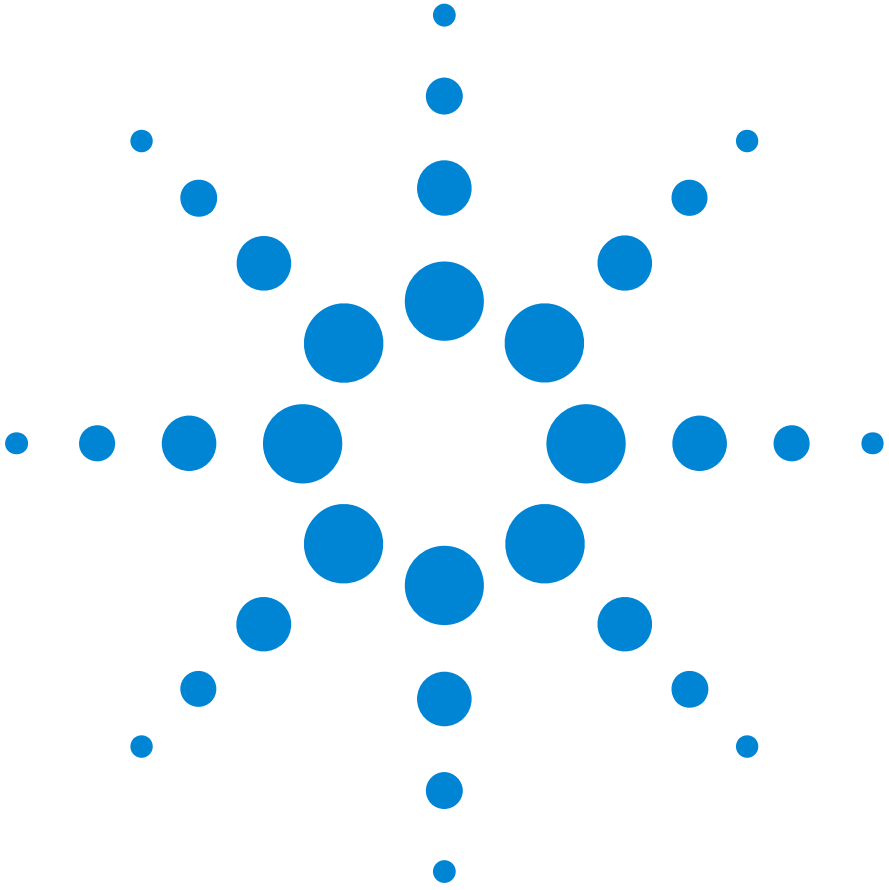


Agilent 81250 Parallel Bit Error Ratio Tester

Programming Reference

Agilent Technologies

Important Notice

Author: Stephan Greisinger, t3 medien GmbH

Contents

Contents

What's New in This Release?

The following sections give an overview of the most important changes and enhancements of the Agilent 81250 Parallel Bit Error Ratio Tester programming interfaces.

The information is organized as follows:

"Major Changes from Rev. 1.0 to 1.1" on page 16

Major Changes from Rev. 1.0 to 1.1

On from revision 1.1, the new E4832A 660 MHz Data Generator/Analyzer module is supported. This module provides four slots for four frontends,with one or two channels each.

Programming Interfaces

The Agilent 81250 provides three basic programming models. It can be controlled

- locally from the control PC,
- remotely via GPIB, or
- remotely via LAN.

These programming models are introduced in *"Programming Models" on page 18*. There you will also find references to other documents providing more detailed information on each model.

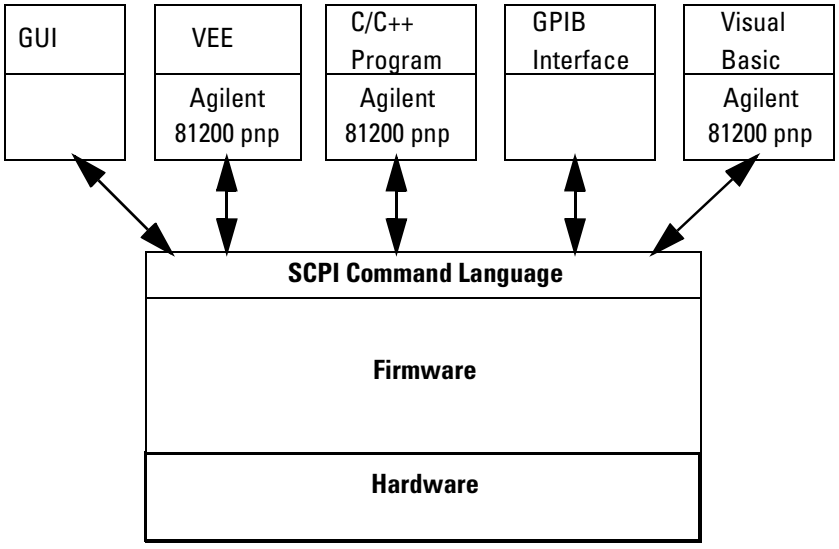
In addition to these programming models, the Command Line Interface window provided by the user interface can be used to try out commands, and to store and replay simple sequences of commands (see *"Command Line Interface" on page 23*).

"Agilent 81250 String Interface Library" on page 24 summarizes the major aspects to be considered when implementing programs for local control or for remote control via LAN using the string interface.

If you wish to use the Agilent 81250 to control other instruments, see *"Using the Agilent 81250 as a GPIB System Controller" on page 28*.

Programming Models

The following picture shows the basic structure of the Agilent 81200's programming interfaces.



Physical Interfaces

The instrument can be controlled via the following interfaces

- GPIB connector

  Using the GPIB connector, the instrument can be controlled from a PC or a UNIX Workstation.

- LAN connector

  Using the LAN connector, the instrument can be connected to a local area network and can be programmed from a PC.

- local control PC

  Control programs can also be implemented on the local control PC.

Firmware Server and SCPI Commands

All interfaces use the same SCPI-like language to communicate with the instrument's firmware server. The firmware server implements a client server architecture, allowing to connect multiple clients simultaneously.

The GUI also communicates via this language and server. Therefore, everything that can be done via the user interface can also be done via the programming interfaces.

Tools and Programming Languages

For either interface, tools like VEE or LabView, and the programming languages C/C++ and Visual Basic can be used for implementation. When using these tools or languages, the Agilent 81200 plug & play drivers can be used, providing sophisticated access to the instrument's features—without the need of using the SCPI commands. For detailed information on the Agilent 81200 plug & play drivers, please refer to the corresponding online help files.

NOTE For compatibility reasons, the system can also be programmed directly by using the SCPI commands. Examples and reference information are provided in this document.

Locking Mechanisms

Different clients may operate on the same system concurrently using different interfaces. To avoid undesired interrupts, the Agilent 81200 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.

Code Examples

The following sections show simple C code examples (not using the Agilent 81200 plug & play drivers). The examples implement the typical structure of control programs including the following steps:

- opening a connection to the instrument,
- executing SCPI commands (in the examples: requesting and printing an instrument identification string),
- closing the connection.

A more advanced example is discussed in *"Example C++ Program" on page 37*.

Programming Model: GPIB

The following code snippet shows how to use the VISA (Virtual Instrument Software Architecture) library to connect to the instrument via GPIB.

```
#include <visa.h>
#include <stdio.h>
int main () {
   ViSession defaultRM, vi; char buf[256];

   // Open session to GPIB device
   viOpenDefaultRM(&defaultRM);
   viOpen(defaultRM, "GPIB0::11::INSTR", VI_NULL,VI_NULL, &vi);

   // Send an DVT:IDN? string to the device, read & print results
   viPrintf(vi, ":DVT:IDN?\n");
   viScanf(vi, "%t", buf);
   printf("Instrument identification string: %s\n", buf);

   // Close session
   viClose(vi);
   viClose(defaultRM);
   return 0;
}
```

After opening the connection, a SCPI command string (query) is sent to the instrument to request the instrument identification string. Finally, the connection is closed again.

NOTE The GPIB daemon running on the Agilent 81250 provides a control panel window, which will be very useful for debugging.

More Information For more information on the VISA library, please refer to the *HP VISA User's Manual*. The complete command reference is also available online (hpvisa.hlp).

The SCPI command structure and the individual commands are described in detail in *"Overview of the SCPI Command Structure" on page 29*.

Programming Model: LAN

The following code snippet shows how to use the Agilent 81250 string interface library to connect to the instrument via LAN.

```
#include <stdio.h>
#include <hp81200.h>

int main() {
   int ret;
   int size= 256;
   char resp[256];

   /* connect to firmware server on your.lan.address*/
   ret= Connect_HP81200("your.lan.address");
   /* get idn-string */
   ret= Call_HP81200
         (":dvt:idn?", resp, &size );
   printf("Hello HP81200A world <%s>\n", resp);

   /* disconnect from firmware server */
   ret= Disconnect_HP81200();
   return 0;
}
```

First, the program connects to the firmware server running on the instrument. You have to fill in the LAN address of the instrument.

Next, a SCPI command string (query) is sent to the instrument to request the instrument identification string. Finally, the connection is closed again.

More Information For more information on the Agilent 81250 string interface library, please refer *"Agilent 81250 String Interface Library" on page 24*.

The SCPI command structure and the individual commands are described in detail in *"Overview of the SCPI Command Structure" on page 29*.

Programming Model: Local

The following code snippet shows how to use the Agilent 81250 string interface library to control the instrument from the local PC.

```
#include <stdio.h>
#include <hp81200.h>

int main() {
   int ret;
   int size= 256;
   char resp[256];

   /* connect to local firmware server */
   ret= Connect_HP81200("");
   /* get idn-string */
   ret= Call_HP81200(":dvt:idn?", resp, &size );
   printf("Hello HP81200A world <%s>\n", resp);

   /* disconnect from firmware server */
   ret= Disconnect_HP81200();
   return 0;
}
```

First, the program connects to the firmware server running on the instrument. Using an empty string with this function results in connecting to the local host.

Next, a SCPI command string (query) is sent to the instrument to request the instrument identification string. Finally, the connection is closed again.

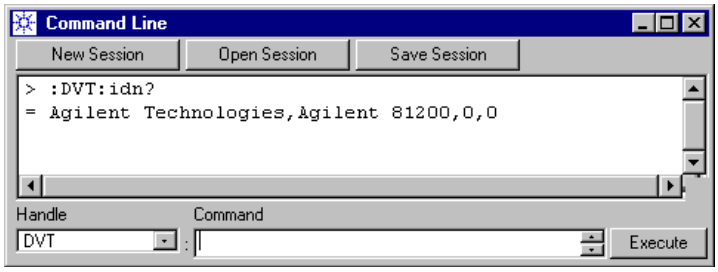
More Information

For more information on the Agilent 81250 string interface library, please refer *"Agilent 81250 String Interface Library" on page 24*.

The SCPI command structure and the individual commands are described in detail in *"Overview of the SCPI Command Structure" on page 29*.

Command Line Interface

The Agilent 81250 user interface features a Command Line window (*Command Line* in the *Go* menu), which 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.

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.

The `Call_HP81200` function allows you to pass through any valid SCPI command as described in *"Overview of the SCPI Command Structure" on page 29*.

Error Handling

All functions in this library return an integer 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. Query the error queue corresponding to the last command. That means, if the last command started with `:dvt:`, pass the command `:dvt:syst:err?` until the error queue is empty, otherwise use `:<handle>:syst:err?` using the handle of your system.

Function Reference

Call_HP81200

Syntax

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

Description Send a command to the Agilent 81250 server and get the response.

Parameters

Command Any valid SCPI command as described in *"Overview of the SCPI Command Structure" on page 29*. If the command is a query, the response is returned in `Response`.

***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 (`*Response`). On return, the variable holds the actual number of valid characters in the buffer.

Return Value Returns 0 on success and a negative number for server problems (use *"GetErrStr_HP81200" on page 27* 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 24*).

NOTE The parameter `Response` must point to a buffer area provided by the caller. The caller is responsible to reserve enough memory to hold the response data. To avoid writes beyond the buffer area, the parameter `BufferSize` must point to an integer variable holding the size of the buffer. The variable is modified on function return to hold the actual number of characters in the buffer. If the response does not fit into the buffer, an error is returned.

Connect_HP81200

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

Description Use this function to connect to an Agilent 81250 server. This must be the first call when using the Agilent 81250 string interface library.

Parameters **ServerName.** A pointer to a string containing the machine name to connect to. If left empty (NULL ptr 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 (one or more of the characters “,”, “;”, “:”, or a tab (“\t”).
- `<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 set.

Examples Examples for legal arguments 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)
- `"; 7"`: override port, resolves to “127.0.0.1; 7”
- `"15.100.10.20; 2208"`: full specified host
- `"mypc.bbn.hp.com; 2208"`: full specified host

Disconnect_HP81200

Syntax `int HP81200_CDECL Disconnect_HP81200()`

Description Disconnect from the Agilent 81200 server. This must be the last call when using the Agilent 81250 string interface library.

GetErrStr_HP81200

Syntax

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

Description Get the error string if one of the calls returns a negative value (see *"Error Handling" on page 24*). 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` must point to a buffer area provided by the caller. The caller is responsible to reserve enough memory to hold the response data. To avoid writes beyond the buffer area, the parameter `BufferSize` must point to an integer variable holding the size of the buffer. The variable is modified on function return to hold the actual number of characters in the buffer. If the response does not fit into the buffer, an error is returned.

Using the Agilent 81250 as a GPIB System Controller

By default, the Agilent 81250 is configured as a standard instrument. The GPIB gateway on the controller E9850A (opt.#012) is used to pass the SCPI commands from the GPIB to the instrument firmware.

If the Agilent 81250 is to control an external instrument, it must be configured as an GPIB system controller.

Change Agilent 81250 Configuration

First you need to change the setting in the Agilent 81250 Config tool. Do this step first, because the next steps will require a reboot.

1 Start the Agilent 81250 Config tool.

2 Uncheck the *GPIB Gateway* check box.

3 Click *OK*.

Change GPIB Settings

Now change the settings of the GPIB interface. The procedure is the same on all controllers that use the Standard I/O libraries.

1 Log on as “Administrator” (Press the “Shift” button during startup, otherwise you will automatically be logged in as user “dvt” again)

The procedure requires a reboot at the end, so it is recommended to exit all applications at this stage.

2 Click the *Start* button on the task bar and choose *Programs – HP I_O Libraries – I_O Config*.

3 In the field *Configured Interfaces* select *hpib/GPIB0* from the list and click *Edit...*

4 Mark the check box *System Controller* and change the *Bus Address* to 21.

To use the Agilent 81250 as an instrument controlled by an external GPIB controller, change the *Bus Address* to an address other than 21 because this address is reserved for the system controller.

5 Click *OK* twice.

A dialog pops up with the message *“You need to reboot NT before any changes you made will take effect.”*

6 Click *OK*.

7 Reboot your system.

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 30*.

The overall structure of the commands is described in *"The Command Tree" on page 32*.

A basic and a more advanced programming example using the Agilent 81200 SCPI commands are given in *"Example Programs" on page 35*.

"SCPI Commands Overview" on page 40 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 57* and *"Commands in DSR Application Subsystems" on page 65*.

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.

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 an 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 (:). This is a typical example:

```
: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 send the abbreviated form. For better program readability, you may 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.
- Numeric suffixes are represented by (*) in command descriptions.
- Queries are denoted by a question mark (?) at the end of the header.

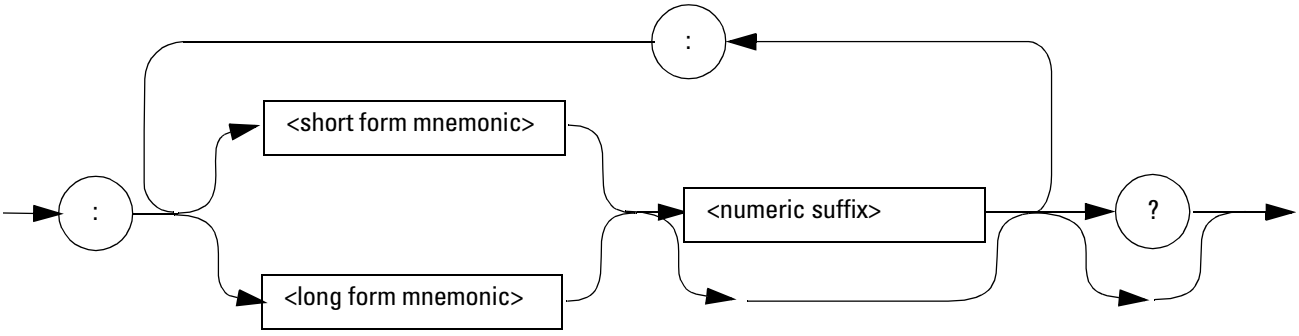
NOTE The instrument will accept either the abbreviated command form or the entire command form, but not any other forms.

The commands are not case sensitive, so upper or lower case letters or a mix of them are all 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, which has to execute the command. This is done by a leading node, the <Handle>. When linking commands, leading nodes can be omitted if identical.

Example

```
:_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 32*.

Lots of parameters can be specified on port, terminal and connector level. These aspects are discussed in *"Multiple Parameter Access" on page 33*.

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.

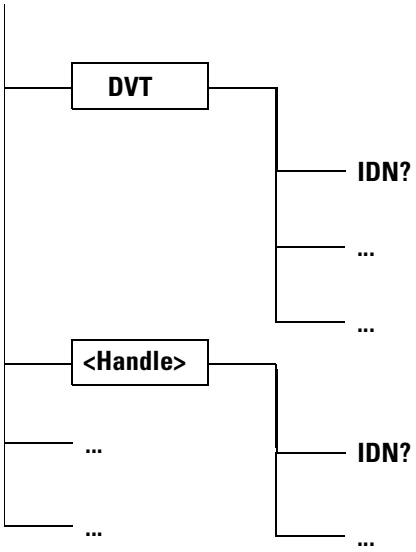
The major administrative task is to manage so-called **virtual instruments**. A virtual instrument is a combination of

- 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 35* for an example.

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

The <handle>:SYSTem:ERRor? query (see *":<Handle>:SYSTem:ERRor?" on page 67*) is used to retrieve the next error in the error queue.

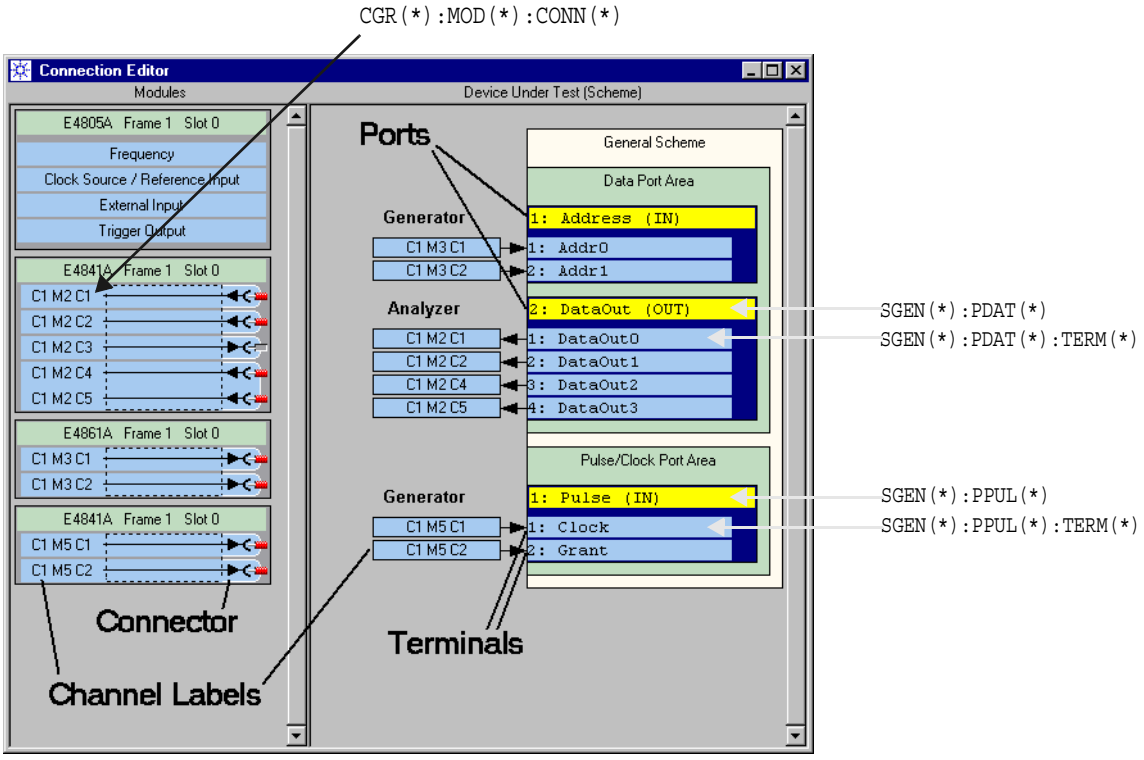
Multiple Parameter Access

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.

The following picture shows some commands and the corresponding elements in the Agilent 81250 user interface:



Data ports and pulse ports

The picture also shows how you can distinguish between data ports (:PDATa(*):) and pulse ports (:PPULse(*):) on command level.

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 145*.

Example Programs

Two example programs are discussed in the following:

- *"Typical SCPI Programming Sequence" on page 35* shows a typical sequence of SCPI commands used for programming the Agilent 81250.
- *"Example C++ Program" on page 37* guides you through an example program and a class library being supplied with the Agilent 81250.

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
:_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:
```

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 38* and *"The doIt() Procedure" on page 38* holding the application code. The program uses a small interface class (see *"The lib.cpp Interface Class Library" on page 37*), providing methods to initialize the system, to deal with handles, to execute the SCPI commands, and to perform error logging.

The files with the example code (main.cpp, lib.cpp, and lib.h) can be found in your Agilent 81250 installation directory hp81200\dsr\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 24*.

For a complete listing, refer to *"Lib.cpp Interface Class Library Code" on page 220*.

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.

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 224*.

The main() procedure implements the following steps:

- Create an HP 81200 object and call the Init() method to create a handle and to connect to the system. stdout is used for error logging.
- Invoke the doIt() procedure, which prepares and executes all required SCPI commands.
- Call 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 224*.

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

- Stop and reset the system.
- Analyze 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.

- Connect 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.

- Switch on all generator and analyzer ports, and apply levels and thresholds.
- Import segment data from the file walk64.txt.
- Set the period.
- Set 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.
- Set measurement mode to “compare and acquire around error”
- Save the setting for later use.
- Start the measurement.
- Wait until the measurement is done (only works online).
- Stop the measurement.
- Export 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.

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:

Generic Administration Commands

Command (Link to detailed description)	Description
":DVT:IDN?" on page 59	Returns the device identity.
":DVT:SYSTem:ERRor?" on page 60	Returns the first entry of the error queue.
":DVT:INSTrument:LIST?" on page 61	Returns a comma-separated list of items containing the names of all applications.
":DVT:INSTrument:RESource:LIST?" on page 61	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 62	Creates a handle for a new virtual instrument.
":DVT:INSTrument:HANDle:LIST?" on page 63	Returns a list of all virtual instrument handles.
":DVT:INSTrument:HANDle:DESTroy" on page 63	Destroys the handle for a virtual instrument and frees all resources.

Virtual Instruments Administration Commands

Command (Link to detailed description)	Description
":DVT:IDN?" on page 59	Returns the device identity.
":DVT:SYSTem:ERRor?" on page 60	Returns the first entry of the error queue.
":DVT:INSTrument:LIST?" on page 61	Returns a comma-separated list of items containing the names of all applications.
":DVT:INSTrument:RESource:LIST?" on page 61	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 62	Creates a handle for a new virtual instrument.
":DVT:INSTrument:HANDle:LIST?" on page 63	Returns a list of all virtual instrument handles.
":DVT:INSTrument:HANDle:DESTroy" on page 63	Destroys the handle for a virtual instrument and frees all resources.
":<Handle>:OPC?" on page 66	The query returns when all modules report that they are ready.
":<Handle>:SYSTem:CINFormation?" on page 68	Returns the configuration information of an Agilent 81250 system.
":<Handle>:IDN?" on page 66	Returns the device identity.
":<Handle>:SYSTem:ERRor?" on page 67	Returns the first entry of the error queue.
":<Handle>:SYSTem:CHECks" on page 67	Switches the system internal error and limit checks off or on.
":<Handle>:SYSTem:CHECks?" on page 68	This query returns the current state of the system internal error and limit checks.

Command (Link to detailed description)	Description
":<Handle>:SYSTem:CLIent[:HANDle]?" on page 71	Returns the name of the virtual instrument currently talking to the server.
":<Handle>:SYSTem:CLIent:LOCK" on page 71	Used to take over the control of the virtual instrument explicitly.
":<Handle>:SYSTem:CLIent:BLOCk" on page 72	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 72	Used to "unlock" or to "set free" the virtual instrument.
":<Handle>:CONFiguration:CGRoups(*)?" on page 73	Returns the number of clock groups that are available in the system.
":<Handle>:CONFiguration:CGRoups(*):MODules(*)?" on page 73	Returns the number of modules that are available in a clock group.
":<Handle>:CONFiguration:CGRoups(*):MODules(*):CONNectors(*)?" on page 74	Returns the number of connectors of a module.
":<Handle>:CONFiguration:CGRoups(*):MODules(*):CONNectors(*):TYPE?" on page 74	Returns the type of the connector.
":<Handle>[:CGRoup(*)]:CINFormation?" on page 99	Returns the clock group configuration information.
":<Handle>[:CGRoup(*)]:MODule(*):CINFormation?" on page 101	Returns the module configuration information.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CINFormation?" on page 103	Returns the connector configuration information.
":<Handle>:CONFiguration:STYPes?" on page 74	Returns a list of Scheme TYPes supported by the Agilent 81250 System.
":<Handle>:CONFiguration:PROFile[:VALue]" on page 75	Sets a profile parameter to a specified value.
":<Handle>:CONFiguration:PROFile[:VALue]?" on page 75	Returns an expression containing the value of a profile parameter.
":<Handle>:CONFiguration:PROFile:REMove" on page 75	Removes the specified parameter from the profile.
":<Handle>:CONFiguration:PROFile:LIST?" on page 76	Lists the parameters contained in the profile.
":<Handle>:SYSTem:TEST:MODule" on page 68	Starts a module selftest. The results of this test are stored in the message queue.
":<Handle>:SYSTem:TEST:GLOBal" on page 69	Starts a system-wide selftest. The results of this test are stored in the message queue.
":<Handle>:MODule:SREVisions" on page 69	Checks the actual BIOS Software revisions against the expected revision located in the firmware.
":<Handle>:SYSTem:MQUeue[:READ]?" on page 70	Returns the first item contained in the message queue.
":<Handle>:SYSTem:MQUeue:LENGth?" on page 70	Returns the length of the message queue.
":<Handle>:SYSTem:PON[:STATus]" on page 70	Moves the messages, which are generated by the modules at system start up, to the message queue.
":<Handle>:SGENeral:INFormation:PCLasses?" on page 112	Returns the list of implemented port classes in a comma separated quoted string list.

Mass Memory Commands

Command (Link to detailed description)	Description
":<Handle>:MMEMory:INFormation?" on page 77	Returns detailed information about the specified object.
":<Handle>:MMEMory:LIST?" on page 76	Returns an expression containing a comma separated list of quoted strings. Each string contains one element found in the directory <"Absolute Path">.
":<Handle>:MMEMory:SEGMent:LOAD" on page 77	Loads new segments into or updates segments in the Agilent 81250 System's database
":<Handle>:MMEMory:SEGMent:SAVE" on page 78	Stores segments to file.
":<Handle>:MMEMory:SEGMent:GET?" on page 79	Returns an expression that contains the contents of segment(s) (vector definition).
":<Handle>:MMEMory:SETTing:NAME?" on page 79	Returns the current setting name.
":<Handle>:MMEMory:SETTing:LOAD" on page 79	Loads the current setting or a specified setting into the Agilent 81250 System's database.
":<Handle>:MMEMory:SETTing:SAVE" on page 80	Saves changes either into the current setting or into a new setting.
":<Handle>:MMEMory:SETTing:NEW" on page 80	Performs a reset and creates an "untitled" segment.
":<Handle>:MMEMory:SETTing:DELete" on page 81	Deletes a specified setting from the Agilent 81250 System's database.
":DVT:INSTrument:HANDle:MMEMory:SETTing:IMPort" on page 64	Allows to import a setting into the Agilent 81250 System.
":DVT:INSTrument:HANDle:MMEMory:SETTing:EXPort" on page 63	Exports a setting into a newly created file.
":DVT:INSTrument:HANDle:MMEMory:SETTing:EXPort?" on page 64	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 85	Creates a new segment.
":<Handle>:EDIT:SEGMent(*):OPEN?" on page 83	Opens the specified segment.
":<Handle>:EDIT:SEGMent(*):SAVE" on page 84	Saves the changes made to the current segment name, or creates a new segment name.
":<Handle>:EDIT:SEGMent(*):CLOSe" on page 87	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 84	Deletes the specified segment.
":<Handle>:EDIT:SEGMent(*):EXISts?" on page 85	Checks whether the specified segment name is located in the Agilent 81250 System's database.
":<Handle>:EDIT:SEGMent(*):RPATh?" on page 86	Returns the absolute path for one segment.
":<Handle>:EDIT:SEGMent(*):PATTern:CODing?" on page 90	Returns the segments coding.

Command (Link to detailed description)	Description
":<Handle>:EDIT:SEGMent(*):PATTern:CODing" on page 89	Sets a segment's coding.
":<Handle>:EDIT:SEGMent(*):PATTern:DATA" on page 87	Creates a state range and fills it with the required pattern.
":<Handle>:EDIT:SEGMent(*):PATTern:DATA?" on page 88	Returns from the segment inspector the vector stream bounded by the specified rectangle.
":<Handle>:EDIT:SEGMent(*):PATTern:LENGth?" on page 91	Returns the numbers of vectors of a segment.
":<Handle>:EDIT:SEGMent(*):PATTern:WIDTh?" on page 91	Returns the numbers of traces of a segment.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:COPY" on page 91	Copies the bounded rectangle of data from the segment into the clipboard.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:PASTE" on page 92	Pastes the data currently stored in the clipboard to the specified segment.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:FILL" on page 92	Sets the bounded area of the segment with the same state value.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:INVert" on page 93	Inverts the states information in the bounded area.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:MIRRor" on page 94	Rotates the specified block about horizontal (VECTor) or vertical axis (TRACe).
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:INSert" on page 94	Inserts new traces or vectors into a segment.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:DELete" on page 95	Deletes traces or vectors from a segment.
":<Handle>:EDIT:SEGMent(*):PATTern:MODify:CONVerse" on page 95	Changes the current coding for the segment and converts existing data in the segment according to the specified mapping
":<Handle>:EDIT:SEGMent(*):PARAmeter:LENGth?" on page 96	Gets the length of the list of parameters available for this segment.
":<Handle>:EDIT:SEGMent(*):PARAmeter:LIST?" on page 96	Gets the list of parameters available for this segment.
":<Handle>:EDIT:SEGMent(*):PARAmeter[:VALue]?" on page 97	Returns the value for a specified parameter from the segment.
":<Handle>:EDIT:SEGMent(*):PARAmeter[:VALue]" on page 97	Sets the value for a specified parameter in the segment.
":<Handle>:EDIT:SEGMent(*):PARAmeter:REMove" on page 97	Removes a parameter from the list of parameters in the segment.
":<Handle>:EDIT:SEGMent(*):TYPE" on page 98	Sets the segment type for the current segment.
":<Handle>:EDIT:SEGMent(*):TYPE?" on page 98	Returns the segment type of the segment specified by the segment inspector index number.

Global Commands

Command (Link to detailed description)	Description
":<Handle>:SGENeral:GLOBal:INITiate:CONTinuous" on page 121	Starts/stops the system.
":<Handle>:SGENeral:GLOBal:INITiate:CONTinuous?" on page 121	Returns the current state of the system.
":<Handle>:SGENeral:GLOBal:SYSTem:STATe?" on page 122	Returns the current state of the system.
":<Handle>:SGENeral:GLOBal:PERiod" on page 115	Sets the period of the Agilent 81250 System.
":<Handle>:SGENeral:GLOBal:PERiod?" on page 115	Returns the period of the Agilent 81250 System.
":<Handle>:SGENeral:GLOBal:FREQuency" on page 115	Sets the frequency of the Agilent 81250 System.
":<Handle>:SGENeral:GLOBal:FREQuency?" on page 116	Returns the frequency of the Agilent 81250 System.
":<Handle>:SGENeral:GLOBal:MUX" on page 116	Defines the global settings for the Segment Resolution.
":<Handle>:SGENeral:GLOBal:MUX?" on page 116	Returns the current MUX factor (Segment Resolution).
":<Handle>:SGENeral:GLOBal:DOFFset" on page 114	Specifies an offset value to the fixed delay for all connectors.
":<Handle>:SGENeral:GLOBal:DOFFset?" on page 114	Returns the currently set delay offset.
":<Handle>:SGENeral:GLOBal:CONNect" on page 113	Connects or disconnects all enabled connectors of the Agilent 81250 System.
":<Handle>:SGENeral:GLOBal:CONNect?" on page 113	Returns the actual connection status of the enabled connectors.
":<Handle>:SGENeral:GLOBal:FETCh:ERRor:ANY?" on page 114	Reports whether there has been found any error.

Clock Reference Input Commands

Command (Link to detailed description)	Description
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer][:SOURce]" on page 137	Sets the clock reference or the external clock mode.
":<Handle>:SGENeral:GLOBal:TRIGger?" on page 137	Returns the actual status of the Clock/Reference Input of the Central resource.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:RCLock:DETect" on page 139	Measures the external clock reference and sets the corresponding mode automatically.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLock:MEASurement" on page 139	Measures the external clock and sets the corresponding mode EXTernal automatically.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:TVOLtage" on page 139	Specifies the termination voltage.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:TVOLtage?" on page 140	Returns the actual setting of the termination voltage.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLock[:VALue]?" on page 138	Measures and returns the supplied external clock at the clock input of the E4805A central module.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLock:MULTiplier" on page 138	Sets the clock multiplier.
":<Handle>:SGENeral:GLOBal:TRIGger[:SEQuence][:LAYer]:CLock:MULTiplier?" on page 138	Returns the current clock multiplication factor.

External Input Commands

Command (Link to detailed description)	Description
":<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer][:SOURce]" on page 142	Controls the start mode of the Agilent 81250 System.
":<Handle>:SGENeral:GLOBal:ARM?" on page 142	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 142	Specifies the input level condition which is used in the GATed | STARTsignal | STOPsignal modes.
":<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]:SENSe?" on page 143	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 143	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 143	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 144	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 144	Returns the actual termination voltage for the external input signal used in the gate or start/stop mode.

Trigger Output Commands

Command (Link to detailed description)	Description
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:DELay" on page 106	Varies the delay of the trigger output signal.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:DELay?" on page 107	Returns the current delay of the trigger output signal.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MUX" on page 107	Sets the frequency multiplier factor for the individual connector.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MUX?" on page 107	Returns the MUX factor associated with the trigger output.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:HIGH" on page 108	Sets the high voltage level of the trigger output signal.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:HIGH?" on page 108	Returns the high voltage level of the trigger output signal.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:LOW" on page 108	Sets the low voltage level of the trigger output signal.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:LOW?" on page 108	Returns the low voltage level of the trigger output signal.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:TVOLtage" on page 109	Specifies the termination voltage of a trigger output connector.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:TVOLtage?" on page 109	Returns the current termination voltage of the trigger output connector.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:IMPedance:EXTernal" on page 109	Specifies the external termination IMPedance.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:IMPedance:EXTernal?" on page 109	Returns the current programmed external termination impedance (load impedance).
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MODE" on page 110	Select the source of the trigger output.
":<Handle>[:CGRoup(*)][:SOURce]:TRIGger:MODE?" on page 110	Returns the actual source used for the trigger signal.

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 112	Returns the list of implemented port classes in a comma separated quoted string list.
"APPend" on page 189	Creates and appends a new port to the list of ports.
"LIST?" on page 190	Returns a comma-separated list of port names.
"ATYPes?" on page 190	Returns a comma-separated list of available predefined port TYPes.
"DELete" on page 190	Deletes the port specified by the suffix of PPULse.
"REName" on page 191	Renames the port specified by the suffix of PPULse.
"NAME?" on page 191	Returns the name of the specified port.
"TYPE?" on page 191	Returns the type of the specified port.
"MUX" on page 156	Sets the frequency multiply factor (MUX factor) for the specified port.
"MUX?" on page 157	Returns the current frequency multiply factor (MUX factor) for the specified port.
"OUTPut:IMPedance:EXTernal" on page 183	Sets the external termination impedance
"OUTPut:IMPedance:EXTernal?" on page 184	Returns the currently programmed value for the termination (load) impedance for the connectors of the specified port.
"CALibration:CDELay" on page 192	Sets a cable delay for the specified port, to synchronize the signals at the DUT terminals.
"CALibration:CDELay?" on page 192	Returns the current cable delay for the specified port.

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 193	Creates and appends a new terminal to the list of terminals.
"LIST?" on page 193	Returns a comma-separated list of terminal names.
"DELete" on page 194	Deletes the terminal specified by the suffix of TERMinal.
"REName" on page 194	Renames the terminal specified by the suffix of TERMinal.
"NAME?" on page 194	Returns the name of the specified terminal.
"TYPE?" on page 195	Returns the type of the specified terminal.
"MOVE" on page 195	Moves the specified terminal to a new position.
"CALibration:CDELay" on page 196	Sets a cable delay for the specified terminal in the specified port to synchronize the signals at the DUT terminals.
"CALibration:CDELay?" on page 196	Returns the current cable delay for the specified terminal in the specified port.

Connector Administration Commands

Command (Link to detailed description)	Description
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CINFormation?" on page 103	Returns the connector configuration information.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):TYPE?" on page 104	Returns the product number of the frontend to which the connector belongs.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):NAME?" on page 104	Returns the name of this connector.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):TNAMe?" on page 104	Returns the Terminal NAMe to which this connector is connected.
"MUX" on page 156	Sets the MUX factor associated with this connector.
"MUX?" on page 157	Returns the MUX factor associated with this connector.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:CDELay" on page 105	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 105	Returns the current cable delay for the specified connector of a specific module.
":<Handle>[:CGRoup(*)]:MODule(*):CONNector(*):CALibration:ZDELay" on page 105	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 106	Returns the current zero delay for the specified connector.
"OUTPut:IMPedance:EXTernal" on page 183	Specifies the external termination impedance.
"OUTPut:IMPedance:EXTernal?" on page 184	Returns the currently programmed external termination impedance (load impedance).

Clockgroup Administration Commands

Command (Link to detailed description)	Description
":<Handle>[:CGRoup(*)]:CINFormation?" on page 99	Returns the clock group configuration information.
":<Handle>[:CGRoup(*)]:MCLock:SOURce[:VALue]" on page 100	Controls which of the clock modules generates the "master" clock.
":<Handle>[:CGRoup(*)]:MCLock:SOURce[:VALue]?" on page 100	Returns the state of the clock module specified.
":<Handle>[:CGRoup(*)]:MODule(*):CINFormation?" on page 101	Returns the module configuration information.
":<Handle>[:CGRoup(*)]:MODule(*):TYPE?" on page 102	Returns the product number in a quoted string.
":<Handle>[:CGRoup(*)]:MODule(*):SLOT?" on page 102	Returns the slot number in which the module is located.
":<Handle>[:CGRoup(*)]:MODule(*):FRAMe?" on page 102	Returns the frame number to which this module belongs.
":<Handle>[:CGRoup(*)]:MODule(*):NAME?" on page 102	Returns the name of this module.
":<Handle>[:CGRoup(*)]:MODule(*):CNAMes?" on page 102	Returns a quoted list of Connector names.

Sequence Commands

Command (Link to detailed description)	Description
":<Handle>:SGENeral:GLOBal:SEQuence[:VALue]" on page 124	Loads a data sequence in the form of an expression into the system.
":<Handle>:SGENeral:GLOBal:SEQuence[:VALue]?" on page 127	Returns the current data sequence of the system.
":<Handle>:SGENeral:GLOBal:SEQuence:EVENTs" on page 128	Defines events for the system.
":<Handle>:SGENeral:GLOBal:SEQuence:EVENTs?" on page 130	Returns a list of events defined for the system.
":<Handle>:SGENeral:GLOBal:SEQuence:FORCe" on page 131	Downloads the actual sequence immediately.
":<Handle>:SGENeral:GLOBal:SEQuence:LLEVel?" on page 131	Returns the actually available loop levels.
":<Handle>:SGENeral:GLOBal:SEQuence:PCONtrol" on page 132	Sets the value of the program control event.
":<Handle>:SGENeral:GLOBal:SEQuence:PCONtrol?" on page 132	Returns the value of the program control event.

Synchronization Commands

Command (Link to detailed description)	Description
":<Handle>:SGENeral:GLOBal:SYNChronization:USED?" on page 133	Returns whether a synchronization flag is found in the programmed sequence.
":<Handle>:SGENeral:GLOBal:SYNChronization:BERThreshold" on page 133	Sets the threshold for the bit error ratio to be accepted during synchronization.
":<Handle>:SGENeral:GLOBal:SYNChronization:BERThreshold?" on page 134	Returns the currently set bit error ratio threshold.
":<Handle>:SGENeral:GLOBal:SYNChronization:SMODe" on page 134	Specifies how the synchronization of the analyzers is achieved.
":<Handle>:SGENeral:GLOBal:SYNChronization:SMODe?" on page 135	Returns the currently set synchronization mode.
":<Handle>:SGENeral:GLOBal:SYNChronization:APAlignment" on page 135	Specifies whether an automatic phase optimization is done after a synchronization in infinite range.
":<Handle>:SGENeral:GLOBal:SYNChronization:APAlignment?" on page 136	Returns whether an automatic phase optimization is done after a synchronization in infinite range.
":<Handle>:SGENeral:GLOBal:SYNChronization:PACCuracy" on page 136	Specifies how accurate the phase optimization is done after a synchronization in infinite synchronization range.
":<Handle>:SGENeral:GLOBal:SYNChronization:PACCuracy?" on page 136	Returns the required phase optimization accuracy.

Analyzer Commands

Command (Link to detailed description)	Description
":<Handle>:SGENeral:GLOBal:CONFigure?" on page 118	Returns the last measurement mode setting in a quoted string.
":<Handle>:SGENeral:GLOBal:CONFigure:CAPTure" on page 118	Activates the Capture measurement mode.
":<Handle>:SGENeral:GLOBal:CONFigure[:ECOunt]" on page 119	Activates the Error Rate measurement mode.
":<Handle>:SGENeral:GLOBal:CONFigure:ECAPture" on page 119	Activates the Error Capture measurement mode.
":<Handle>:SGENeneral:GLOBal:CONFigure:CCAPture" on page 120	Sets the instrument to Compare and CAPture mode.
"FETCh[:ECOunt]?" on page 199	Activates the Error Count measurement mode.
"ECOunt:RESet" on page 200	Resets the "received bit counter" and the "failed bit counter" to zero.
":<Handle>:SGENeral:GLOBal:FETCh:ERRor:ANY?" on page 114	Reports whether there has been found any error.
"FETCh:ERRor:ANY?" on page 199	Returns whether there has been found any error for the specified port or terminal.

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 182	Sets the external termination voltage.
"OUTPut:TVOLtage?" on page 183	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 158	Sets the high level of the pulse.
"VOLTage[:LEVel][:IMMediate]:HIGH?" on page 159	Returns the high level value for the specified port/terminal/connector.
"VOLTage[:LEVel][:IMMediate]:LOW" on page 159	Sets the low level of the pulse.
"VOLTage[:LEVel][:IMMediate]:LOW?" on page 160	Returns the low level value for the specified port/terminal/connector.
"VOLTage[:LEVel][:IMMediate]:CAConfiguration:LOW" on page 161	Sets the additional low level of the pulse when channel addition mode is active.
"VOLTage[:LEVel][:IMMediate]:CAConfiguration:LOW?" on page 161	Returns the additional low level value for the specified port/terminal/connector when channel addition mode is active.

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 145	Sets the time from the start of the period to the first edge of the pulse.
"PULSe:DELay?" on page 146	Returns the delay value for the specified port/terminal/connector.
"PULSe:WIDTh" on page 146	Sets the width (duration) of the pulse.
"PULSe:WIDTh?" on page 147	Returns the width value for the specified port/terminal/connector.
"PULSe:DCYCle" on page 147	Sets the duty cycle of a repetitive pulse waveform.
"PULSe:DCYCle?" on page 148	Returns the duty cycle value for the specified port/terminal/connector.
"PULSe:HOLD" on page 148	Sets width or duty cycle to hold.
"PULSe:HOLD?" on page 149	Returns the hold parameter valid for the specified port/terminal/connector.
"PULSe:TRANsition[:LEADing]" on page 149	Sets the leading edge value for the specified port/terminal/connector.
"PULSe:TRANsition[:LEADing]?" on page 150	Returns the leading edge value for the specified port/terminal/connector.
"PULSe:TRANsition:TRAiling" on page 151	Sets the trailing edge value for the specified port/terminal/connector.
"PULSe:TRANsition:TRAiling?" on page 152	Returns the trailing edge value for the specified port/terminal/connector.
"PULSe:TRANsition:CAConfiguration[:LEADing]" on page 152	Sets the additional leading edge value for the specified port/terminal/connector, when channel addition mode is active.
"PULSe:TRANsition:CAConfiguration[:LEADing]?" on page 154	Returns the additional leading edge value for the specified port/terminal/connector, when channel addition mode is active.
"PULSe:TRANsition:CAConfiguration:TRAiling" on page 154	Sets the trailing edge value for the specified port/terminal/connector.
"PULSe:TRANsition:CAConfiguration:TRAiling?" on page 156	Returns the additional trailing edge value for the specified port/terminal/connector when channel addition mode is active.
"MUX" on page 156	SetS the frequency multiply factor for the specified port/terminal/connector.
"MUX?" on page 157	Returns the current frequency multiply factor (MUX factor) for the specified port/terminal/connector.

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 179	Controls whether the connectors of the specified port/terminal/connector are opened or closed.
"OUTPut[:STATe]?" on page 180	Returns the current connection state for the connectors of the specified port/terminal/connector.
"OUTPut:CSTate" on page 181	Controls whether the complement connectors of the specified port/terminal/connector are opened or closed.
"OUTPut:CSTate?" on page 182	Returns the current connection state for the complement connectors of the specified port/terminal/connector.
"OUTPut:TVOLtage" on page 182	Sets the external termination voltage.
"OUTPut:TVOLtage?" on page 183	Returns the current programmed value for the external termination voltage for the connectors of the specified port/terminal/connector.
"OUTPut:POLarity" on page 180	Sets the output polarity of the specified data port/terminal/connector to either normal or inverted.
"OUTPut:POLarity?" on page 181	Returns the output polarity of the specified port/terminal/connector.
"OUTPut:IMPedance:EXTernal" on page 183	Sets the external termination impedance, the real load impedance of the DUT.
"OUTPut:IMPedance:EXTernal?" on page 184	Returns the currently set value for the termination (load) impedance.
"OUTPut:TCONfig" on page 184	Selects the termination model for output frontends.
"OUTPut:TCONfig?" on page 186	Returns the termination model for output frontends.
"OUTPut:DIMPedance:EXTernal" on page 186	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 187	Returns the differential termination resistor.
"OUTPut:CAConfiguration[:MODE]" on page 187	Selects the channel-add mode of a port/terminal/ connector.
"OUTPut:CAConfiguration[:MODE]?" on page 188	Returns the current channel addition mode of the specified port/terminal/connector.

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 164	Enables or disables an analyzer input connector.
":INPut[:STATe]?" on page 164	Returns the current status of the specified analyzer input connector.
"INPut:TVOLtage" on page 168	Specifies the termination voltage of an analyzer input connector.
"INPut:TVOLtage?" on page 169	Returns the current termination voltage of the specified analyzer input connector.
"INPut:THReshold" on page 169	Specifies the threshold voltage of an analyzer input connector.
"INPut:THReshold?" on page 169	Returns the current threshold voltage of the specified analyzer input connector.
"INPut:DELay" on page 172	Specifies the delay of the sampling edge.
"INPut:DELay?" on page 172	Returns the current delay of the sampling edge for the specified analyzer input connector.
"INPut:DELay:CYCLe" on page 173	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:CYCLe?" on page 173	Returns the current delay of the sampling edge for the specified analyzer input connector in terms of cycles.
"INPut:DELay:TIME" on page 174	Specifies the delay of the sampling edge.
"INPut:DELay:TIME?" on page 174	Returns the current delay of the sampling edge for the specified analyzer input connector.
"INPut:DELay:ACTual?" on page 175	Returns the actual delay after synchronization.
"INPut:DELay:SWEep" on page 176	Sets the delay sweep value.
"INPut:DELay:SWEep?" on page 176	Returns the current delay sweep value.
"INPut:POLarity" on page 165	Sets the input polarity of the specified data port/terminal/connector to either normal or inverted.
"INPut:POLarity?" on page 165	Returns the input polarity of the specified port/terminal/connector.
"INPut:TYPE" on page 166	Selects how the E4837A differential input frontends are sampled.
"INPut:TYPE?" on page 167	Returns the operation mode of the E4837A differential input frontend.
"INPut:MODE" on page 167	Selects between single-ended termination and differential termination mode.
"INPut:MODE?" on page 168	Returns the termination mode of an input frontend.
"SENSe:VOLTage:RANGe" on page 162	Selects the allowed input voltage range for the E4837A differential input frontend.
"SENSe:VOLTage:RANGe?" on page 163	Returns the currently selected input voltage range of an E4837A differential input frontend.

Command (Link to detailed description)	Description
"INPut:IMPedance[:INTernal]" on page 170	Specifies the internal termination impedance of the analyzer input connector.
"INPut:IMPedance[:INTernal]?" on page 170	Returns the current internal impedance of the specified analyzer input connector.
"INPut:SERial" on page 171	Specifies a serial impedance on this analyzer input connector.
"INPut:SERial?" on page 171	Returns the current serial impedance used for the specified analyzer input connector.
"INPut:TCONfig" on page 177	Selects the termination model for input frontends.
"INPut:TCONfig?" on page 178	Returns the termination model for input frontends.
"INPut:DIMPedance" on page 178	Sets the impedance between the IN and \IN\ inputs of an E4837A frontend when the differential termination model is selected.
"INPut:DIMPedance?" on page 179	Returns the differential termination resistor.

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 201	Controls the output connectors data format of the specified port/terminal/connector.
"FORMat?" on page 202	Returns the current data format state for the output connectors of the specified port/terminal/connector.

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 :IDN?
  - :SYSTem Subsystem
  - :INSTrument Subsystem
  - :INSTrument:HANDle Subsystem

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 miss the underscore)
```

Top-Level Commands

On top level of the DVT subsystem there is only one command available: *":IDN?" on page 59*.

:IDN?

Command :DVT:IDN?

Syntax :DVT:IDN?

Description Identity. Returns the device identity.

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
```

: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"
```

:INSTrument Subsystem

The :INSTrument subsystem provides a mechanism to identify available applications and create handles for virtual instruments (see *"The Command Tree" on page 32*).

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"

:INSTrument:HANDle Subsystem

: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 **<unquoted string>** Suggest value for the Handle

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

<quoted string> 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?`.

Creates a new virtual instrument. A <Character Response Data> element containing the name of the new handle is returned. <Handle> is a <Character Response Data> element which will be used as the handle if possible. If the specified 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"
```

might return `_TEST`

:DVT:INSTrument:HANDle:LIST?

Syntax :DVT:INSTrument:HANDle:LIST?

Get a list of all virtual instrument handles. A comma separated list of <Character Response Data> elements is returned. 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 **<unquoted string>** The handle of the virtual instrument to be destroyed.

Destroy a handle created by CREate? and free all associated resources.

Example
```
DVT:INST:HAND:DEST _TEST
```

:DVT:INSTrument:HANDle:MMEMory:SETTing:EXPort

Syntax :DVT:INSTrument:HANDle:MMEMory:SETTing:EXPort <Handle>, <“FileName”>[,<YDELay | NDELay | ODELay>]

Parameters **<unquoted string>.** The Handle of the virtual instrument.

<quoted string>. 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 exports a setting into a newly created file. The optional parameter filters the cable delay parameters.

Example
```
:DVT:MMEM:SETT:EXP _TEST, "c:\data\81200\export\settings\
withdel.txt", YDEL
```

:DVT:INSTrument:HANDle:MMEMory:SETTing:EXPort?

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

Parameters

<unquoted string> 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 complete setting. The optional parameter filters the cable delay parameters.

:DVT:INSTrument:HANDle:MMEMory:SETTing:IMPort

Syntax :DVT:INSTrument:HANDle:MMEMory:SETTing:IMPort <Handle>, <“Filename”>|<(Expression)>

Parameters

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

<quoted string> The filename of the SCPI like command set, or the command set in form of an expression. The commands have to be listed without the <handle> subnode.

Description This command allows to import a setting into the Agilent 81250 System.

Example

```
:DVT:INST:HAND:MMEM:SETT:IMP _TEST, "test1.txt"
```

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 57*.

For basic information on the command syntax and the command tree, please refer to *"Overview of the SCPI Command Structure" on page 29*.

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

- Full command header
- 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

Administration Commands

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

:<Handle>:IDN?

Syntax :<Handle>:IDN?

Description Identity. Returns the device identity.

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.

After an SGEN:GLOB:INIT:CONT ON this does not necessarily mean that data is already available at the outputs, due to the pipelined system architecture.

Returns 1 when all pending operations are finished.

Example

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

:SYSTem Subsystem

:<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 As 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"
```

:<Handle>:SYSTem:CHECks

Syntax :<Handle>:SYSTem:CHECks <ON | OFF>

Parameters **<ON | OFF>**

This command switches the system internal error and limit checks off or on. Switching off the internal error and limit checks speeds up the downloading of commands, this is recommended when the commands are tested before against any errors or limit conflicts.

Example

```
:_TEST:SYST:CHEC OFF
```

:<Handle>:SYSTem:CHECks?

Syntax :<Handle>:SYSTem:CHECks?

Return Value This query returns the current state of the system internal error and limit checks.

Example
```
:_TEST:SYST:CHEC?
```
might return for example:
```
OFF
```

:<Handle>:SYSTem:CINFormation?

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

Return Value This query returns the system configuration information. The response is a comma-separated list of expressions. With the optional quoted argument a "FileName" can be specified where the configuration information is stored.

Example
```
:_TEST:SYST:CINF? SHOR
```
might return for example:
```
((E4805A,660),(E4841A,660,(E4843A,660,GENERATOR),(E4843A,660,
GENERATOR),(E4844A,660,ANALYZER),(E4844A,660,ANALYZER)),(E4841A,660
,(E4842A,660,GENERATOR),(E4842A,660,GENERATOR),(E4847,330,ANALYZER)
,(E4847A,330,ANALYZER)))
```

:<Handle>:SYSTem:TEST:MODule

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 messages
- SHORt generates a less detailed message

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

Example for a SHORt message:

```
(1, 1, 1, 3, 'E4805', 'Passed', 'E4805 Ver:3.1', 'No error')
```

for a DETailed message:

```
(1, 1, 3, 2, 'E4805', 'Failed', '< AGILENT, E4805, 16Oct97, <#001>,
"", Sp-Bios Version = 3.1, Ur-Bios Version = 1.3 >',
'131116,Sequencer clock divider test 2 failed, 131116,Sequencer
clock divider test 3 failed,131120,Trigger out divider test failed
14,131120,Trigger out divider test failed 15')
```

for a DETailed message for an E4861 module:

```
(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')
```

:<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 messages
- SHORt generates a less detailed message

:<Handle>:MODule: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?

Return Values **BIOS: Update is needed**

BIOS: Update is not needed The module selftest message is replaced by the expected revision.

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

- DETailed generates a more detailed messages
- SHORt generates a less detailed message

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

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

Return Value Returns the first item contained in the message queue. The reported item is deleted and the next execution of this command reports the next message contained in the queue.

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

:<Handle>:SYSTem:MQUeue:LENGth?

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

Return Value Returns the length of the message queue.

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

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

Description The messages, which are generated by the modules at system start up will be moved to 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.

:SYSTem:CLIent Subsystem

:<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 will return 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 the client being blocked. No error code will be 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

Parameters **none**

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" will be returned.

Example

```
:_TEST:SYST:CLI:LOCK
```

:<Handle>:SYSTem:CLIent:BLOCk

Syntax :<Handle>:SYSTem:CLIent:BLOCk

Parameters **none**

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

If another client sends a command to a blocked virtual instrument, the command will not return until the instrument is unlocked. No error message will be returned.

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

Example

```
:_TEST:SYST:CLI:BLOC
```

:<Handle>:SYSTem:CLIent:UNLock

Syntax :<Handle>:SYSTem:CLIent:UNLock

Parameters **none**

Is used to “unlock” or to “set free” the virtual instrument.

Example

```
:_TEST:SYST:CLI:UNL
```

:CONFiguration Subsystem

:<Handle>:CONFiguration

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

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

Return Value Returns the number of Clock GRoups which are available. The (*) specifier is ignored

Related Commands *"[:CGRoup(*)] Subsystem" on page 99*

Example

```
:_TEST:CONF:CGR1?
```

might return

```
1
```

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

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

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

Related Commands *"[:CGRoup(*)] Subsystem" on page 99*

Example

```
:_TEST:CONF:CGR1:MOD?
```

might return

```
3
```

:<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 99*

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 99*

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 111*.

Example

```
:_TEST:CONF:STYP?
```

might return

```
"SCHEME_GENERAL"
```

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

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

Parameters **<quoted string>** Parameter Name

(Expression)

Set the parameter <“Parameter Name”> to the specified value <(Expression)>.

Return Value no return value

Example

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

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

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

Parameters **<quoted string>** Parameter Name

Return Value Returns an expression containing the specified values for <“ParameterName”>.

Example

```
:_TEST:CONF:PROF? "PARAM1"
```

Returns:

```
(10d,10,"TEXT",10u,10f)
```

:<Handle>:CONFiguration:PROFile:REMove

Syntax :<Handle>:CONFiguration:PROFile:REMove <“ParameterName”>

Parameters **<quoted string>** Parameter Name

Removes the specified parameter contained in the “profile”.

Return Value no return value

Example

```
:_TEST:CONF:PROF:REM "PARAM1"
```

:<Handle>:CONFiguration:PROFile:LIST?

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

Parameters **<quoted string>.** Parameter Name

Lists the parameters contained in the “profile”.

Return Value no return value

Example

```
:_TEST:CONF:PROF:LIST?
```

might return:

```
"PARAM1"
```

:MMEMory Subsystem

:<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 **<quoted string>** Absolute Path name

Return Value This query returns an expression containing a comma-separated list of quoted strings. Each string contains one element found in the directory <“Absolute Path”>. If <“Absolute Path”> is an empty string then 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”> doesn’t point to a directory then 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")
```

:<Handle>:MMEMory:INFormation?

Syntax :<Handle>:MMEMory:INFormation? <“Absolute Path”>

Parameters **<quoted string>** Absolute Path name

Return Value Returns detailed information about the specified 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 data base object e.g. Setting, Segment, etc. (quoted string)

Example `:_TEST:MMEM:INF? "C:/"`

Returns the object type of drive ‘C:’, ““ or 0 means not applicable:

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

:<Handle>:MMEMory:SEGMent:LOAD

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

Parameters First parameter

<“FileName”> or <(Expression)> Either specify a file name containing segments or create 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 203*).

Optional second parameter

[ON or OFF] ON is the default, and it means, if a segment name exists already, 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 a specific setting. If the setting name is omitted, then the segments are placed in the global segment pool.

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

Return Value no return value

Example

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

:<Handle>:MMEMory:SEGMent:SAVE

Syntax

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

Parameters First parameter

<"NewFileName"> Specify 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 203*).

Optional second parameter

[<"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 third parameter

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

The command is used to export segments or updated segments to an external mass storage.

Return Value no return value

Example

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

The segment "PAYLOAD" located in the local setting "TEST1" is saved to the file "C:\TRAFFIC.TXT"

:<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 are meant either in the global or local segment pool.

Return Value Returns an expression which contains the contents of segment(s) (vector definition, see *"Segment Import and Export Language" on page 203*). If the "SettingName" and the "SegmentName" is omitted all segments are addressed in the global segment pool. If there is a "SettingName" and "SegmentName" specified, a specific local segment is meant.

Example

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

:<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"
```

:<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 current setting at first switch on of the system 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.

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 Optional first parameter

[<"SettingName">] If not specified the changes will be save into the current setting. At the first time the name of the current setting is "Default".

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

Return Value no return value

Example
```
:_TEST:MMEM:SETT:SAVE "TEST1"
```

:<Handle>:MMEMory:SETTing:NEW

Syntax :<Handle>:MMEMory:SETTing:NEW

Parameters **no parameters**

The command does a reset 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 no return value

Example
```
:_TEST:MMEM:SETT:NEW
```

:<Handle>:MMEMory:SETTing:DELete

Syntax :<Handle>:MMEMory:SETTing:DELete <“SettingName”>

Parameters First parameter

<“SettingName”> Specify the setting to be deleted.

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

Return Value no return value

Example

```
:_TEST:MMEM:SETT:DEL "TEST1"
```

:EDIT:SEGMent(*) Subsystem

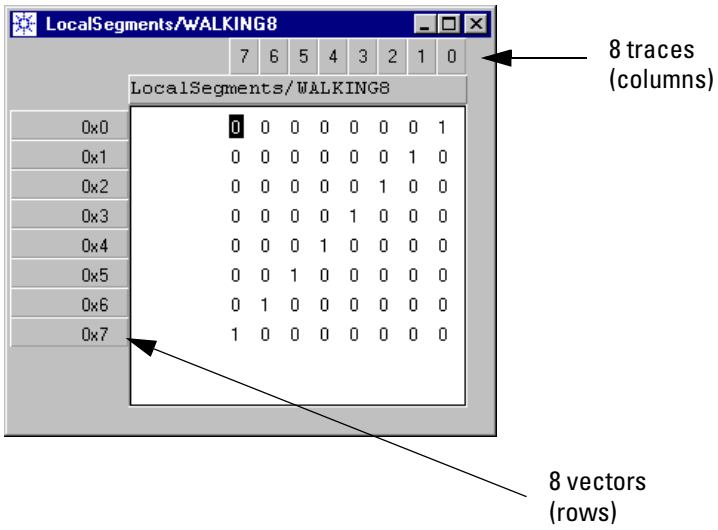
:<Handle>:EDIT:SEGMent(*)

The :EDIT:SEGMent(*) subsystem gives editing capabilities for creating new segments, or updating of existing segments. Data segments represent the pattern to generate for the outputs and the expected data or captured data segments for inputs. 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 which is generated or received at the same clock cycle. Each column in the data segment is called a **trace** which 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 bits long codings up to four states are valid. For one bit long 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 have to open that segment. In return you get an inspector number that will be used for referencing the 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.

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 203*) and import the segments (see *":<Handle>:MMEMory:SETTing:LOAD" on page 79*). 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 which is stored in the Agilent 81250 System’s database.

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/’ 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 then 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.

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. The suffix (*) of SEGMent is ignored.

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 122*). Then stop the system by the command :sgen:glob:init:cont OFF (see “:<*Handle*>:*SGENeral:GLOBal:INITiate:CONTinuous” on page 121*). Now it is possible to open the data segments.

Example

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

might return the following segment inspector index number:

```
3
```

This value is used for the subsequent examples of this command subsystem.

:<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.

This command is used to save segments. When the optional argument is used then it is a "save as" process, if the argument is omitted, then it is a "save" to the current segment name.

(*) Inspector number of the segment.

Example Save the segment "test" which 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"
```

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

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

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

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

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 then 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 create.

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

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

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

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, no 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 about them, but the user can still work with them as normal segments.

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.

Return Value TRUE or FALSe

Description Queries about 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 will be examined first. If the segment is not found there, the GlobalSegments pool will be examined before returning FALSE.

Example

```
:_TEST:EDIT:SEGM:EXIS? "walking8"
```

might return

```
TRUE
```

:<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.

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

Description Returns the absolute path for one segment.

To query about 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 an 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 no argument neither suffix is provided, the inspector number 1 will be used by default.

Example

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

might return

“GlobalSegments/walking8”

```
:_TEST:EDIT:SEGM3:RPAT?
```

might return

“LocalSegments/mysegment”

:<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 arise an error.

The input data 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 88*.

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"
```

:<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.

Return Value Returns the data stream, corresponding to the region specified in the input.

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. Then all the data corresponding to the next trace appears, 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 10 bits 2 bytes are needed, and the trailing 6 bits will be returned as 0s. Then the 2 bytes for the next trace will appear in the data stream.

Every byte is filled starting from the most significative 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 90*.

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

BIN Format #abbbbccccccccccc where…

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

Example

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

might return the following segment data:

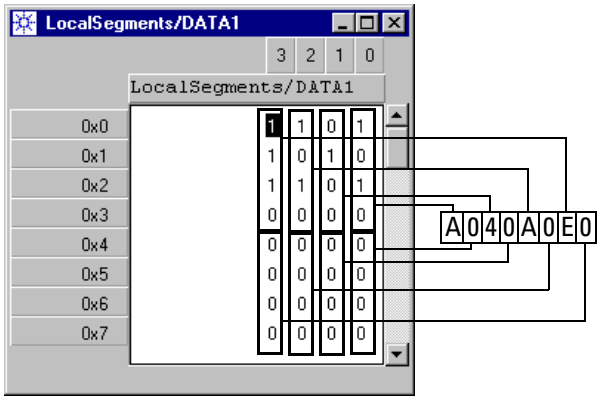
```
A040A0E0
```

```
:_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.



:<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.

If some previous coding has been set in this segment, the new coding needs to have the same length.

New codings can only be set for database segments. Attempting to set a new coding in an analyzer segment will arise 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 (low, don’t care, and high), as typically used for analyzer segments.

  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"
```

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

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

Return Value Returns the symbol coding being used in this segment.

Example

```
:_TEST:EDIT:SEGM3:PATT:COD?
```

might return the following segment coding:

```
"01"
```

:<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
```

:<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 then the exceeding data from the clipboard will be ignored.

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

Example
```
:_TEST:EDIT:SEGM3:PATT:MOD:PASTE 0,76
```

:<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.

The 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 the fifth argument might be whichever from 0 to 3. If a bigger number is provided the highest bits will be ignored and the less significant bits will be used to index the coding.

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

Example

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

:<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 states information in the bounded area.

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

Attempting to invert data in a region in an analyzer segment will arise 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
```

:<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 Rotates the specified block about horizontal (VECTor) or vertical axis (TRACe).

Attempting to rotate data in one region in an analyzer segment will arise an error.

Example :_TEST:EDIT:SEGM3:PATT:MOD:MIRR 0,0,5,10,VECT

:<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, new traces or new vectors.

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

HowMany Number of insertions to do.

(*) 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 added will be initialized to 0.

Attempting to insert new data in an analyzer segment will arise 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 deletions to do.

(*) 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 will arise an error.

Example

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

:<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 Changes the current coding for the segment and converts existing data in the segment according to the specified mapping.

For example, if the old coding was “01” and the mapping is “1x” the conversion will be done by changing all the 0's into 1's and all the 1's into x's. This means all the states in the mapping should belong to the new coding and the mapping should have the same quantity as states as the old coding for the segment.

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

Example

```
:_TEST:EDIT:SEGM3:PATT:MOD:CONV "0 x1","1x"
```

:<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
```

:<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.

Example

```
:_TEST:EDIT:SEGM3:PARA:LIST?
= _Type
```

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

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

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

(*) Inspector number of the segment.

Return Value Returns the value for a specified parameter from the segment. If this command is executed on analyzer segments an error will be raised.

Example
```
:_TEST:EDIT:SEGM3:PARA? "_Type"
= (MEMORY)
```

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

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

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

<(Expression)> New value for the parameter.

(*) Inspector number of the segment.

Description Attempts to set the value for a specified parameter in the segment. If this command is executed on analyzer segments an error will be raised.

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.

(*) Inspector number of the segment.

Description Removes a parameter from the list of parameters in the segment. If this command is executed on analyzer segments an error will be raised.

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.

(*) Inspector number of the segment.

Description Sets the segment type for the current segment.

Example

```
:_TEST:EDIT:SEGM3:TYPE MEM
```

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

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:

:<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, "E4841A", "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:

```
((E4805A,660),(E4841A,660,(E4843A,660,GENERATOR),(E4843A,660,
GENERATOR),(E4844A,660,ANALYZER),(E4844A,660,ANALYZER)),(E4841A,660
,(E4842A,660,GENERATOR),(E4842A,660,GENERATOR),(E4847,330,ANALYZER)
, (E4847A,330,ANALYZER)))
```

[:CGRoup(*)]:MCLock Subsystem

:<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>

Return Value As 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]?

Returns the state of the clock module specified. When ON is returned, this is the “master” clock module in the Agilent 81250 System.

Example

```
:_TEST:cgroup1:mcl:sour?
ON
```

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

:<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. 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, "E4841A", "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:

```
(E4841A,660,DE37700188,DE3700188,(E4844A,660,ANALYZER,0,0,S1),(E4844A,660,ANALYZER,0,0,S1),(E4842A,660,GENERATR,0,0,S1),(E4843A,660,GENERATR,0,0,S1))
```

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

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

Returns the product number in a quoted string.

Example
```
:_TEST:CGRoup1:MOD1:TYPE?
"4831A"
```

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

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

Returns the slot number in which the module is located.

Example
```
:_TEST:CGR1:MOD1:SLOT?
3
```

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

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

Returns the frame number to which this module belongs.

Example
```
:_TEST:CGR1:MOD1:FRAMe?
1
```

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

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

Returns the name of this module.

Example
```
:_TEST:CGR1:MOD1:NAME?
```

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

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

Returns a quoted list of Connector NAMes.

Example
```
:_TEST:CGR1:MOD1:CNAM?
"0102001","0102002","0102003","0102004"
```

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

:<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:

```
(E4844A,660,ANALYZER,0,0,S1)
```

:<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?
"E4843A"
```

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

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

Returns the name of this connector.

Example

```
:_TEST:CGR1:MOD1:CONN1:NAME?
"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 isn’t connected.

Example

```
:_TEST:CGR1:MOD1:CONN1:TNAM?
"Data1"
```

:<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 signal applied to the DUT terminals.

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 a specific module.

Example
```
:_TEST:MOD1:CONN1:CAL:CDEL?
6.500000E-009
```

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

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

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

Sets a zero delay for the specified connector of a specific the Agilent 81250 System's module.

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?
1.000000E-009
```

[: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
```

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

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

Returns the current delay of the trigger output signal.

Example
```
:_TEST:CGR1:TRIG:DEL?
```

It 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:

$$\left(\frac{1}{16} = 0{,}0625\right), \left(\frac{1}{8} = 0{,}125\right), \left(\frac{1}{4} = 0{,}25\right), \left(\frac{1}{2} = 0{,}5\right), 1, 2, 4, 8, 16,$$

The range of this parameter depends on the value of the “system” MUX factor (Segment Resolution).

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.

Example
```
:_TEST:CGR1:TRIG:MUX?
```

```
2
```

:<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:HIGH

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:HIGH <NRf>

Sets the more positive peak of the trigger output signal.

Example

```
:_TEST:CGR1:TRIG:VOLT 2
```

:<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:HIGH?

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:HIGH?

Returns the high voltage level of the trigger output signal.

Example

```
:_TEST:CGR1:TRIG:VOLT?
2.000000E+000
```

:<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:LOW

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:LOW <NRf>

Set the more negative peak of the trigger output signal.

Example

```
:_TEST:CGR1:TRIG:VOLT:LOW -1
```

:<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:LOW?

Syntax :<Handle>[:CGRoup(*)][:SOURce]:TRIGger:VOLTage[:LEVel][:IMMediate]:LOW?

Returns the low voltage of the trigger output signal.

Example

```
:_TEST:CGR1:TRIG:VOLT:LOW?
-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 termination voltage range of the trigger output is -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?

Returns the current termination voltage of the trigger output connector.

Example

```
:_TEST:CGROUP1:SOURCE:TRIGGER:TVOLTAGE?
```

It might return:

```
-2.000000E+000
```

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

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

The external termination IMPedance can be specified by this command. Any negative value is interpreted as into “open”.

Example

```
:_TEST:CGR1:TRIG:IMP 500
```

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

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

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

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?

Returns the actual source used for the trigger signal.

Example

```
:_TEST:CGR1:TRIG:MODE?
```

might return:

```
SEQ
```

:SGENeral Subsystem

:<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 99*.

The following subsystems are available:

- *":SGENeral:INFormation Subsystem" on page 112*
- *":SGENeral:GLOBal Subsystem" on page 113*
- 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 197*.

:SGENeral:INFormation Subsystem

:<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"
```

:SGENeral:GLOBal Subsystem

:<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 118*
- *":SGENeral:GLOBal:INITiate:CONTinuous Subsystem" on page 121*
- *":SGENeral:GLOBal:SYSTem Subsystem" on page 122*
- *":SGENeral:GLOBal:SEQuence Subsystem" on page 123*
- *":SGENeral:GLOBal:TRIGger Subsystem" on page 137*
- *":SGENeral:GLOBal:ARM Subsystem" on page 140*

:<Handle>:SGENeral:GLOBal:CONNect

Syntax :<Handle>:SGENeral:GLOBal:CONNect <ON | OFF>

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
```

:<Handle>:SGENeral:GLOBal:DOFFset

Syntax :<Handle>:SGENeral:GLOBal:DOFFset <DelayOffset>

Parameters **<DelayOffset>.** A numeric value out of the specified range, e.g. 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 currently set delay offset.

Example

```
:_TEST:SGEN:GLOB:DOFF?
```

might return:

```
1.000000E-008
```

:<Handle>:SGENeral:GLOBal:FETCh:ERRor:ANY?

Syntax :<Handle>:SGENeral:GLOBal:FETCh:ERRor:ANY?

Description The query returns 0 for no errors, 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
```

:<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
```

:<Handle>:SGENeral:GLOBal:MUX?

Syntax :<Handle>:SGENeral:GLOBal:MUX?

Return Value The query returns the current 'MUX' factor (Segment Resolution).

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. Valid values for the MUX Segment Resolution are 1, 2, 4, 8, or 16. 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 156*.

Table 1 Clock Rates, Segment Resolution, and Memory Depth for E4832A Modules

System Clock Frequency Mbit/s	Segment Resolution bits	Memory Depth bits	Possible Frequency Multipliers
20.834 – 41.666	1	131,008	1, 2, 4, 8, 16
41.667 – 83.333	2	262,016	1/2, 1, 2, 4, 8
83.334 – 166.666	4	524,032	1/4, 1/2, 1, 2, 4
166.667 – 333.333	8	1,048,064	1/8, 1/4, 1/2, 1, 2
333.334 – 666.667	16	2,097,152	1/16, 1/8, 1/4, 1/2, 1

E4841A modules have only half the memory depth (64 Kbit to 1 Mbit).

Table 2 Clock Rates, Segment Resolution, and Memory Depth for E4861A Modules

System Clock Frequency Mbit/s	Segment Resolution bits	Memory Depth bits	Possible Frequency Multipliers
333.334 – 666.666	16	2,097,152	1, 2, 4
666.667 – 1,333.333	32	4,294,304	1/2, 1, 2
1,333.334 – 2,666.667	64	8,388,608	1/4, 1/2, 1

For more details, see *"How to Set the Clock Frequency"* in the *User Guide*.

The relationships between the different values are shown in the table below.

Table 3 Matrix of Segment Resolution, FMR, Memory Depth and Clock Frequency

Segment Resolution	Frequency Multiplier Range[a]	Memory Depth[b]	System Clock Rates
1 bit (=1)	1, 2, 4, 8, 16	128 Kbit	≤ 41.67 MHz
2 bits (=2)	1/2, 1, 2, 4, 8	256 Kbit	≤ 83.83 MHz
4 bits (=4)	1/4, 1/2, 1, 2, 4	512 Kbit	≤ 166.67 MHz
8 bits (=8)	1/8, 1/4, 1/2, 1, 2	1 Mbit	≤ 333.33 MHz
16 bits (=16)	1/16, 1/8, 1/4, 1/2, 1	2 Mbit	≤ 666 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
```

:SGENeral:GLOBal:CONFigure Subsystem

:<Handle>:SGENeral:GLOBal:CONFigure

In order to 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?

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?
```

It might return:

```
"ECO FAIL"
```

:<Handle>:SGENeral:GLOBal:CONFigure:CAPTure

Syntax :<Handle>:SGENeral:GLOBal:CONFigure:CAPTure

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 happened.

Example

```
:_TEST:SGEN:GLOB:CONF:CAPT
```

:<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 199*.

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.

On E4841A modules the allowed range for the frequency multiplier is half the CAPT or ECO mode.

For example, on an E4844A frontend the allowed range for the frequency multiplier is:

1 <= segment resolution * frequency multiplier <= 16
in CAPT or ECO mode.

1 <= segment resolution * frequency multiplier <= 8
in ECAP or CCAP mode.

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 resulted 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 equipped only with E4841A modules, the minimum number of stop bits is 608.

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 a clock uncertainty a few additional bits (depending on resolution) are recorded.

Example

```
:_TEST:SGEN:GLOB:CONF:ECAP 1024
```

:<Handle>:SGENeneral:GLOBal:CONFigure:CCAPture

Syntax

:<Handle>:SGENeneral:GLOBal:CONFigure:CCAPture

Range/Value Coupling

On E4832A and E4861A modules there is no restriction for the frequency multiplier.

On E4841A modules the allowed range for the frequency multiplier is half the CAPT or ECO mode.

For example, on an E4844A frontend the allowed range for the frequency multiplier is:

1 <= segment resolution * frequency multiplier <= 16
in CAPT or ECO mode.

1 <= segment resolution * frequency multiplier <= 8
in ECAP or CCAP mode.

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:GLOB:CONF?
= CCAP
```

:SGENeral:GLOBal:INITiate: CONTinuous Subsystem

:<Handle>:SGENeral:GLOBal:INITiate:CONTinuous

This subsystem controls the initialization of the trigger subsystem. In other words these command is 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 This query returns 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
```

:SGENeral:GLOBal:SYSTem Subsystem

:<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 This query 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 parameter 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
```

:SGENeral:GLOBal:SEQuence Subsystem

:<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

:<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 } ")" ;

```
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 you 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.

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).

Reset Value

For a system with an E4805A or E4805B clock module:
(3.0,",(LOOP5,0,INF,'00',(BLOCK,0,80,0,'00',())))

For a system with an E4831A clock module:
(3.0,",(LOOP2,0,INF,'00',(BLOCK,0,80,0,'00',())))

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 131*).

Segment-Name Valid keywords depend on the port type and measurement mode:

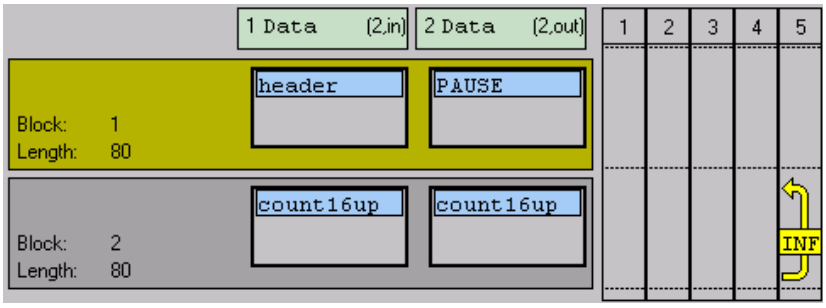
- Output Port: PAUSE0 and PAUSE1
- Input Port, Capture: ACQUIRE and PAUSE
- Input Port, Bit Error Rate: EXPECTED0, EXPECTED1, PAUSE
- Input Port, Compare and Acquire around Error: EXPECTED0, EXPECTED1, DONTCARE and PAUSE.

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:



:<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,(B
LOCK,0,80,"count16up",0,0,"count16up",0,0))))
```

:<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" ;
```

Range/Value Coupling

The command is only available for E4805A/B clock modules.

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))))
```

:<Handle>:SGENeral:GLOBal:SEQuence:EVENTs?

Syntax :<Handle>:SGENeral:GLOBal:SEQuence:EVENTs?

Range/Value Coupling: The query is only available for E4805A/B 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))))
```

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))))
```

:<Handle>:SGENeral:GLOBal:SEQuence:FORCe

Syntax :<Handle>:SGENeral:GLOBal:SEQuence:FORCe

Description The sequence including events and data is downloaded 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:

- 2 for a System with an E4831A clock module,
- 5 for a system with an E4805A/B clock module.

The graphical user interface uses this command to offer 2 or 5 looping levels in the Sequence Editor.

Example

```
:_TEST:SGEN:GLOB:SEQ:LLEV?
```

returns for the E4805A:

```
5
```

:<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.

Value only available for E4805A/B clock modules.

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 Value only available for E4805A/B clock modules.

Returns the state of the program control event line.

Example

```
SGEN:GLOB:SEQ:PCON?
```

Might return:

```
1
```

:SGENeral:GLOBal: SYNChronization Subsystem

:<Handle>:SGENeral:GLOBal:SYNChronization

This command system includes commands and queries to set up the automatic alignment of incomming and expected data.

:<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
```

:<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>

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 incomming 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.

Example `SGEN:GLOB:SYNC:SMOD BSYN`

:<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`

:<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.

Note that this value will only be used if SGEN:GLOB:SYNC:USED? returns TRUE.

Example
```
SGEN:GLOB:SYNC:PACC?
```
might return:
```
2E-1
```

: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>

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

Example

```
:_TEST:SGEN:GLOB:TRIG EXT5
```

:<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 137*.

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 an E4805A/B or E4831A 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:MULTiplier

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
```

:<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]: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
```

:<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?
```

It might return:

```
1.000000E+000
```

:SGENeral:GLOBal:ARM Subsystem

:<Handle>:SGENeral:GLOBal:ARM[:SEQuence][:LAYer]

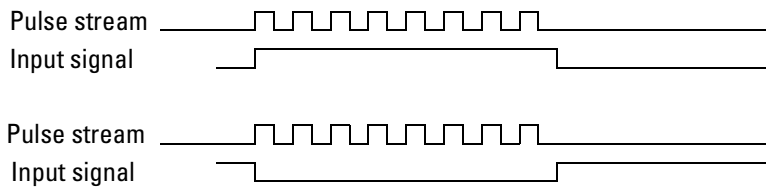
This command subsystem is used to control the system by an external signal.

The EXTernal INPut connector at the front panel of the E4805A/B or E4831A clock module is used to start the system by a so called GATE or by a TRIGGER signal.

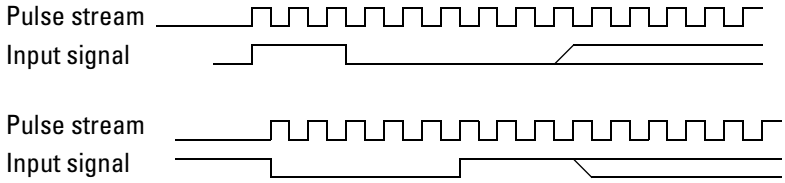
Following modes are available:

...:SOUR:IMM A pulse stream (sequence) is generated immediately as soon as the system is started by the start command.

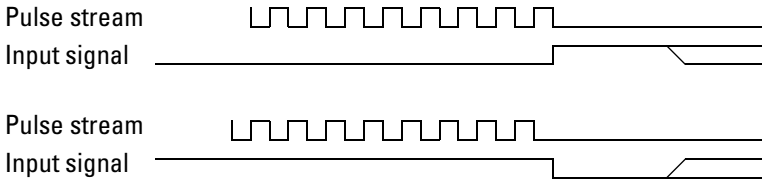
...:SOUR:GATEd A pulse stream (sequence) is generated as soon as the level at the external input connector exceeds the input threshold. A sequence can be generated by a positive or a by a negative input signal depending on the parameter set.



...: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.



...: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

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.

Example

```
:_TEST:SGEN:GLOB:ARM IMM
```

:<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?
```

It might return:

```
PLEV
```

:<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 actual threshold for the external input signal used in the gate or start/stop mode.

Example

```
:_TEST:SGEN:GLOB:ARM:THR?
```

It 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
```

:<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?
```

It might return:

```
-2.000000E+000
```

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.

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>).

Sets the time from the start of the period to the first edge of the pulse. Reference is the trigger output.

Valid range for E4831A, E4832A, E4841A: 0 … max(3µs, PERiod).

Valid range for E4861A: 0 … 300 ns.

Example

```
:_TEST:SGEN:PDAT1:PULS:DEL 5e-9
:_TEST:SGEN:PDAT1:TERM1:SOUR:PULS:DEL 5e-9
:_TEST:SGEN:PPUL1:PULS:DEL 5e-9
:_TEST:SGEN:PPUL1:TERM1:SOUR:PULS:DEL 5e-9
:_TEST:MOD2:CONN3:PULS:DEL 4e-9
```

PULSe:DELay?

Syntax :<Handle>:SGENeral:**PDATa(*):**[:SOURce]:PULSe:DELay?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[:SOURce]:PULSe:DELay?

:<Handle>:SGENeral:**PPULse(*):**[:SOURce]:PULSe:DELay?

:<Handle>:SGENeral:**PPULse(*):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:SGEN:PDAT1:TERM1:SOUR:PULS:DEL?
:_TEST:SGEN:PPUL1:PULS:DEL?
:_TEST:SGEN:PPUL1:TERM1:SOUR:PULS:DEL?
:_TEST:MODULE2:CONNECTOR3:PULSE:DELAY?
```

might return

```
5.000000E-009
```

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
```

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
```

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

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 115*
  - *":<Handle>:SGENeral:GLOBal:FREQuency" on page 115*
  - *"MUX" on page 156*
  - *"FORMat" on page 201*
- 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 146*
  - *"PULSe:DCYCle" on page 147*
  - *"MUX" on page 156*
  - *"FORMat" on page 201*

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
```

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 115*
  - *":<Handle>:SGENeral:GLOBal:FREQuency" on page 115*
  - *"MUX" on page 156*
  - *"FORMat" on page 201*
- 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 146*
  - *"PULSe:DCYCle" on page 147*
  - *"MUX" on page 156*
  - *"FORMat" on page 201*

Example

```
:_TEST:SGEN:PDAT1:PULS:TRAN:TRA 2e-9
:_TEST:SGEN:PDAT1:TERM1:SOUR:PULS:TRAN:TRA
:_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
```

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 187*).

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 115*
  - *":<Handle>:SGENeral:GLOBal:FREQuency" on page 115*
  - *"MUX" on page 156*
  - *"FORMat" on page 201*
- 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 146*
  - *"PULSe:DCYCle" on page 147*
  - *"MUX" on page 156*
  - *"FORMat" on page 201*

NOTE *"PULSe:TRANsition:CAConfiguration:TRAiling" on page 154* holds actually the same value. Both parameters are cross-updated.

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
```

PULSe:TRANsition:CAConfiguration:TRAiling

Syntax

:<Handle>:SGENeral:**PDATa(*)**[:SOURce]:PULSe:TRANsition:CAConfiguration:TRAiling <ChannelAddTrailingEdge>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**[: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>).

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 187*).

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 115*
  - *":<Handle>:SGENeral:GLOBal:FREQuency" on page 115*
  - *"MUX" on page 156*
  - *"FORMat" on page 201*
- 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 146*
  - *"PULSe:DCYCle" on page 147*
  - *"MUX" on page 156*
  - *"FORMat" on page 201*

NOTE *"PULSe:TRANsition:CAConfiguration[:LEADing]" on page 152* holds actually the same value. Both parameters are cross-updated.

Example

```
:_TEST:SGEN:PDAT1:PULS:TRAN:CAC:TRA 2e-9
:_TEST:SGEN:PDAT1:TERM1:SOUR:PULS:TRAN:CAC:TRA 3e-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
```

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:

$$\left(\frac{1}{16} = 0{,}0625\right), \left(\frac{1}{8} = 0{,}125\right), \left(\frac{1}{4} = 0{,}25\right), \left(\frac{1}{2} = 0{,}5\right), 1, 2, 4, 8, 16,$$

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.

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?
```

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
E4842A: –2.20 to 4.30 V
E4843A: –1.75 to 3.10 V
E4846A: –1.45 to 3.50 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
```

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
```

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
E4842A: –2.50 to 3.80 V
E4843A: –2.20 to 2.95 V
E4846A: –1.75 to 3.35 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
```

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
```

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 187*).

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 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
```

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
```

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 E4837A/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
```

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 E4837A/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
```

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]?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):**INPut[:STATe]?

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

Return Value Returns the current status of the specified analyzer input connector.

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 E4837A and 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 E4837A and E4835A frontends.

Example

```
:_TEST:SGEN:PDAT1:INP:POL?
:_TEST:SGEN:PDAT1:TERM1:INP:POL?
:_TEST:CGROUP1:MODULE2:CONNECTOR4:INPUT:POLARITY?
```

might return

```
NORM
```

INPut:TYPE

Syntax

:<Handle>:SGENeral:**PDATa(*)**:INPut:TYPE <BALanced | NORMal | INVerted>

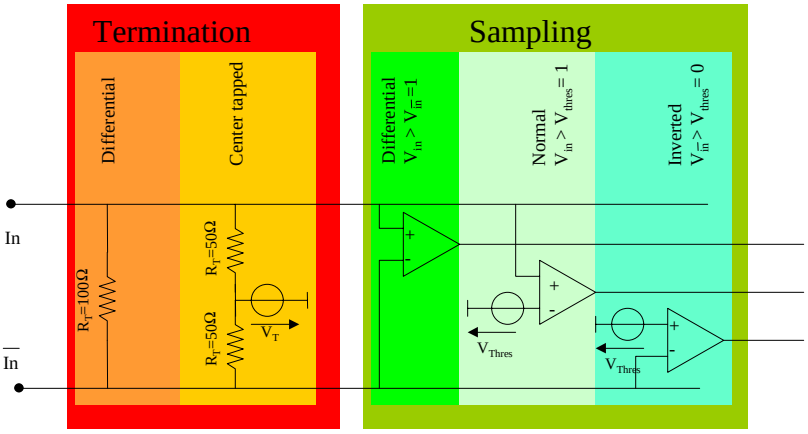
:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:INPut:TYPE <BALanced | NORMal | INVerted>

:<Handle>:CGRoup(*):MODule(*):**CONNector(*)**:INPut:TYPE <BALanced | NORMal | INVerted>

Parameters

<BALanced | NORMal | INVerted> Selects how the E4837A, 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 170* and *"INPut:TVOLtage" on page 168* specify the termination of the input.

In BALanced operation the termination is not used, but is checked against absolute limits. *"INPut:SERial" on page 171* 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
```

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 E4837A, 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

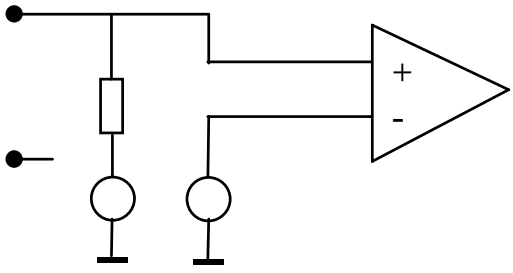
Syntax :<Handle>:SGENeral:**PDATa(*)**:INPut:MODE <DIFF | SNOR | SCOM>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*)**:INPut:MODE <DIFF | SNOR | SCOM>

:<Handle>:CGRoup(*):MODule(*):**CONNector(*)**:INPut:MODE <DIFF | SNOR | SCOM>

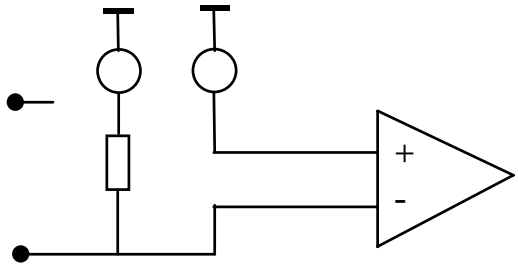
Parameters **<DIFF | SNOR | SCOM>** Selects between single-ended operation and differential operation modes (if supported by the frontend).

- 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, E4837A, E4863A, and E4865A frontends).
- SNOR configures the frontend to single ended operation using the normal input as shown in the following picture (this is the reset value for E4844A, and E4845A frontends):



Note that INPut:TCONfig and INPut:MODE is ignored in this case.

- 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
```

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 value in Volts (<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
```

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 E4837A and 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
```

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. Any negative value is interpreted as “HiZ” (high impedance). This is only supported with the E4847A 330 MSa/s, HiZ, dual input frontend. 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
```

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 zero.

For some electronic devices, which 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
```

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:CYCLe is set to zero.

Otherwise, if the delay is specified by INPut:DELay:CYCLe 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
```

INPut:DELay:CYCLe

Syntax :<Handle>:SGENeral:**PDATa(*):**INPut:DELay:CYCLe <Cycles>

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):**INPut:DELay:CYCLe <Cycles>

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):**INPut:DELay:CYCLe <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:CYCLe 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:CYCLe 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:CYCLe?

Syntax :<Handle>:SGENeral:**PDATa(*):**INPut:DELay:CYCLe?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):**INPut:DELay:CYCLe?

:<Handle>[:CGRoup(*)]:MODule(*):**CONNector(*):**INPut:DELay:CYCLe?

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
```

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:CYCLe 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:CYCLe 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
```

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 three different 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 174*, *"INPut:DELay:CYCLe?" on page 173* and *"INPut:DELay:SWEep?" on page 176*.

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
```

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
```

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 170* and *"INPut:TVOLtage" on page 168*.

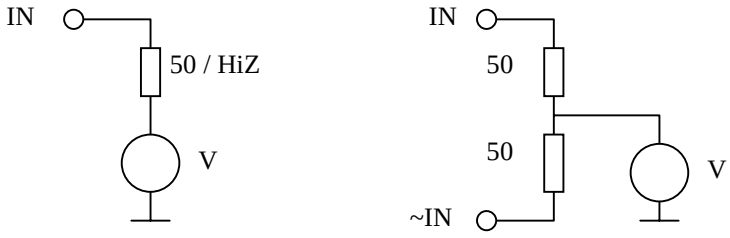
- DIFFerential selects termination via a resistor against the complement input (DIFFerential is only supported for E4835A, E4837A, E4863A and E4865A differential input frontends).

  If DIFFerential is selected, the differential termination resistor can be set with *"INPut:DIMPedance" on page 178*.

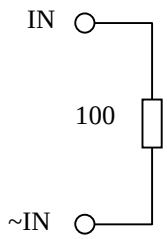
Description

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.



- Via a resistor to the complement input (only available for the differential input frontend E4837A)



Example

```
:_TEST:SGEN:PDAT1:INP:TCON DIFF
:_TEST:SGEN:PDAT1:TERM1:INP:TCON DIFF
:_TEST:MOD1:CONN2:INP:TCON VOLT
```

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 177*).

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, E4837A, E4863A and E4865A frontend when the differential termination model is selected (see *"INPut:TCONfig" on page 177*).

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
```

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 177*).

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?
```

will return

```
100
```

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.

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
```

OUTPut[:STATe]?

Syntax

:<Handle>:SGENeral:**PDATa(*):**OUTPut[:STATe]?

:<Handle>:SGENeral:**PDATa(*):TERMinal(*):**OUTPut:[STATe]?

:<Handle>:SGENeral:**PPULse(*):**OUTPut[:STATe]?

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):**OUTPut:[STATe]?

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

Return Value Returns the current connection state for the specified output connectors.

```
:_TEST:SGEN:PDAT1:OUTP?
:_TEST:SGEN:PDAT1:TERM1:OUTP:STAT?
:_TEST:SGEN:PPUL1:OUTP?
:_TEST:SGEN:PPUL1:TERM1:OUTP:STAT?
:_TEST:MOD2:CONN1:OUTP?
```

might return

ON

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
```

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> Switch 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
```

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 value in Volts (<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
```

OUTPut:IMPedance: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:IMPedance: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:SGEN:PPUL1:TERM1:OUTP:IMP:EXT?
:_TEST:MOD2:CONN3:OUTP:IMP:EXT?
```

might return:

```
500
```

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).

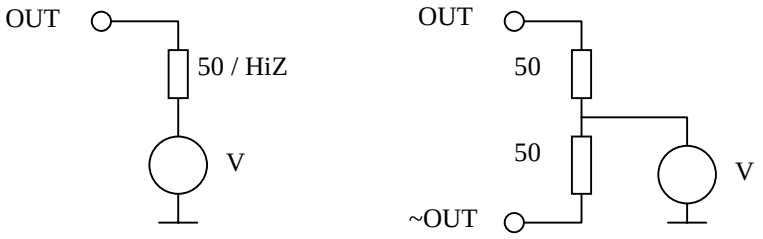
  If VOLTage is selected, the termination resistor and termination voltage can be set with *"OUTPut:IMPedance:EXTernal" on page 183* and *"OUTPut:TVOLtage" on page 182*.

- DIFFerential selects termination via a resistor against the complement output. DIFFerential is only available on the differential output frontends E4838A, E4843A, E4862A and E4864A.

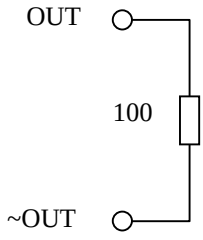
  If DIFFerential is selected, the differential termination resistor can be set with *"OUTPut:DIMPedance:EXTernal" on page 186*.

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.



- 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 184*).

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
```

OUTPut:DIMPedance: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>

CGRoup(*):MODule(*):**CONNector(*)**:OUTPut:DIMPedance:EXTernal <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 184*).

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
:_TEST:MOD2:CONN3:OUTP:DIMP:EXT 20
```

OUTPut:DIMPedance: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 184*).

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
```

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 and E4841A module and on the second frontend slot of an E4831A 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 and E4841A module if the whole module is equipped with output frontends.
- A2 is an analogue add available only for E4838A frontends. It can be used on the second and fourth frontend slot of an E4832A and E4841A module and on the second frontend slot of an E4831A 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 161*

  – *"PULSe:TRANsition:CAConfiguration[:LEADing]" on page 152*

  – *"PULSe:TRANsition:CAConfiguration:TRAiling" on page 154*

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
```

OUTPut:CAConfiguration[: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
```

Port Administration Commands

The following commands are available for administration of 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 145*.

APPend

Syntax

:<Handle>:SGENeral:**PDATa(*):**APPend <"Port Type"> [, <No. of Terminals>] [, <"Port Name">]

:<Handle>:SGENeral:**PPULse(*):**APPend <"Port Type"> [, <No. of Terminals>] [, <"Port Name">]

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.

Creates and APPends a new port to the list of ports. The port type is required. The number of terminals and the port name are optional. The suffix of PDATa and PPULse is ignored.

Example

```
:_TEST:SGEN:PDAT:APP "INPUT_PORT", 4, "DataBus"
:_TEST:SGEN:PPUL:APP "INPUT_PORT", 4, "DataBus"
```

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
```

REName

Syntax :<Handle>:SGENeral:**PDATa(*):**REName <“New Name”>

:<Handle>:SGENeral:**PPULse(*):**REName <“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?
"AddrBus"
```

```
:_TEST:SGEN:PPUL1:NAME?
"AddrBus"
```

TYPE?

Syntax :<Handle>:SGENeral:**PDATa(*):**TYPE?

:<Handle>:SGENeral:**PPULse(*):**TYPE?

Return Value Returns the TYPE of this port.

Example

```
:_TEST:SGEN:PDAT1:TYPE?
"INPUT_PORT"
```

```
:_TEST:SGEN:PPUL1:TYPE?
"INPUT_PORT"
```

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.

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
```

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"
```

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"
```

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

```
"E4846A"
```

MOVE

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"
```

CALibration:CDELay

Syntax :<Handle>:SGENeral:**PDATa(*):TERMinal(*):**CALibration:CDELay <Cable Delay>

:<Handle>:SGENeral:**PPULse(*):TERMinal(*):**CALibration:CDELay <Cable Delay>

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

Sets a cable delay for the specified terminal in the specific port to synchronize the signals at the DUT terminals.

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 specific port.

Example

```
:_TEST:SGEN:PDAT1:TERM1:CAL:CDEL?
:_TEST:SGEN:PPUL1:TERM1:CAL:CDEL?
```

might return

```
6.500000E-009
```

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 103*.

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
```

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)
```

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)`

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>)]

Parameters **<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.

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 the 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"

Syntax Examples:

(@4) specifies terminal 4.

(@3:5,9) specifies terminals 3, 4, 5 and 9.

Example `:_TEST:SGEN:PDAT1:FETC? (@1:3)`

It might return:

```
2.3450000000000E+005,0,2.3300000000000E+005,0,2.3200000000000E+005,
5
```

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

Syntax :<Handle>:SGENeral:PDATa(*):ECOunt:RESet[(<ChannelList>)]

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)`

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 E4846A, 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
```

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
```

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:

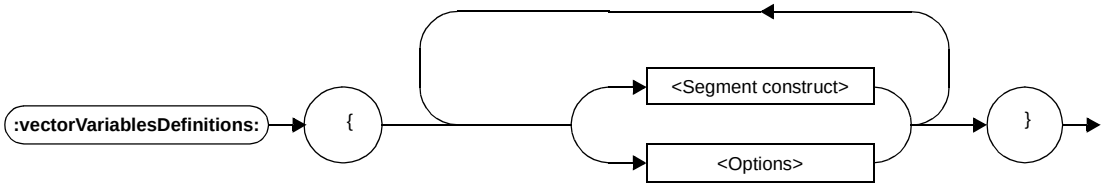
The Language Syntax

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.

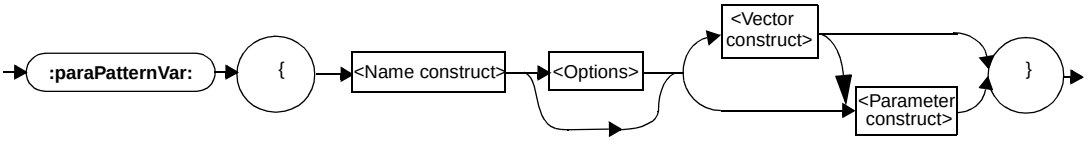
The format of the **Vector Variable construct** is shown below:



<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 206*.

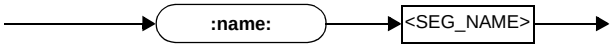
Segment Construct

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:



Name Construct

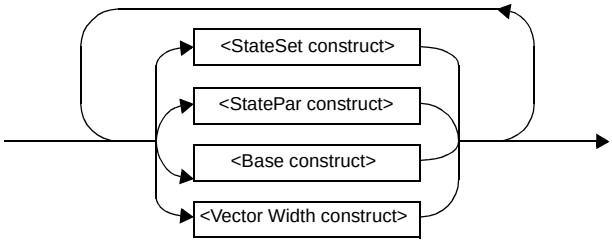
The **Name construct** specifies a name for the segment:



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:

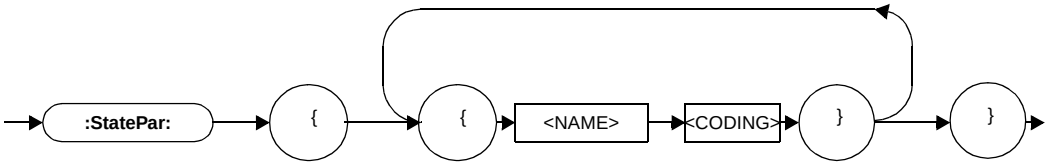


Options defined within a vector variable scope remain valid for the duration of the vector variable construct. However, options may also be specified within a segment construct in which case they are valid only for the duration of the segment construct. As soon as the segment construct goes out of scope the options valid at the vector variable scope re-apply (see *"Scopes" on page 211* for a more detailed definition of scopes).

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.

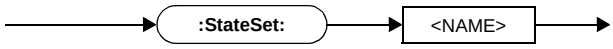
A state set consists of a state set name and its corresponding coding. The state set name is referred to by the StatePar construct:



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 209*

StateSet Construct

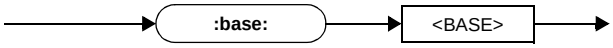
The **StateSet** construct selects a coding previously defined by the StatePar command. The name supplied as an argument indicates which coding should be used:



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)**.

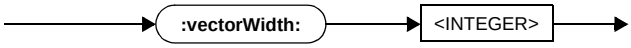


<BASE> is one of **w**, **h**, or **d**. The default base is **w**.

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.

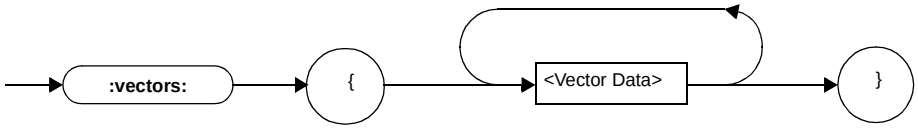
If no vector width statement is present, then the length of the first supplied vector is used as the width for all subsequent vectors:



where <INTEGER> is an integer value between 1 and 1024.

Vector Construct

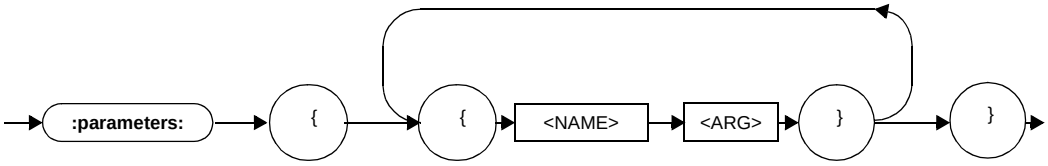
The **Vector construct** specifies the content of connector trace data:



<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 207*.

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 214* 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.

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":

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.

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
```

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:

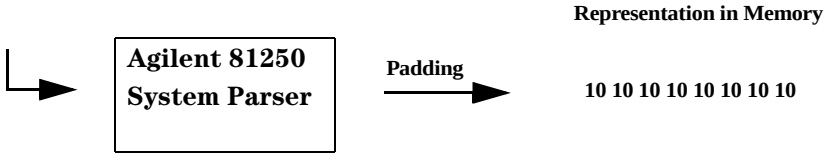
Without Padding

```
: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.

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

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
_Type	(MEMORY)	(PRBS)	(PRWS)
_Rotating	not applicable [n/a]	( )	( )
_Polynom	n/a	(“2^5-1”)...(“2^15-1”)	(“2^5-1”)...(“2^15-1”)
_Logic	n/a	(NORMAL) or (INVERTED)	(NORMAL) or (INVERTED)
_PRxSType	n/a	(PURE) or (ERRORED) or (DENSITY) or (EXTENDED)	(PURE) or (ERRORED) or (DENSITY) or (EXTENDED)
_Errors [when (ERRORED) is selected]	n/a	(0)...(max length of selected PRxS)	(0)...(max length of selected PRxS)
_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”)
_Which [when (EXTENDED) is selected]	n/a	(ZERO) or (ONE)	(ZERO) or (ONE)

Predefined Parameter Names	Memory Segments	PRBS Segments	PRWS Segments
_Number [when (EXTENDED) is selected]	n/a	(0)...(max length of selected PRxS)	(0)...(max length of selected PRxS)

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 216* shows a data pattern segment, a memory type segment.
- *"Example: PRBS Type Segment" on page 217* shows a pure 2^8-1 PRBS with normal output mode, a PRBS segment type.
- *"Example: PRWS Type Segment" on page 217* shows a 2^{10}-1 PRWS with 10 errors inserted and with inverted output mode.

Example: Memory 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
      {                        #star 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 '()'
      }
   }
}
```

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 Code

The following listings provide the complete code for the *"Example C++ Program" on page 37*:

- *"Lib.cpp Interface Class Library Code" on page 220*
- *"Main.cpp Application Code" on page 224*

Lib.cpp Interface Class Library Code

```
//
// 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)
{
   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)
      {
         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);
}
```

```
// 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 (;;)
         {
```

```
            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);
         }
      }
   }
}
```

Main.cpp Application Code

```
// 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];
   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++)
      {
         // 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);
}



// ------------------------------------------------
// 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",
        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
// ------------------------------------------------
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
   //

   // 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);


// 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))
   {
      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];

      // 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;
   }

   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;
}



   // ------------------------------------------------------------
   // 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);
         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);

   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");



// ------------------------------------------------
// 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);
   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
   // 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);


    // 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);
}
```