bits is 0. Then, of course, the number of errors is also 0. Ignore these results and query again.

:ECOunt:RESet

www.valuetronics.com

Syntax :<Handle>:SGENeral:PDATa(*):ECOunt:RESet[(<ChannelList>)]

Description This command resets the “received bit counter” and the “failed bit counter” to zero. If the argument is omitted all connected terminals within a port are reset to zero. The channel list specifies a list of terminals e.g. (@1:3,5) addresses the terminals 1,2,3,5 and these counters are set to 0.

Example

```
:_test:SGEN:PDAT1:ECO:RES (@1:3,5)
```


www.valuetronics.com

Clock Mode Commands

NOTE The clock mode command is only valid for the E4862B data generator frontends.

The clock mode command is used to specify whether pulse (normal) or clock (high performance) mode is used. In clock mode, a precision clock signal is generated,with 50 % duty cycle and very low jitter (given by the clock module – the E4808A clock module should be used).

:SIGNal:MODE

This command sets the mode of the clock signal of E4862B data generator frontends. The performance mode can be either `CLOCk` (high-performance) or `PULSe` (normal).

NOTE Executing this command while the system is running causes the system to be restarted.

Syntax

:<HANDLE>**[:CGRoup*]:MODule*:CONNector*:DIGital**:STIMulus:SIGNal:MODE PULSe | CLOCk

:<HANDLE>:SGENeral:**PDATa*:DIGital:STIMulus**:SIGNal:MODE PULSe | CLOCk

:<HANDLE>:SGENeral:**PDATa*:TERMinal*:DIGital**:STIMulus:SIGNal:MODE PULSe | CLOCk

:<HANDLE>:SGENeral:**PPULse*:DIGital**:STIMulus:SIGNal:MODE PULSe | CLOCk

:<HANDLE>:SGENeral:**PPULse*:TERMinal***:DIGital:STIMulus:SIGNal:MODE PULSe | CLOCk

Parameters

PULSe Pulse mode (normal) is used.

CLOCk Clock mode (high performance) is used.

Example

```
:_test:SGEN:PDAT1:DIG:SIGN:STIM:MODE CLOC
:_test:SGEN:PDAT1:TERM1:DIG:STIM:SIGN:MODE CLOCk
:_test:SGEN:PPUL1:DIG:SIGN:MODE CLOCk
:_test:SGEN:PPUL1:TERM1:DIG:STIM:SIGN:MODE CLOCk
:_test:CGRoup1:MODule2:CONNector4:DIGital:SIGNal:MODE CLOCk
```


www.valuetronics.com

:SIGNal:MODE?

Syntax

:<HANDLE>**[:CGRoup*]:MODule*:CONNector*:DIGital**:STIMulus:SIGNal:MODE?

:<HANDLE>:SGENeral:**PDATa*:DIGital:STIMulus**:SIGNal:MODE?

:<HANDLE>:SGENeral:**PDATa*:TERMinal*:DIGital**:STIMulus:SIGNal:MODE?

:<HANDLE>:SGENeral:**PPULse*:DIGital**:STIMulus:SIGNal:MODE?

:<HANDLE>:SGENeral:**PPULse*:TERMinal***:DIGital:STIMulus:SIGNal:MODE?

Return Value

Returns the mode for the clock signal of E4862B data generator frontends: either `CLOCk` or `PULSe`.

- PULSe

  Pulse mode (normal) is used.

- CLOCk

  Clock mode (high performance) is used.

Example

```
:_test:SGEN:PDAT1:DIG:SIGN:STIM:MODE?
:_test:SGEN:PDAT1:TERM1:DIG:STIM:SIGN:MODE?
:_test:SGEN:PPUL1:DIG:SIGN:MODE?
:_test:SGEN:PPUL1:TERM1:DIG:STIM:SIGN:MODE?
:_test:CGRoup1:MODule2:CONNector4:DIGital:SIGNal:MODE?
```

might return:

```
CLOC
```


www.valuetronics.com

Format Parameter Commands

The commands for specifying the format parameters are available on port, terminal and connector level. The parameters can be specified separately for pulse and data ports.

:FORMat

Syntax

:<Handle>:SGENeral:**PDATa(*):**DIGital[:STIMulus]:SIGNal:FORMat <RZ| NRZ | R1>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):**DIGital:[STIMulus]:SIGNal:FORMat <RZ| NRZ| R1>

:<Handle>:SGENeral:**PPULse(*):**DIGital[:STIMulus]:SIGNal:FORMat <RZ| NRZ| R1>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):**DIGital:[STIMulus]:SIGNal:FORMat <RZ| NRZ| R1>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):**DIGital[:STIMulus]:SIGNal:FORMat <RZ| NRZ| R1>

Parameters

<RZ | NRZ | R1> Sets the output connector data format to either RZ, NRZ or R1.

The format of the data out stream can be specified here. Setting the Format to RZ or R1 may generate an error that the “WIDTh” is out of range. The E4862A and E4864A frontends only supports NRZ.

Example

```
:_test:SGEN:PDAT1:DIG:SIGN:FORM RZ
:_test:SGEN:PDAT1:TERM1:DIG:STIM:SIGN:FORM RZ
:_test:SGEN:PPUL1:DIG:SIGN:FORM RZ
:_test:SGEN:PPUL1:TERM1:DIG:STIM:SIGN:FORM RZ
:_test:CGROUP1:MODULE2:CONNECTOR4:DIGITAL:SIGNAL:FORMAT RZ
```


www.valuetronics.com

:FORMat?

Syntax

:<Handle>:SGENeral:**PDATa(*):**DIGital[:STIMulus]:SIGNal:FORMat?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):**DIGital:[STIMulus]:SIGNal :FORMat?

:<Handle>:SGENeral:**PPULse(*):**DIGital[:STIMulus]:SIGNal:FORMat?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):**DIGital:[STIMulus]:SIGNal :FORMat?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):**DIGital[:STIMulus]:SIGNal :FORMat?

Return Value

Returns the current data format state.

Example

```
:_test:SGEN:PDAT1:DIG:SIGN:FORM?

:_test:SGEN:PDAT1:TERM1:DIG:STIM:SIGN:format?

:_test:SGEN:PPUL1:DIG:SIGN:FORM?

:_test:SGEN:PPUL1:TERM1:DIG:STIM:SIGN:format?

:_test:CGROUP1:MODULE2:CONNECTOR4:DIGITAL:SIGNAL:FORMAT?
```

might return:

```
RZ
```


www.valuetronics.com

Segment Import and Export Language

The Segment Import and Export Tool enables the Agilent 81250 System to import and export connector trace data from/to an ASCII source file.

The syntax of the segment import and export tool enables you to create data segments in any text editor.

You can also export segments that have been set up in the graphical user interface to ASCII files and modify these files as required.

The following details are described:

www.valuetronics.com

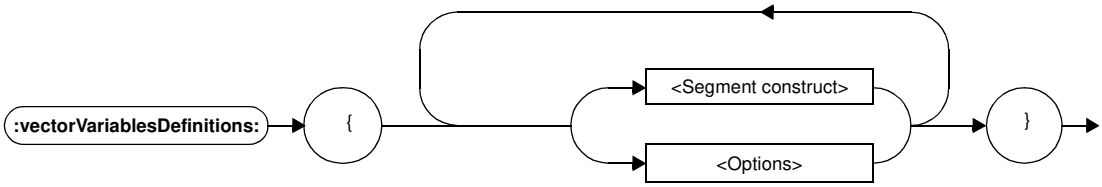
The Language Syntax

Syntax flow diagrams (also known as railroad diagrams) are used to provide pictorial representations of syntax.

Vector Variable Construct

The vector import language is contained in an ASCII text file and contains one or more **Vector Variable constructs**. A Vector Variable construct is the highest level construct that may be specified and is itself built from lower level constructs.

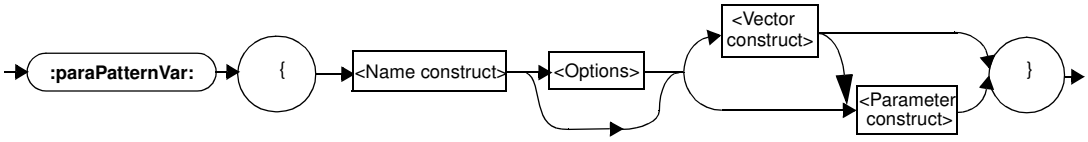
The format of the **Vector Variable construct** is shown below:



<Options> contains 1 or more default parameters that may be overridden as necessary. There are two scopes for options, **vector variable scope and pattern scope**. The options constructs will be described in more detail in *"Options" on page 303*.

Segment Construct

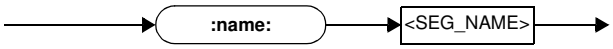
The **Segment construct** contains **segment** information. A segment is a unit of information acceptable to the Agilent 81250 System and may contain either connector trace data or parameters necessary for the operation of the instrument. The following syntax diagram shows the composition of a segment construct:



www.valuetronics.com

Name Construct

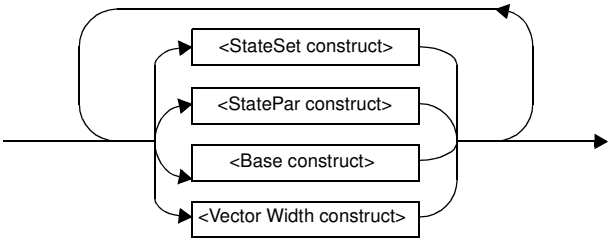
The **Name construct** specifies a name for the segment:



where <SEG_NAME> is an ASCII string representing the segment name. The segment name *must* begin with a letter, which may be followed by an alphanumeric string. Underscores are allowed, whitespaces are not.

Options

The **Options** syntax is defined in the following diagram:



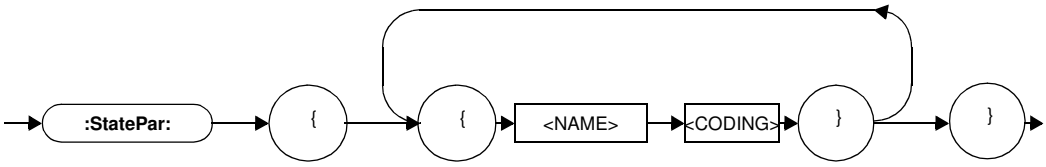
Options defined within a vector variable scope remain valid for the duration of the vector variable construct. However, options may also be specified within a segment construct in which case they are valid only for the duration of the segment construct. As soon as the segment construct goes out of scope the options valid at the vector variable scope re-apply (see *"Scopes" on page 308* for a more detailed definition of scopes).

www.valuetronics.com

StatePar Construct

The **StatePar** construct allows specification of one or more **state sets**. A state set is the specification of waveform characters that may legally be used within the **:vectors:** statement.

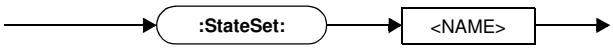
A state set consists of a state set name and its corresponding coding. The state set name is referred to by the StatePar construct:



where <NAME> is an ASCII string representing the name of the coding, and <CODING> is an ASCII string representing the coding. Two state sets are supplied as default: **{D “01”}** and **{R “0 x1”}**, see *“Coding” on page 306*

StateSet Construct

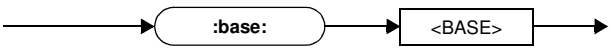
The **StateSet** construct selects a coding previously defined by the StatePar command. The name supplied as an argument indicates which coding should be used:



where <NAME> is an ASCII string representing the name of the coding. The default state set is **D**.

Base Construct

The **Base** construct selects the nature of the strings representing the input vectors. There are three possibilities: **hex (h)**, **decimal (d)** or **waveform (w)**.



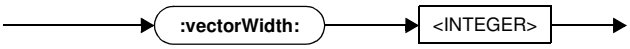
<BASE> is one of **w**, **h**, or **d**. The default base is **w**.

www.valuetronics.com

Vector Width Construct

The **Vector width construct** allows the specification of the width in states of each vector. Should the number of states supplied be less than that specified in the **:vectorWidth:** command, then the vector is padded out on its left side using the left most supplied state. Should the supplied vector be too long then it is clipped on the left side until it contains exactly the necessary number of states.

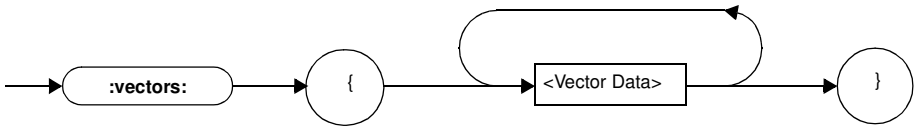
If no vector width statement is present, then the length of the first supplied vector is used as the width for all subsequent vectors:



where <INTEGER> is an integer value between 1 and 1024.

Vector Construct

The **Vector construct** specifies the content of connector trace data:

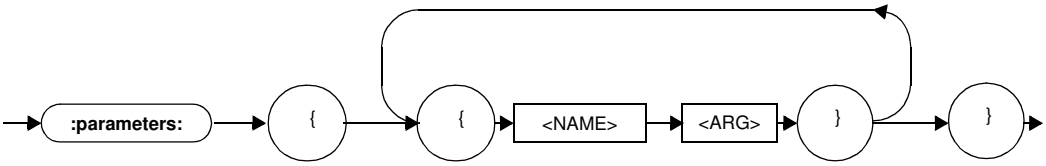


<Vector Data> is one or more strings of hex, decimal or waveform characters. Each string is delimited by a new line. The waveform characters are specified by the currently active state set. The vector format (i.e. hex, decimal or waveform) is specified by the **:base:** command described in *"Base Construct" on page 304*.

www.valuetronics.com

Parameter Construct

The counterpart to the vector construct is the **Parameter construct**. This construct enables the input of supplementary segment information:



where <NAME> is an ASCII string representing the name of the argument and <ARG> is the argument type. Each argument pair is enclosed within brackets. The range of possible parameters is described more fully in the section *"Parameter Segments" on page 311* below.

Concepts

This section describes some of the more important concepts of the vector import language.

Coding

In this section, the coding mechanism is described in more detail.

The goal of the Vector Import tool is to be able to load one or more sequences of binary data to the Agilent 81250 System. The binary data contains waveforms to be presented to the connectors of a hardware device. However, the use of pure binary is difficult for humans to interpret and therefore for programming purposes an alternative representation is necessary.

Firstly, we demonstrate the decoding of a vector that is in waveform characters.

www.valuetronics.com

Using Base w

Consider the following portion of import language code:

```
:statePar: { {B "0 x1")   #define state set B to have coding "0 x1"
:stateSet: B              #select State set B
:base: w                  #use waveform characters
:vectorWidth: 5           #vector is 5 states wide
:vectors:
{
 001xx                    #this is the vector
}
```

Let us examine the *:statePar:* statement. A state set name B is associated with the coding “0 x1”. The order of the characters “0 x1” is particularly important as it implies the underlying binary values. Starting with 0, the coding assumes increasing order from left to right.

Each of the waveform characters represents a state. To represent three waveform characters, two bits are required. With two bits there are 4 binary values available. The value 01 must be represented by ‘blank’ because this is not a valid waveform character. Therefore, the binary values of the states are: 0 = 00, ‘blank’= 01, x = 10 and 1 = 11. Enough information is now available to decode the vector: 001xx = 00 00 11 10 10

Coding	Memory Representation	Hardware Representation
0	00	0
‘blank’	n/ a	n/ a
x	10	Don’t care
1	11	1

Coding	Memory Representation	Hardware Representation
0	0	0
1	1	1

Here is the same example using a hexadecimal base and the coding “0 x1”:

www.valuetronics.com

Using Base h (hexadecimal)

```
:statePar: { { B "0 x1"}  #define state set B to have coding "0 x1"
:stateSet: B              #select state set B
:base: h                  #use hex characters
:vectorWidth: 5           #vector is 5 states wide
:vectors:
{
 03a                      #this is the vector
}
```

Each hexadecimal character needs 4 bits and so in the above example the vector is 12 bits wide. As only 10 bits are necessary to represent 5 vectors, the leftmost 2 bits are ignored:

0000 0011 1010

Finally, the same example using base d (decimal):

Using Base d

```
:statePar: { { B "0 x1"}  #define state set B to have coding "0 x1"
:stateSet: B              #select state set B
:base: d                  #use decimal characters
:vectorWidth: 5           #vector is 5 states wide
:vectors:
{
 58                       #this is the vector
}
```

The decimal value 58 is decoded as a 32 bit unsigned integer:

00000000 00000000 00000**00 00111010**

the 10 rightmost bits being the vector.

When using hexadecimal and decimal bases, decoding still requires a valid state set to know the bit width of each state.

Scopes

Scopes are important to the lifetime of optional variables. There are 2 scopes, vector variable scope and parameter scope. In the following diagram, the `:base:` parameter is used to illustrate scope. At the start of the program fragment, base w is defined (although base w is default we define it for demonstration purposes). The first grey shaded area shows the scope of the base w command. Waveform characters are thus used in SegA. In SegB a new base h is specified. The white box shows the scope of the base h command.

www.valuetronics.com

As the base h command was defined with parameter scope, as soon as the definition for SegB ends, so does the scope of the base h command. The base reverts to that of the previously defined base w.

Then, still within vector variable scope, base d is defined. This remains the base until the end of the vector variables definition.

```
:vectorVariablesDefinitions:              #start of vector variable scope
{
   :base: w
   :paraPatternVar:
   {
      :name: SegA
      :vectors:
      {
         1101110110101
      }
   }

   :paraPatternVar:
   {
      :name: SegB
      :base: h                            # start of pattern scope
      :vectors:                           # base h is valid
      {                                   #
         f7df                             #
      }                                   #
   }                                      #
                                          # until here


# vector variable scope
# base w is valid

   :base: d                               # base d supersedes base w
   :paraPatternVar:
   {
      :name: SegC
      :vectors:
      {
         243
      }
   }
}                                         # end of vector variable scope
```


www.valuetronics.com

Vector Padding and Clipping

The vector import tool ensures that all vectors input from the source file have the correct length. All following padding and clipping examples are shown in base w; i.e. waveform characters. Padding and clipping in hex or decimal bases behave as though the representation were first converted to waveform characters.

Padding

A vector that is shorter than necessary must be padded to the correct length. For example if a vector width of 10 states is being used, the Vector Import Tool would expand 01x to 000000001x. The Import Tool knowing that 7 states were missing, takes the leftmost state, in this case '0' and fills out the vector on the left side until it is of the correct length. Padding can be useful when the same vector cluster is repeated many times. Consider the following example:

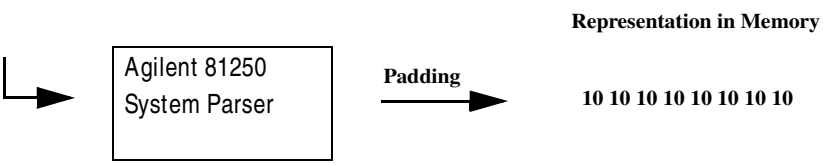
Without Padding

```
:statePar: { {B "0 x1"} } #define state set B to have coding "0 x1"
:stateSet: B              #select state set B
:base: w                  #use waveform characters
:vectorWidth: 8           #vector is 8 states wide
:vectors
{
xxxxxxxx
}
```

It would be possible to rewrite this code snippet thus:

With Padding

```
:statePar: { {B "0 x1"} }  #define state set B to have coding
"0 x1"
:stateSet: B               #select state set B
:base: w                   #use waveform characters
:vectorWidth: 8            #vector is 8 states wide
:vectors
{
x
}
```



The Vector Import Tool recognizes that 7 states are missing, and uses the left-most (and only) state to pad out the vector. A reduction in disk storage is possible in vector files with a high degree of repetition.

www.valuetronics.com

Clipping

A supplied vector that is too long will be clipped on its left side until it is of the correct length. For example, using a vector width of 10, xxxx000000001x would be clipped to 000000001x.

Parameter Segments

Parameter segments allow the input of parameter variables into the Agilent 81250 System. The data part of a parameter segment consists of a string. The string consists of one or more name-parameter pairs enclosed by parentheses. The string and the name-parameter pairs compare to the rules of SCPI expressions.

There are three data segment types.

- Memory segments
- PRBS segments
- PRWS segments
- SFI-5 segments

Predefined Parameter Names

For the three segment types the following predefined parameter names and values are valid:

Predefined Parameter Names	Memory Segments	PRBS Segments	PRWS Segments	SFI-5 Segments
_Type	(MEMORY)	(PRBS)	(PRWS)	(SFI5)
_Rotating	not applicable [n/a]	( )	( )	( )
_Polynom	n/a	(“2^5-1”)...(“2^15-1”)[a]	(“2^5-1”)...(“2^15-1”)[a]	("2^7-1"), ("2^11-1"), ("2^15-1"),("2^23-1"), ("2^31-1")
_Logic	n/a	(NORMAL) or (INVERTED)	(NORMAL) or (INVERTED)	(NORMAL) or (INVERTED)
_PRxSType	n/a	(PURE) or (ERRORED) or (DENSITY) or (EXTENDED)	(PURE) or (ERRORED) or (DENSITY) or (EXTENDED)	(PURE)
_FramingBytes	n/a	n/a	n/a	32 bit binary-string enclosed in ’()’
_ExpansionHeader	n/a	n/a	n/a	32 bit binary-string enclosed in ’()’
_Errors [when (ERRORED) is selected]	n/a	(0)...(max length of selected PRxS)	(0)...(max length of selected PRxS)	n/a

www.valuetronics.com

Predefined Parameter Names	Memory Segments	PRBS Segments	PRWS Segments	SFI-5 Segments
_Density [when (DENSITY) is selected]	n/a	(“1/8”) or (“1/4”) or (“1/2”) or (“3/4”) or (“7/8”)	(“1/8”) or (“1/4”) or (“1/2”) or (“3/4”) or (“7/8”)	n/a
_Which [when (EXTENDED) is selected]	n/a	(ZERO) or (ONE)	(ZERO) or (ONE)	n/a
_Number [when (EXTENDED) is selected]	n/a	(0)...(max length of selected PRxS)	(0)...(max length of selected PRxS)	n/a

[a] PRBS and PRWS both also support 2^23 and 2^31. These polynomials must be PURE, because they do not fit into the RAM.

Default Segment for SFI-5

There is a default segment for SFI-5, which has the following settings:

Parameter	Value
_Type	(SFI5)
_Rotating	( )
_Polynom	("2^15-1")
_Logic	(NORMAL) or (INVERTED)
_PRxSType	(PURE)
_FramingBytes	("11110110111101100010100000101000")
_ExpansionHeader	("10101010101010101010101010101010")

www.valuetronics.com

Default Settings

The Vector Import tool provides default settings for the following parameters:

Parameter	Default Value
:statePar:	{D 01} / R "0 x1"}
:stateSet:	D
:base:	w
:vectorWidth:	length of first vector

The default values are valid in both Vector variable and parameter scopes.

Examples

The following portion of vector import code illustrates the use of a parameter segment:

- *"Example: Memory Type Segment" on page 314* shows a data pattern segment, a memory type segment.
- *"Example: PRBS Type Segment" on page 314* shows a pure 2^8-1 PRBS with normal output mode, a PRBS segment type.
- *"Example: PRWS Type Segment" on page 315* shows a 2^{10}-1 PRWS with 10 errors inserted and with inverted output mode.
- *"Example: SFI5 Type Segment" on page 315* shows a 2^{15}-1 SFI-5 segment.

www.valuetronics.com

Example: M emory Type Segment

```
:vectorVariablesDefinitions:
{
   :paraPatternVar:
   {
      :name: InitData
      :statePart: { {A 01} }
      :stateSet: A
      :vectorWidth: 4
      :vectors:
      {
         1111
         0000
         1111
         0000
         1111
         0000
         1111
         0000
         1111
         0000
         1111
         0000
      }
      :parameters:              #Predefined parameter names
      {                         #start with '_'
         {_Type (MEMORY)}       #parameter values
      }                         #are always
   }                            #enclosed in '()'
}
```

Example: PRBS Type Segment

```
:vectorVariablesDefinitions:
{
   :paraPatternVar:
   {
      :name: prw
      :parameters:
      {
         {_Type (PRWS) }            #Predefined parameter names
         {_Rotating () }            #start with '_'
         {_PRxSType (ERRORED) }     #parameter values
         {_Polynom ("2^10-1") }     #are always
         {_Logic (INVERTED) }       #enclosed in '()'
      }
   }
}
```


www.valuetronics.com

Example: PRWS Type Segment

```
:vectorVariablesDefinitions:
{
   :paraPatternVar:
   {
      :name: payload
      :parameters:
      {
         {_Type (PRBS) }           #Predefined parameter names
         {_Rotating () }           #start with '_'
         {_PRxSType (PURE) }       #parameter values
         {_Polynom ("2^8-1") }     #are always
         {_Logic (NORMAL) }        #enclosed in '()'
      }
   }
}
```

Example: SFI5 Type Segment

```
:vectorVariablesDefinitions:
{
   :paraPatternVar:
   {
      :name: sfi5_example
      :parameters:
      {
                                #Predefined parameter names
                                #start with '_'
                                #parameter values
                                #are always
                                #enclosed in '()'
         {_Type (SFI5) }
         {_FramingBytes ("11110110111101100010100000101000") }
         {_ExpansionHeader ("10101010101010101010100000000000") }
         {_Rotating () }
         {_PRxSType (PURE) }
         {_Logic (NORMAL) }
         {_Polynom ("2^15-1") }
      }
   }
}
```


www.valuetronics.com

www.valuetronics.com

Example Code

NOTE Instead of using the SCPI commands, we recommend to use the plug & play programming interface. For an online help on the VXIPNP functions, refer to `hp81200.hlp` provided with the plug & play functions.

The following is an example of how to use plug & play functions to simplify programming:

- *"Main.cpp Application Code Using VXI Plug&Play" on page 318*

The following provide the complete code for the *"Example C++ Program" on page 38*:

- *"Lib.cpp Interface Class Library Code" on page 331*
- *"Main.cpp Application Code" on page 336*

www.valuetronics.com

Main.cpp Application Code Using VXI Plug&Play

The following is an example of implementing plug & play functionality. It is recommended that you use the plug & play functionality to program the system.

```
// main.cpp
// This is a small demo program showing how to use
// the remote interface of the 81200A user software
//
// It demonstrates how to:
// - connect to a firmware server
// - set up ports and terminals,
// - apply levels / thresholds
// - import data segment
// - set up a sequence
// - set measurement mode
// - start measurement
// - find out when measurement is done
// - stop measurement
// - get captured data
// - export captured data to a file
//
// To simplify code the VXI Plug&Play interface is used.
//
// Any error that occurs is logged to stdout or a file.


#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "hp81200.h"

#define SYSTEM_NAME1 "DSRA"

#define SETTING_NAME "prog_sample"

static ViSession  hInstrument;
static FILE* fp;
```


www.valuetronics.com

```
static bool errHandling(ViStatus lStatus, const char *szFctName)
{
   ViChar szErrMsg[hp81200_DEFAULT_BUFFER_SIZE];
   ViInt32 lErrCode;


   if (lStatus != VI_SUCCESS)
   {
      if (lStatus == hp81200_INSTR_ERROR_DETECTED)
      {
      do
         {
         if (hp81200_error_query(hInstrument, &lErrCode, szErrMsg)
         != VI_SUCCESS)
         {
            fprintf(fp, "Instrument error: FctName<%s>
            ErrCode<undetermined> "\
            "ErrMsg<undetermined>\n", szFctName);
            break;
         }
         else if (lErrCode != 0)
         fprintf(fp, "Instrument error: FctName<%s> ErrCode<%d>
         ErrMsg<%s>\n",
         szFctName, lErrCode, szErrMsg);
         } while(lErrCode != 0);
      }
      else
      {
         // Not an instrument error


         if (hp81200_error_message(hInstrument, lStatus, szErrMsg)
         != VI_SUCCESS)
         fprintf(fp, "Error: FctName<%s> ErrMsg<undetermined>\n",
         szFctName);
         else
         fprintf(fp, "Error: FctName<%s> ErrMsg<%s>\n", szFctName,
         szErrMsg);
      }
   }
   return (lStatus == VI_SUCCESS);
}
```


www.valuetronics.com

```
static bool doIt()
{
   ViChar szParam[hp81200_DEFAULT_BUFFER_SIZE];
   ViChar szAnalyzers[hp81200_DEFAULT_BUFFER_SIZE] = {0};
   ViChar szGenerators[hp81200_DEFAULT_BUFFER_SIZE] = {0};



   ViInt32 lAnalyzerCount = 0;
   ViInt32 lGeneratorCount = 0;
   ViInt32 lAnalyzerPortNumber = 0;
   ViInt32 lGeneratorPortNumber = 0;

   ViStatus lStatus = 0;


   // ------------------------------------------------------------

   // stop system
   // reset system

   // ------------------------------------------------------------

   lStatus = hp81200_systemStop(hInstrument);
   errHandling(lStatus, "hp81200_systemStop");

   lStatus = hp81200_settingNew(hInstrument);
   errHandling(lStatus, "hp81200_settingNew");


   // ------------------------------------------------------------

   // First, we want to generate a list of all analyzers and
   // all generators, so that afterwards we can connect to ports

   // ------------------------------------------------------------


   // Determine number of clock groups

   ViInt32 lClkGeneratorCount;

   lStatus = hp81200_nrOfClockGroups_Q(hInstrument,
   &lClkGeneratorCount);

   errHandling(lStatus, "hp81200_nrOfClockGroups_Q");

   // Loop over all clock groups

   for (ViInt32 c = 1; c <= lClkGeneratorCount; c++)
   {

      ViInt32 lModuleCount;
```


www.valuetronics.com

```
// Determine number of modules within this clock group
lStatus = hp81200_nrOfModules_Q(hInstrument, c,
&lModuleCount);
errHandling(lStatus, "hp81200_nrOfModules_Q");

// Loop over all modules

for (ViInt32 m = 1; m <= lModuleCount; m++)
{

   ViInt32 lConnectorCount;
   lStatus = hp81200_nrOfConnectors_Q(hInstrument, c, m,
   &lConnectorCount);

   errHandling(lStatus, "hp81200_nrOfConnectors_Q");

   // Loop over all connectors

   for (ViInt32 co = 1; co <= lConnectorCount; co++)
   {

      ViChar szConnectorType[16];
      lStatus = hp81200_typeOfConnector_Q(hInstrument, c, m,
      co,szConnectorType);

      errHandling(lStatus, "hp81200_typeOfConnector_Q");

      // Put in analyzer or generator list

      sprintf(szParam, "%02d%02d%03d", c, m, co);
      if (strcmp(szConnectorType, "ANALYZER") == 0)
      {

         // Analyzer

         if (strcmp(szAnalyzers, "") != 0)
         strcat(szAnalyzers, ",");
         strcat(szAnalyzers, szParam);
         lAnalyzerCount ++;

      }

      else

      {

         // Generator

         if (strcmp(szGenerators, "") != 0)
         strcat(szGenerators, ",");
         strcat(szGenerators, szParam);
         lGeneratorCount++;

      }

   }

}
}
```


www.valuetronics.com

```
// ----------------------------------------------------
// Now connect analyzers and generators to one port each
// ----------------------------------------------------

if (lGeneratorCount > 0)
{

   lStatus = hp81200_portCreation(hInstrument, "DATA",
   "INPUT_PORT", lGeneratorCount, "input");

   errHandling(lStatus, "hp81200_portCreation");
   lGeneratorPortNumber = 1;
   sprintf(szParam, "(@%s)", szGenerators);
   lStatus = hp81200_terminalConnect(hInstrument, "DATA",
   lGeneratorPortNumber,1, szParam);

   errHandling(lStatus, "hp81200_terminalConnect");

}


if (lAnalyzerCount > 0)
{

   lStatus = hp81200_portCreation(hInstrument, "DATA",
   "OUTPUT_PORT", lAnalyzerCount, "output");

   errHandling(lStatus, "hp81200_portCreation");

   lAnalyzerPortNumber = lGeneratorPortNumber + 1;
   sprintf(szParam, "(@%s)", szAnalyzers);

   lStatus = hp81200_terminalConnect(hInstrument, "DATA",
   lAnalyzerPortNumber, 1, szParam);

   errHandling(lStatus, "hp81200_terminalConnect");

}


// ----------------------------------------------
// Switch on everything, apply levels / thresholds
// ----------------------------------------------

if (lGeneratorPortNumber > 0)
{

   lStatus = hp81200_portOutputState(hInstrument, "DATA",
   lGeneratorPortNumber, "ON");

   errHandling(lStatus, "hp81200_portOutputState");

   lStatus = hp81200_portHighLowLevel(hInstrument, "DATA",
   lGeneratorPortNumber, 2, 0, 0, 50);

   errHandling(lStatus, "hp81200_portHighLowLevel");

}
```


www.valuetronics.com

```
if (lAnalyzerPortNumber > 0)
{
   lStatus = hp81200_portInputState(hInstrument,
   lAnalyzerPortNumber, "ON");
   errHandling(lStatus, "hp81200_portInputState");
   lStatus = hp81200_portInputLevel(hInstrument,
   lAnalyzerPortNumber, 0, 1, 50, 0);
   errHandling(lStatus, "hp81200_portInputLevel");
}


// ------------------------------------------------
// Import data (overwrite mode)
// ------------------------------------------------
sprintf(szParam, "%s\\samples\\segments\\walk64.txt",
getenv("DVTDSRBASEDIR"));
lStatus = hp81200_segmentLoadFromFile(hInstrument, szParam,
"ON", "");
errHandling(lStatus, "hp81200_segmentLoadFromFile");


// ------------------------------------------------
// Set period
// ------------------------------------------------
hp81200_systemPeriod(hInstrument, 1E-6);
errHandling(lStatus, "hp81200_systemPeriod");


// ------------------------------------------------
// Assign a sequence that uses imported segment
// ------------------------------------------------


// Determine number of loop levels available
ViInt32 lLoopLevelCount;
lStatus = hp81200_sequenceAvailableLoopLevel_Q(hInstrument,
&lLoopLevelCount);
errHandling(lStatus, "hp81200_sequenceAvailableLoopLevel_Q");
```


www.valuetronics.com

```
// Make infinite loop with trigger using the imported demo
// segment

sprintf(szParam, "(1.0,'',(LOOP%d,1,INF,(BLOCK,0,64",
lLoopLevelCount);
if (lGeneratorPortNumber > 0)
sprintf(szParam, "%s,'walking64',0,0", szParam);


if (lAnalyzerPortNumber > 0)
sprintf(szParam, "%s,'walking64',0,0", szParam);
sprintf(szParam, "%s)))", szParam);
lStatus = hp81200_sequence(hInstrument, szParam);
errHandling(lStatus, "hp81200_sequence");


// Generate trigger signal from sequence

lStatus = hp81200_systemTriggerOutSource(hInstrument, 1, "SEQ");
errHandling(lStatus, "hp81200_systemTriggerOutSource");


// ------------------------------------------------
// Set measurement mode
// Compare and acquire about error
// ------------------------------------------------


if (lAnalyzerPortNumber > 0)
{

   lStatus = hp81200_analyzerErrorCapture(hInstrument, 32768);
   errHandling(lStatus, "hp81200_analyzerErrorCapture");

}


// ------------------------------------------------
// Save setting for later use
// ------------------------------------------------

lStatus = hp81200_settingSave(hInstrument, SETTING_NAME);
errHandling(lStatus, "hp81200_settingSave");


// ------------------------------------------------
// Start measurement
// ------------------------------------------------

lStatus = hp81200_systemStart(hInstrument);
errHandling(lStatus, "hp81200_systemStart");
```


www.valuetronics.com

```
// ------------------------------------------------
// Poll measurement done
// THIS ONLY WORKS ONLINE!!!
// ------------------------------------------------

   while (true)
   {

      ViChar szExpState[16];

      lStatus = hp81200_systemExpandedState_Q(hInstrument,
      szExpState);

      errHandling(lStatus, "hp81200_systemExpandedState_Q");

      if (strncmp(szExpState, "FIN", 3) == 0)
      break;

   }


   // ------------------------------------------------
   // Stop measurement
   // ------------------------------------------------

   lStatus = hp81200_systemStop(hInstrument);
   errHandling(lStatus, "hp81200_systemStop");


   // ------------------------------------------------
   // Demonstrate how to deal with captured data
   // Export the captured data and the error memory
   // ------------------------------------------------


   if (lAnalyzerPortNumber > 0)
   {

      sprintf(szParam, "Analyzer\\Capture.%d",
      lAnalyzerPortNumber);

      ViInt32 lCaptureInspector;

      lStatus = hp81200_segmentOpen_Q(hInstrument, szParam,
      &lCaptureInspector);

      errHandling(lStatus, "hp81200_segmentOpen_Q");

      sprintf(szParam, "Analyzer\\ErrMem.%d", lAnalyzerPortNumber);

      ViInt32 lErrMemInspector;

      lStatus = hp81200_segmentOpen_Q(hInstrument, szParam,
      &lErrMemInspector);

      errHandling(lStatus, "hp81200_segmentOpen_Q");
```


www.valuetronics.com

```
// Get pattern width (number of traces)
ViInt32 lWidth;

lStatus = hp81200_segmentPatternWidth_Q(hInstrument,
lCaptureInspector,&lWidth);

errHandling(lStatus, "hp81200_PatternWidth_Q");


// Get pattern length (number of vectors)

ViInt32 lLength;


lStatus = hp81200_segmentPatternLength_Q(hInstrument,
lCaptureInspector, &lLength);

errHandling(lStatus, "hp81200_PatternLength_Q");


// Get coding

ViChar szPatternCode[3]; // Possible coding here is only "01"

lStatus = hp81200_segmentPatternCoding_Q(hInstrument,
lCaptureInspector, szPatternCode);

errHandling(lStatus, "hp81200_PatternCoding_Q");


// Write information we got so far:

fprintf(fp, "Analyzer.%d: Coding <%s>, Width %d, Length
%d\n",lAnalyzerPortNumber, szPatternCode, lWidth, lLength);


// Get some data, but assure it does not overflow our small
result-buffer

// To make it human readable, data is returned as a hex
string

if (lLength > 10)
lLength = 10;

if (lLength > 0)
{

// Calculate necessary buffer size
// NOTE: Traces are aligned on byte boundaries
//       Each character (hex digit) uses 4 vectors (with "01"
//       coding)

ViInt32 lHexDigitCount = lLength/4;

if (lLength%4)

// Remaining vectors require another hex digit

lHexDigitCount++;
```


www.valuetronics.com

```
// Check if byte-aligned

if (lHexDigitCount%2)

// Byte-align

lHexDigitCount++;

// Count is required for each trace

lHexDigitCount *= lWidth;

ViInt32 lBufferLength = lHexDigitCount+2+1;
ViPChar lpszPattern = (ViPChar)malloc(lBufferLength);

if (lpszPattern == NULL)
{

   fprintf(fp, "Insufficient RAM - run aborted\n" );
   return false;

}

// Take a sample (in hex)

lStatus = hp81200_segmentPatternData_Q(hInstrument,
lCaptureInspector, 0, 0, lWidth-1, lLength-1, lBufferLength,
lpszPattern);

errHandling(lStatus, "hp81200_PatternData_Q");

fprintf(fp, "Analyzer.%d: %s\n", lAnalyzerPortNumber,
lpszPattern);

// Convert data from Hex format to Binary format
// NOTE: Bits within a trace are "from left to right",
//       i.e. first bit is bit 7, then bit 6 and so on
//       for a 1 bit coding

// Display trace-wise

// Divide data into traces

ViInt32 lHexDigitsPerTrace = lHexDigitCount/lWidth;

// Process each trace

for (ViInt32 t = 0; t < lWidth; t++)
{

   fprintf(fp, "Trace %d: ", t);
```


www.valuetronics.com

```
            // Process each hex digit in trace
            for (ViInt32 l = 0; l < lHexDigitsPerTrace; l++)
            {
               int nHexDigit;
               ViInt32 lHexDigitIndex = l + (t*lHexDigitsPerTrace);

               if ((*(lpszPattern+lHexDigitIndex) >= '0') &&
                  (*(lpszPattern+lHexDigitIndex) <= '9'))
                  nHexDigit = *(lpszPattern+lHexDigitIndex) - '0';

               else if ((*(lpszPattern+lHexDigitIndex) >= 'A') &&
                           (*(lpszPattern+lHexDigitIndex) <= 'F'))
                  nHexDigit = *(lpszPattern+lHexDigitIndex) - 'A' +
                  10;

               else

               // Illegal digit - ignore

               nHexDigit = 0;

               // Process each bit (vector) in hex digit

               for (int v = 3; v >= 0; v--)
               {

                  if (((l*4) + (4-v)) > lLength)
                  // Ignore padding required for byte-alignment
                  // break;

                  if (nHexDigit / (1 << v))
                  {

                     // Bit is set
                     fprintf(fp,  "1");

                     // Set bit has been treated => remove it
                     nHexDigit -= (1 << v);

                  }

                  else

                  // Bit is not set
                  fprintf(fp,  "0");
               }
            }
            fprintf(fp, "\n");
         }


         // Pattern buffer can now be discarded
         free(lpszPattern);

      }
```


www.valuetronics.com

```
    // Save segments under a different name to the local setting
    lStatus = hp81200_segmentSave(hInstrument, lCaptureInspector,
    "LocalSegments\\prog_sample_capture");
    errHandling(lStatus, "hp81200_segmentSave");


    lStatus = hp81200_segmentSave(hInstrument, lErrMemInspector,
    "LocalSegments\\prog_sample_errmem");
    errHandling(lStatus, "hp81200_segmentSave");


    // Export segments
    // This is down now before the segments are closed, because
    // they are now still in memory

    lStatus = hp81200_segmentSaveToFile(hInstrument,
    "c:\\temp\\prog_sample_capture.txt", SETTING_NAME,
    "prog_sample_capture");
    errHandling(lStatus, "hp81200_segmentSaveToFile");

    lStatus = hp81200_segmentSaveToFile(hInstrument,
    "c:\\temp\\prog_sample_errmem.txt", SETTING_NAME,
    "prog_sample_errmem");
    errHandling(lStatus, "hp81200_segmentSaveToFile");


    // Close the segments!
    lStatus = hp81200_segmentClose(hInstrument, lCaptureInspector);
    errHandling(lStatus, "hp81200_segmentClose");


    lStatus = hp81200_segmentClose(hInstrument, lErrMemInspector);
    errHandling(lStatus, "hp81200_segmentClose");
    }


    return true;
}
int main()
{
    ViStatus lStatus = 0;
    ViBoolean bIDQuery = VI_FALSE;
    ViBoolean bResetDevice = VI_FALSE;
    ViString szSystemNameHp81200 = SYSTEM_NAME1;
    ViString szApplicationName = "DSR";
```


www.valuetronics.com

```
    // Errors are logged to a file (which may be stdout).
    //fp = fopen("c:\\temp\\sample_4.txt", "w");
    fp = stdout;


    // Initialize the software connection to the 81200 system
    lStatus = hp81200_init("VXI0::0::INSTR", bIDQuery, bResetDevice,
    &hInstrument );
    errHandling(lStatus, "hp81200_init");


    // For remote access
    lStatus = hp81200_connect(hInstrument, "fox00099","2203");
    errHandling(lStatus, "hp81200_connect");
    lStatus = hp81200_systemSelect(hInstrument, szSystemNameHp81200,
    szApplicationName);
    errHandling(lStatus, "hp81200_systemSelect");
    fprintf(fp, "System %s:\n", szSystemNameHp81200);
    doIt();

    hp81200_close(hInstrument);

    // Wait so that user can see the results
    if ((fp == stdout) || (fp == stderr))
    {
       printf("Press any key to continue\n");
       (void)getchar();
    }
    else
    fclose(fp);

    return (lStatus != VI_SUCCESS);
    // (TRUE) 1 => failure, (FALSE) 0 => success
}
```


www.valuetronics.com

Lib.cpp Interface Class Library Code

The following code provides the library functions for initializing the system, dealing with handles, executing the SCPI commands, and logging errors.

```
//
// small interface class to the 81200A
//
// provides error logging, handle-handling
//
// This example is limited in functionality, because there
// are fixed command and response buffers.
//

#include <ctype.h>
#include <string.h>
#include <hp81200.h>


#include "lib.h"


// init
HP81200::HP81200()
: itsErrorFile(0), itsConnected(false)
{
   strcpy(itsHandle, "");
   strcpy(itsResultBuffer, "");
}


// make sure we released handle and disconnected from the system
HP81200::~HP81200()
{
   // assure that handle is destroyed...
   if (strcmp(itsHandle, "") != 0)
      (void)Exit();
}


// Connect to 81200 server on local machine or "theServerName".
// Creates a handle for the system "theSystemName".
// This handle is internally handled, so that we do not need
// to specify it with each command.
bool HP81200::Init(  const char* theServerName,
                     const char* theHandleSuggestion,
                     const char* theSystemName,
                     FILE* theErrorFile)
```


www.valuetronics.com

```
{
   bool b;
   char aCmd[128];
   int ret;

   // remember error log file
   itsErrorFile = theErrorFile;

   // connect to firmware server
   // if already connected do nothing
   ret = Connect_HP81200(theServerName);
   itsConnected = (ret == 0);
   if (!itsConnected && (ret != -4))
   {
      WriteErrorLog(ret);
      return false;
   }

   // create a handle
   sprintf(aCmd, ":dvt:inst:hand:cre? %s,'DSR','%s'",
      theHandleSuggestion, theSystemName);
   b = Call(aCmd);

   strcpy(itsHandle, itsResultBuffer);

   return b;
}

// Release the handle, disconnect from the system
bool HP81200::Exit()
{
   if (strcmp(itsHandle, "") != 0)
   {
      bool b;
      char aCmd[128];

      // destroy handle
      sprintf(aCmd, ":dvt:inst:hand:dest %s", itsHandle);
      b = Call(aCmd);
      if (!b) return b;

      // clear remembered handle
      strcpy(itsHandle, "");
   }

   if (itsConnected)
   {
      int ret;

      ret = Disconnect_HP81200();
      if (ret != 0)
      {
```


www.valuetronics.com

```
            WriteErrorLog(ret);
            return false;
         }
      }

      return true;
}

// Call 81200, handle is automatically supplied by the class.
// Errors are logged to the logfile.
// In case of an error, false is returned.
// Results from the call are ignored.
bool HP81200::Call(const char* theCmd)
{
      int ret;
      char aCmd[1024];
      int aResultLen;

      if (strcmp(GetHandle(theCmd), "dvt") == 0)
         // already valid command with handle dvt, just use it...
         sprintf(aCmd, "%s", theCmd);
      else if (theCmd[0] == ':')
         // command for local handle
         sprintf(aCmd, ":%s%s", itsHandle, theCmd);
      else
         // command for local handle
         sprintf(aCmd, ":%s:%s", itsHandle, theCmd);

      aResultLen = sizeof(itsResultBuffer);
      ret = Call_HP81200(aCmd, itsResultBuffer, &aResultLen);
      if (ret != 0)
      {
         WriteErrorLog(ret, theCmd);
         return false;
      }

      return true;
}

// Call 81200, handle is automatically supplied by the class.
// Errors are logged to the logfile.
// In case of an error, false is returned.
// Results are returned as a pointer to an internal buffer,
// so before calling Call again, the results must be otherwise
// saved.
bool HP81200::Call(const char* theCmd, const char*& theResult)
{
      theResult = itsResultBuffer;

      return Call(theCmd);
}
```


www.valuetronics.com

```
// extract handle part of a command string (private method)
const char* HP81200::GetHandle(const char* theCmd)
{
   if (theCmd == 0)
      return itsHandle;
   else if ((theCmd[0] == ':') &&
      (tolower(theCmd[1]) == 'd') &&
      (tolower(theCmd[2]) == 'v') &&
      (tolower(theCmd[3]) == 't'))
      // already valid command vor dvt-handle
      return "dvt";
   else if ((tolower(theCmd[0]) == 'd') &&
      (tolower(theCmd[1]) == 'v') &&
      (tolower(theCmd[2]) == 't'))
      // already valid command vor dvt-handle
      return "dvt";
   else if (theCmd[0] == ':')
      return itsHandle;
   else
      return itsHandle;
}


// write to error log file (private method)
void HP81200::WriteErrorLog(int theRet, const char* theCmd)
{
   if (itsErrorFile != 0)
   {
      char aErrorStr[1024];
      int aErrorLen = sizeof(aErrorStr);


      // write header for the error
      if (theCmd != 0)
         fprintf(itsErrorFile, "Error for cmd <%s>\n", theCmd);

      if (theRet < 0)
      {
         // get and print error message corresponding
         // to the return value
         GetErrStr_HP81200(theRet, aErrorStr, &aErrorLen);
         fprintf(itsErrorFile, "\t%s\n", aErrorStr);
      }
      else
      {
         // get and print error message from the error queue
         char aResultBuffer[1024];
         int aResultLen;
         char aCmd[128];

         sprintf(aCmd, ":%s:syst:err?", GetHandle(theCmd));
         for (;;)
         {
```


www.valuetronics.com

```
                aResultLen = sizeof(aResultBuffer);
                int ret = Call_HP81200(aCmd, aResultBuffer,
                                       &aResultLen);

                // break out of loop if error queue empty
                if ((ret != 0) || (aResultBuffer[0] == '0'))
                   break;

                fprintf(itsErrorFile, "\t%s\n", aResultBuffer);
             }
          }
       }
    }
```


www.valuetronics.com

Main.cpp Application Code

The following code is a typical example program for accessing the user software functionality.

```
// main.cpp
// This is a small demo program showing how to use
// the remote interface of the 81200A
// user software
//
// It demonstrates how to:
// - connect to a firmware server
// - set up ports and terminals,
// - apply levels / thresholds
// - import data segment
// - set up a sequence
// - set measurement mode
// - start measurement
// - find out when measurement is done
// - stop measurement
// - get captured data
// - export captured data to a file
//
// to make this code easier to read,
// it is based on a small class HP81200
// which is provided in lib.c / lib.h
// Here the error handling and the "handle" is hidden.
// Any error that occurs is logged to stdout or file.
//


#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#include "lib.h"


#define SYSTEM_NAME1 "DSRA"


#define SETTING_NAME "prog_sample"


static bool doIt(HP81200& h, FILE* fp)
{
   const char* aResult;
   char aCmd[4096];
   char aBuf[1024];
   char aAnalyzers[1024];
```


www.valuetronics.com

```
char aGenerators[1024];
int aAnaCnt;
int aGenCnt;
int aAnaPort;
int aGenPort;

// ------------------------------------------------------------
// stop system
// reset system
// ------------------------------------------------------------
h.Call("sgen:glob:init:cont off");
h.Call("mmem:sett:new");


// ------------------------------------------------------------
// first, we want to generate a list of all analyzers and
// all generators, so that we can afterwards connect to ports
// ------------------------------------------------------------
strcpy(aAnalyzers, "");
strcpy(aGenerators, "");
aAnaCnt = 0;
aGenCnt = 0;
aAnaPort = 0;
aGenPort = 0;

// find out how many clock groups we have
h.Call("conf:cgr?", aResult);
int aCgr;
sscanf(aResult, "%d", &aCgr);

// loop over all clock groups
for (int c = 1; c <= aCgr; c++)
{
   // find out how many modules within this clock group
   sprintf(aCmd, "conf:cgr%d:mod?", c);
   h.Call(aCmd, aResult);
   int aMod;
   sscanf(aResult, "%d", &aMod);

   // loop over all modules
   for (int m = 1; m <= aMod; m++)
   {
      // find out how many connectors we have...
      sprintf(aCmd, "conf:cgr%d:mod%d:conn?", c, m);
      h.Call(aCmd, aResult);
      int aConn;
      sscanf(aResult, "%d", &aConn);

      // loop over connectors
      for (int co = 1; co <= aConn; co++)
```


www.valuetronics.com

```
      {
         // query for type
         sprintf(aCmd, "conf:cgr%d:mod%d:conn%d:type?", c, m,
                 co);
         h.Call(aCmd, aResult);

         // put to analyzer or generator list
         sprintf(aBuf, "%02d%02d%03d", c, m, co);
         if (strcmp(aResult, "ANALYZER") == 0)
         {
            if (strcmp(aAnalyzers, "") != 0)
               strcat(aAnalyzers, ",");
            strcat(aAnalyzers, aBuf);
            aAnaCnt += 1;
         }
         else
         {
            if (strcmp(aGenerators, "") != 0)
               strcat(aGenerators, ",");
            strcat(aGenerators, aBuf);
            aGenCnt += 1;
         }
      }
   }
}

   // ------------------------------------------------------
// now connect analyzers and generators to one port each
// ------------------------------------------------------
if (aGenCnt > 0)
{
   sprintf(aCmd, "sgen:pdat:app 'INPUT_PORT',%d,'input'",
           aGenCnt);
   h.Call(aCmd);

   aGenPort = 1;
   sprintf(aCmd, "sgen:conn:pdat%d:term1 (@%s)", aGenPort,
           aGenerators);
   h.Call(aCmd);
}

if (aAnaCnt > 0)
{
   sprintf(aCmd, "sgen:pdat:app 'OUTPUT_PORT',%d,'output'",
           aAnaCnt);
   h.Call(aCmd);

   aAnaPort = aGenPort + 1;
   sprintf(aCmd, "sgen:conn:pdat%d:term1 (@%s)", aAnaPort,
           aAnalyzers);
   h.Call(aCmd);
}
```


www.valuetronics.com

```
// ------------------------------------------------
// switch on everything, apply levels / thresholds
// ------------------------------------------------
if (aGenPort > 0)
{
   sprintf(aCmd, "sgen:pdat%d:outp on", aGenPort);
   h.Call(aCmd);

   sprintf(aCmd, "sgen:pdat%d:volt:high 2", aGenPort);
   h.Call(aCmd);
}

if (aAnaPort > 0)
{
   sprintf(aCmd, "sgen:pdat%d:inp on", aAnaPort);
   h.Call(aCmd);

   sprintf(aCmd, "sgen:pdat%d:inp:thr 1", aAnaPort);
   h.Call(aCmd);
}


// ------------------------------------------------
// import data (overwrite mode)
// ------------------------------------------------
sprintf(aCmd,
   "mmem:segm:load '%s\\samples\\segments\\walk64.txt'",
   getenv("DVTDSRBASEDIR"));
h.Call(aCmd);


// ------------------------------------------------
// set period
// ------------------------------------------------
h.Call("sgen:glob:per 1e-6");


// ------------------------------------------------
// assign a sequence that uses imported segment
// ------------------------------------------------


// how many loop levels are available?
h.Call("sgen:glob:seq:llev?", aResult);
int aLoopLevels = 0;
sscanf(aResult, "%d", &aLoopLevels);


// make infinite loop with trigger using the imported
// demo segment
sprintf(aCmd,
        "sgen:glob:seq (1.0,'',(LOOP%d,1,INF,(BLOCK,0,64",
```


www.valuetronics.com

```
        aLoopLevels);
if (aGenPort > 0)
   sprintf(aCmd, "%s,'walking64',0,0", aCmd);
if (aAnaPort > 0)
   sprintf(aCmd, "%s,'walking64',0,0", aCmd);
sprintf(aCmd, "%s)))", aCmd);
h.Call(aCmd);


// generate trigger signal from sequence
h.Call("cgr:trig:mode seq");


// ------------------------------------------------
// set meausurement mode
// compare and acquire around error
// ------------------------------------------------
if (aAnaPort > 0)
   h.Call("sgen:glob:conf:ecap");


// ------------------------------------------------
// save setting for later use
// ------------------------------------------------
sprintf(aCmd, "mmem:sett:save '%s'", SETTING_NAME);
h.Call(aCmd);


// ------------------------------------------------
// start measurement
// ------------------------------------------------
h.Call("sgen:glob:init:cont on");


// ------------------------------------------------
// poll measurement done
// THIS ONLY WORKS ONLINE!!!
// ------------------------------------------------
for (;;)
{
   h.Call("sgen:glob:syst:stat?", aResult);
   if (strncmp(aResult, "FIN", 3) == 0)
      break;
}


// ------------------------------------------------
// stop measurement
// ------------------------------------------------
h.Call("sgen:glob:init:cont off");

// ------------------------------------------------
// - demonstrate the edit subsystem how to deal with
//   captured data
// - export the captured data and the error memory
```


www.valuetronics.com

```
// ------------------------------------------------
if (aAnaPort > 0)
{
   sprintf(aCmd, "edit:segm:open? 'Analyzer\\Capture.%d'",
           aAnaPort);
   h.Call(aCmd, aResult);
   int aCapture;
   sscanf(aResult, "%d", &aCapture);

   sprintf(aCmd, "edit:segm:open? 'Analyzer\\ErrMem.%d'",
           aAnaPort);
   h.Call(aCmd, aResult);
   int aErrMem;
   sscanf(aResult, "%d", &aErrMem);


   // get pattern width
   sprintf(aCmd, "edit:segm%d:patt:widt?", aCapture);
   h.Call(aCmd, aResult);
   int aWidth;
   sscanf(aResult, "%d", &aWidth);


   // get pattern length
   sprintf(aCmd, "edit:segm%d:patt:leng?", aCapture);
   h.Call(aCmd, aResult);
   int aLength;
   sscanf(aResult, "%d", &aLength);


   // get Coding
   sprintf(aCmd, "edit:segm%d:patt:cod?", aCapture);
   h.Call(aCmd, aResult);


   // write information we got so far:
   fprintf(fp, "Analyzer.%d: Coding <%s>,
           Width %d, Length %d\n",
           aAnaPort, aResult, aWidth, aLength);


   // get some data, but assure that we will not overflow our
   // small result-buffer
   // to make it human readable, we get the data as a hex-string
   if (aLength > 10) aLength = 10;
   if (aLength > 0)
   {
      sprintf(aCmd, "edit:segm%d:patt:data? 0,0,%d,%d,HEX",
              aCapture, aWidth-1, aLength-1);
      h.Call(aCmd, aResult);
      fprintf(fp, "Analyzer.%d: %s\n", aAnaPort, aResult);

      //
      // extract the vectors and print as '0' and '1' to fp
```


www.valuetronics.com

```
   //

   // get vector data as definite length block
   sprintf(aCmd, "edit:segm%d:patt:data? 0,0,%d,%d,BIN",
           aCapture, aWidth-1, aLength-1);
   h.Call(aCmd, aResult);

   // extract raw data from definite length block
   // a definite length block looks like
   // #dllllbbbbbbbb
   // where d is the number of length digits l
   // then length binary bytes b are following
   // Example: #90000000012xxxxxxxxxxx
   // where x stands for any value between 0 and 255,
   // it may not be a readable character!
   memcpy(aBuf, &aResult[2], (size_t)(aResult[1] - '0'));
   aBuf[aResult[1]] = '\0';
   int aBytes = 0;
   sscanf(aBuf, "%d", &aBytes);
   memcpy(aBuf, &aResult[2+aResult[1]-'0'], (size_t)aBytes);

   // print out the data trace-wise
   // - each trace begins on a byte boundary
   // - bits within a trace are "from left to right",
   //   meaning that first bit is bit 7, than bit 6
   //   and so on for a 1 bit coding.
   for (int t = 0; t < aWidth; t++)
   {
      fprintf(fp, "Trace %d: ", t);
      for (int l = 0; l < aLength; l++)
      {
         fprintf(fp, "%c", '0' +
         ((aBuf[t * ((aLength+7)/8) + l/8] >> (7 - l%8)) & 1)
         );
      }
      fprintf(fp, "\n");
   }
}


// save segments under a different name to the local setting
sprintf(aCmd,
   "edit:segm%d:save 'LocalSegments\\prog_sample_capture'",
   aCapture);
h.Call(aCmd);
sprintf(aCmd,
   "edit:segm%d:save 'LocalSegments\\prog_sample_errmem'",
   aErrMem);
h.Call(aCmd);
```


www.valuetronics.com

```
      // export segments
      // we do this now before the segments are closed, because
      // they are now still in memory
      sprintf(aCmd,
         "mmem:segm:save 'c:\\temp\\prog_sample_capture.txt', '%s',
         'prog_sample_capture'", SETTING_NAME);
      h.Call(aCmd);

      sprintf(aCmd,
         "mmem:segm:save 'c:\\temp\\prog_sample_errmem.txt', '%s',
         'prog_sample_errmem'", SETTING_NAME);
      h.Call(aCmd);


      // don't forget to close the segments again!
      sprintf(aCmd, "edit:segm%d:clos", aCapture);
      h.Call(aCmd);

      sprintf(aCmd, "edit:segm%d:clos", aErrMem);
      h.Call(aCmd);
   }



   return true;
}


int main()
{
   bool b;
   HP81200 a;
   FILE* fp;


   //fp = fopen("c:\\temp\\sample_4.txt", "w");
   fp = stdout;


   // init access to local 81200 firmware server, system "DSRA"
   // Errors are logged to a file (which may be stdout).
   b = a.Init("", "a", SYSTEM_NAME1, fp);
   if (b) {
      fprintf(fp, "System %s:\n", SYSTEM_NAME1);
      b = doIt(a, fp);
   }


   // release handle, disconnect from server
   (void)a.Exit();


   // wait, so that user could see the results
   if ((fp == stdout) || (fp == stderr))
   {
```


www.valuetronics.com

```
      printf("press any key to continue\n");
      (void)getchar();
   }
   else
      fclose(fp);


   return 0;
}

// Connect to 81200 server on local machine or "theServerName".
// Creates a handle for the system "theSystemName".
// This handle is internally handled, so that we do not need
// to specify it with each command.
bool HP81200::Init(  const char* theServerName,
                     const char* theHandleSuggestion,
                     const char* theSystemName,
                     FILE* theErrorFile)
{
   bool b;
   char aCmd[128];
   int ret;

   // remember error log file
   itsErrorFile = theErrorFile;

   // connect to firmware server
   // if already connected do nothing
   ret = Connect_HP81200(theServerName);
   itsConnected = (ret == 0);
   if (!itsConnected && (ret != -4))
   {
      WriteErrorLog(ret);
      return false;
   }

   // create a handle
   sprintf(aCmd, ":dvt:inst:hand:cre? %s,'DSR','%s'",
      theHandleSuggestion, theSystemName);
   b = Call(aCmd);

   strcpy(itsHandle, itsResultBuffer);

   return b;
}

// Release the handle, disconnect from the system
bool HP81200::Exit()
{
   if (strcmp(itsHandle, "") != 0)
   {
      bool b;
      char aCmd[128];
```


www.valuetronics.com

```
      // destroy handle
      sprintf(aCmd, ":dvt:inst:hand:dest %s", itsHandle);
      b = Call(aCmd);
      if (!b) return b;

      // clear remembered handle
      strcpy(itsHandle, "");
   }

   if (itsConnected)
   {
      int ret;

      ret = Disconnect_HP81200();
      if (ret != 0)
      {
         WriteErrorLog(ret);
         return false;
      }
   }

   return true;
}

// Call 81200, handle is automatically supplied by the class.
// Errors are logged to the logfile.
// In case of an error, false is returned.
// Results from the call are ignored.
bool HP81200::Call(const char* theCmd)
{
   int ret;
   char aCmd[1024];
   int aResultLen;

   if (strcmp(GetHandle(theCmd), "dvt") == 0)
      // already valid command with handle dvt, just use it...
      sprintf(aCmd, "%s", theCmd);
   else if (theCmd[0] == ':')
      // command for local handle
      sprintf(aCmd, ":%s%s", itsHandle, theCmd);
   else
      // command for local handle
      sprintf(aCmd, ":%s:%s", itsHandle, theCmd);

   aResultLen = sizeof(itsResultBuffer);
   ret = Call_HP81200(aCmd, itsResultBuffer, &aResultLen);
   if (ret != 0)
   {
      WriteErrorLog(ret, theCmd);
      return false;
   }
```


www.valuetronics.com

```
   return true;
}

// Call 81200, handle is automatically supplied by the class.
// Errors are logged to the logfile.
// In case of an error, false is returned.
// Results are returned as a pointer to an internal buffer,
// so before calling Call again, the results must be otherwise
// saved.
bool HP81200::Call(const char* theCmd, const char*& theResult)
{
   theResult = itsResultBuffer;

   return Call(theCmd);
}

int main()
{
   bool b;
   HP81200 a;
   FILE* fp;


   //fp = fopen("c:\\temp\\sample_4.txt", "w");
   fp = stdout;


   // init access to local 81200 firmware server, system "DSRA"
   // Errors are logged to a file (which may be stdout).
   b = a.Init("", "a", SYSTEM_NAME1, fp);
   if (b) {
      fprintf(fp, "System %s:\n", SYSTEM_NAME1);
      b = doIt(a, fp);
   }


   // release handle, disconnect from server
   (void)a.Exit();


   // wait, so that user could see the results
   if ((fp == stdout) || (fp == stderr))
   {
      printf("press any key to continue\n");
      (void)getchar();
   }
   else
      fclose(fp);


   return 0;
}
```


www.valuetronics.com

```
// ------------------------------------------------------------
// stop system
// reset system
// ------------------------------------------------------------
h.Call("sgen:glob:init:cont off");
h.Call("mmem:sett:new");

// ------------------------------------------------------------
// first, we want to generate a list of all analyzers and
// all generators, so that we can afterwards connect to ports
// ------------------------------------------------------------
strcpy(aAnalyzers, "");
strcpy(aGenerators, "");
aAnaCnt = 0;
aGenCnt = 0;
aAnaPort = 0;
aGenPort = 0;


// find out how many clock groups we have
h.Call("conf:cgr?", aResult);
int aCgr;
sscanf(aResult, "%d", &aCgr);


// loop over all clock groups
for (int c = 1; c <= aCgr; c++)
{
   // find out how many modules within this clock group
   sprintf(aCmd, "conf:cgr%d:mod?", c);
   h.Call(aCmd, aResult);
   int aMod;
   sscanf(aResult, "%d", &aMod);


   // loop over all modules
   for (int m = 1; m <= aMod; m++)
   {
      // find out how many connectors we have...
      sprintf(aCmd, "conf:cgr%d:mod%d:conn?", c, m);
      h.Call(aCmd, aResult);
      int aConn;
      sscanf(aResult, "%d", &aConn);


      // loop over connectors
      for (int co = 1; co <= aConn; co++)
      {
         // query for type
         sprintf(aCmd, "conf:cgr%d:mod%d:conn%d:type?", c, m,
                 co);
         h.Call(aCmd, aResult);

         // put to analyzer or generator list
         sprintf(aBuf, "%02d%02d%03d", c, m, co);
```


www.valuetronics.com

```
            if (strcmp(aResult, "ANALYZER") == 0)
            {
               if (strcmp(aAnalyzers, "") != 0)
                  strcat(aAnalyzers, ",");
               strcat(aAnalyzers, aBuf);
               aAnaCnt += 1;
            }
            else
            {
               if (strcmp(aGenerators, "") != 0)
                  strcat(aGenerators, ",");
               strcat(aGenerators, aBuf);
               aGenCnt += 1;
            }
         }
      }
   }

   // ------------------------------------------------
   // now connect analyzers and generators to one port each
   // ------------------------------------------------
   if (aGenCnt > 0)
   {
      sprintf(aCmd, "sgen:pdat:app 'INPUT_PORT',%d,'input'",
              aGenCnt);
      h.Call(aCmd);

      aGenPort = 1;
      sprintf(aCmd, "sgen:conn:pdat%d:term1 (@%s)", aGenPort,
              aGenerators);
      h.Call(aCmd);
   }

   if (aAnaCnt > 0)
   {
      sprintf(aCmd, "sgen:pdat:app 'OUTPUT_PORT',%d,'output'",
              aAnaCnt);
      h.Call(aCmd);

      aAnaPort = aGenPort + 1;
      sprintf(aCmd, "sgen:conn:pdat%d:term1 (@%s)", aAnaPort,
              aAnalyzers);
      h.Call(aCmd);
   }

   // ------------------------------------------------
   // switch on everything, apply levels / thresholds
   // ------------------------------------------------
   if (aGenPort > 0)
   {
      sprintf(aCmd, "sgen:pdat%d:outp on", aGenPort);
      h.Call(aCmd);
```


www.valuetronics.com

```
   sprintf(aCmd, "sgen:pdat%d:volt:high 2", aGenPort);
   h.Call(aCmd);
}

if (aAnaPort > 0)
{
   sprintf(aCmd, "sgen:pdat%d:inp on", aAnaPort);
   h.Call(aCmd);

   sprintf(aCmd, "sgen:pdat%d:inp:thr 1", aAnaPort);
   h.Call(aCmd);
}
// ------------------------------------------------
// import data (overwrite mode)
// ------------------------------------------------
sprintf(aCmd,
   "mmem:segm:load '%s\\samples\\segments\\walk64.txt'",
   getenv("DVTDSRBASEDIR"));
h.Call(aCmd);
// ------------------------------------------------
// set period
// ------------------------------------------------
h.Call("sgen:glob:per 1e-6");


// ------------------------------------------------
// assign a sequence that uses imported segment
// ------------------------------------------------


// how many loop levels are available?
h.Call("sgen:glob:seq:llev?", aResult);
int aLoopLevels = 0;
sscanf(aResult, "%d", &aLoopLevels);


// make infinite loop with trigger using the imported
// demo segment
sprintf(aCmd,
        "sgen:glob:seq (1.0,'',(LOOP%d,1,INF,(BLOCK,0,64",
        aLoopLevels);
if (aGenPort > 0)
   sprintf(aCmd, "%s,'walking64',0,0", aCmd);
if (aAnaPort > 0)
   sprintf(aCmd, "%s,'walking64',0,0", aCmd);
sprintf(aCmd, "%s)))", aCmd);
h.Call(aCmd);


// generate trigger signal from sequence
h.Call("cgr:trig:mode seq");
```


www.valuetronics.com

```
// ------------------------------------------------
// set meausurement mode
// compare and acquire around error
// ------------------------------------------------
if (aAnaPort > 0)
   h.Call("sgen:glob:conf:ecap");

// ------------------------------------------------
// save setting for later use
// ------------------------------------------------
sprintf(aCmd, "mmem:sett:save '%s'", SETTING_NAME);
h.Call(aCmd);

// ------------------------------------------------
// start measurement
// ------------------------------------------------
h.Call("sgen:glob:init:cont on");

// ------------------------------------------------
// poll measurement done
// THIS ONLY WORKS ONLINE!!!
// ------------------------------------------------
for (;;)
{
   h.Call("sgen:glob:syst:stat?", aResult);
   if (strncmp(aResult, "FIN", 3) == 0)
      break;
}

// ------------------------------------------------
// stop measurement
// ------------------------------------------------
h.Call("sgen:glob:init:cont off");

// ------------------------------------------------
// - demonstrate the edit subsystem how to deal with
//   captured data
// - export the captured data and the error memory
// ------------------------------------------------
if (aAnaPort > 0)
{
   sprintf(aCmd, "edit:segm:open? 'Analyzer\\Capture.%d'",
           aAnaPort);
   h.Call(aCmd, aResult);
   int aCapture;
   sscanf(aResult, "%d", &aCapture);

   sprintf(aCmd, "edit:segm:open? 'Analyzer\\ErrMem.%d'",
           aAnaPort);
   h.Call(aCmd, aResult);
```


www.valuetronics.com

```
int aErrMem;
sscanf(aResult, "%d", &aErrMem);


// get pattern width
sprintf(aCmd, "edit:segm%d:patt:widt?", aCapture);
h.Call(aCmd, aResult);
int aWidth;
sscanf(aResult, "%d", &aWidth);


// get pattern length
sprintf(aCmd, "edit:segm%d:patt:leng?", aCapture);
h.Call(aCmd, aResult);
int aLength;
sscanf(aResult, "%d", &aLength);


// get Coding
sprintf(aCmd, "edit:segm%d:patt:cod?", aCapture);
h.Call(aCmd, aResult);


// write information we got so far:
fprintf(fp, "Analyzer.%d: Coding <%s>,
        Width %d, Length %d\n",
        aAnaPort, aResult, aWidth, aLength);


// get some data, but assure that we will not overflow our
// small result-buffer
// to make it human readable, we get the data as a hex-string
if (aLength > 10) aLength = 10;
if (aLength > 0)
{
   sprintf(aCmd, "edit:segm%d:patt:data? 0,0,%d,%d,HEX",
           aCapture, aWidth-1, aLength-1);
   h.Call(aCmd, aResult);
   fprintf(fp, "Analyzer.%d: %s\n", aAnaPort, aResult);

   //
   // extract the vectors and print as '0' and '1' to fp
   //

   // get vector data as definite length block
   sprintf(aCmd, "edit:segm%d:patt:data? 0,0,%d,%d,BIN",
           aCapture, aWidth-1, aLength-1);
   h.Call(aCmd, aResult);

   // extract raw data from definite length block
   // a definite length block looks like
   // #dllllbbbbbbbb
   // where d is the number of length digits l
   // then length binary bytes b are following
```


www.valuetronics.com

```
   // Example: #90000000012xxxxxxxxxxx
   // where x stands for any value between 0 and 255,
   // it may not be a readable character!
   memcpy(aBuf, &aResult[2], (size_t)(aResult[1] - '0'));
   aBuf[aResult[1]] = '\0';
   int aBytes = 0;
   sscanf(aBuf, "%d", &aBytes);
   memcpy(aBuf, &aResult[2+aResult[1]-'0'], (size_t)aBytes);


   // print out the data trace-wise
   // - each trace begins on a byte boundary
   // - bits within a trace are "from left to right",
   //   meaning that first bit is bit 7, than bit 6
   //   and so on for a 1 bit coding.
   for (int t = 0; t < aWidth; t++)
   {
      fprintf(fp, "Trace %d: ", t);
      for (int l = 0; l < aLength; l++)
      {
         fprintf(fp, "%c", '0' +
         ((aBuf[t * ((aLength+7)/8) + l/8] >> (7 - l%8)) & 1)
         );
      }
      fprintf(fp, "\n");
   }
}


// save segments under a different name to the local setting
sprintf(aCmd,
   "edit:segm%d:save 'LocalSegments\\prog_sample_capture'",
   aCapture);
h.Call(aCmd);
sprintf(aCmd,
   "edit:segm%d:save 'LocalSegments\\prog_sample_errmem'",
   aErrMem);
h.Call(aCmd);


// export segments
// we do this now before the segments are closed, because
// they are now still in memory
sprintf(aCmd,
   "mmem:segm:save 'c:\\temp\\prog_sample_capture.txt', '%s',
   'prog_sample_capture'", SETTING_NAME);
h.Call(aCmd);

sprintf(aCmd,
   "mmem:segm:save 'c:\\temp\\prog_sample_errmem.txt', '%s',
   'prog_sample_errmem'", SETTING_NAME);
h.Call(aCmd);
```


www.valuetronics.com

```
    // don't forget to close the segments again!
    sprintf(aCmd, "edit:segm%d:clos", aCapture);
    h.Call(aCmd);

    sprintf(aCmd, "edit:segm%d:clos", aErrMem);
    h.Call(aCmd);
}
```


www.valuetronics.com

Copyright Agilent Technologies 2003
Printed in Germany February 2003

5988-4886EN

Agilent Technologies

www.valuetronics.com