Standard Commands for Programmable Instruments (SCPI)

Volume 1: Syntax and Style

VERSION 1999.0
May, 1999
Printed in U.S.A.

NOTICE, STATEMENT of INTENT
and
AUTHORIZATION TO COPY

This document defines the Standard-Commands-for-Programmable-Instruments (SCPI) Consortium's SCPI standard. Although the Consortium hereby expressly disclaims any and all warranties whatsoever regarding the SCPI standard, the Consortium has published this document with the intent that the SCPI standard will be seriously considered and adopted, in whole, by the test and measurement marketplace. Consistent with that intent, permission is hereby granted by the Consortium to make copies of this entire document as a whole, PROVIDED THAT this **NOTICE, STATEMENT of INTENT and AUTHORIZATION TO COPY** shall prominently appear on each such whole copy. Further consistent with that intent, permission is hereby granted by the consortium to make copies of portions of this document, absent this **NOTICE, STATEMENT of INTENT and AUTHORIZATION TO COPY**, PROVIDED THAT each such portion-copies shall only be used in a manner which is consistent with the purposes of implementing, promoting, disseminating, or enhancing (as opposed to deviating from) the SCPI standard as herein defined, such as the inclusion of such a portion-copy in an instrument (or instrument-related) product manual which implements SCPI (as defined herein) or in use for design of such an instrument (or instrument related) product. However, under no circumstances may copies of this document, or any portion of this document, be made solely for purposes of sale of the copy or copies.

For information, contact:

Fred Bode, Executive Director
SCPI Consortium
2515 Camino del Rio South, Suite 340
San Diego, CA 92108

Phone: (619) 297-1210
Fax: (619) 297-5955
Email: fbode@vxinl.com
Web Page: http://www.scpiconsortium.org

European SCPI Consortium Contact:

John Pieper
ACEA
P.O. Box 134
7640 AC Wierden
The Netherlands

Phone: +31 546 57 79 94
Fax: +31 546 57 55 75
Email: acea@compuserve.com
Web Page: http://ourworld.compuserve.com/homepages/acea/

Foreword

Commercial computer-controlled test instruments introduced in the 1960s used a wide variety of non-standard, proprietary interfaces and communication protocols. In 1975, the Institute of Electrical and Electronic Engineers approved *IEEE 488-1975*. *IEEE 488* defined a standard electrical and mechanical interface for connectors and cables. It also defined handshaking, addressing, and general protocol for transmitting individual bytes of data to and from instruments and computers. This standard has been updated and is now *IEEE 488.1-1987*.

Although it solved the problem of how to send bytes of data between instruments and computers, *IEEE 488* did not specify the data bytes' meanings. Instrument manufacturers freely invented new commands as they developed new instruments. The format of data returned from instruments varied as well. By the early 1980s, work began on additional standards to specify how to interpret data sent via *IEEE 488*.

In 1987, the IEEE released *IEEE 488.2-1987, Codes, Formats, Protocols and Common Commands for Use with IEEE 488.1-1987*. This standard defined the roles of instruments and controllers in a measurement system and a structured scheme for communication. In particular, *IEEE 488.2* described how to send commands to instruments and how to send responses to controllers. It defined some frequently used "housekeeping" commands explicitly, but each instrument manufacturer was left with the task of naming any other types of command and defining their effect. *IEEE 488.2* specified how certain types of features should be implemented if they were included in an instrument. It generally did not specify which features or commands should be implemented for a particular instrument. Thus, it was possible that two similar instruments could each conform to *IEEE 488.2*, yet they could have an entirely different command set.

Standard Commands for Programmable Instruments (SCPI) is the new instrument command language for controlling instruments that goes beyond *IEEE 488.2* to address a wide variety of instrument functions in a standard manner. SCPI promotes consistency, from the remote programming standpoint, between instruments of the same class and between instruments with the same functional capability. For a given measurement function such as frequency or voltage, SCPI defines the specific command set that is available for that function. Thus, two oscilloscopes made by different manufacturers could be used to make frequency measurements in the same way. It is also possible for a SCPI counter to make a frequency measurement using the same commands as an oscilloscope.

SCPI commands are easy to learn, self-explanatory and account for both novice and expert programmer's usage. Once familiar with the organization and structure of SCPI, considerable efficiency gains can be achieved during control program development, independent of the control program language selected.

Table of Contents

Chapter 6 Program Headers

Chapter 7 Parameters

Chapter 8 Expressions

1 Introduction

1.1 Requirements

This volume, "Syntax and Style," is global in nature and shall be used with all other volumes of the SCPI standard.

1.2 Organization

The first five chapters of *Syntax and Style* are an introduction to the overall concept of SCPI 1999. They describe an overview of the standard, reference other standards, describe SCPI compliance criteria, and how to interpret command tables and syntax flow diagrams. The next three chapters in this volume describe the program headers, parameters and expressions from which SCPI commands and responses are built. Following this are chapters which describe the status reporting model, style guidelines for creating new commands, the effect of *RST on parameter values, and facilities for managing named sets of data in an instrument.

This volume, Syntax and Style, is the first of the four-volume set that makes up the SCPI 1999 Standard. The second and largest volume is the SCPI Command Reference, which contains the actual language constructs which appear in instruments. The third volume, Data Interchange Format, defines a standard representation for data sets which may be used between instruments and applications, between applications, or directly between instruments. The fourth volume, Instrument Classes, defines the SCPI commands and behavior needed to implement functionality sets associated with common classes of instruments. These four volumes form the complete 1999 SCPI Standard and should be used as a set.

This document is intended to apply to "systems" instruments. It defines both organization and content of messages at the controller-to-instrument and instrument-to-controller information interchange level. This document was developed so that, an instrument designed in accordance with it, shall be able to conform to *IEEE Std. 488.1-1987 Standard Digital Interface for Programmable Instrumentation*, and *IEEE Std. 488.2-1987 Codes, Formats and Common Commands For Use With IEEE Std. 488.1-1987*. Conformance to *IEEE 488.1-1987* and *IEEE 488.2-1987* is not required by this document, recognizing that some instruments implement physical interfaces other than the *IEEE 488.1-1987*. However, SCPI is based upon the concepts and terminology used within these standards.

This document may impact the related person-to-instrument and the instrument-to-person information interchange levels. Though the main purpose of this document is to define the interface as it relates to remote control, implementation of these codes and formats may affect the "front panel" of the instrument involved.

1.3 SCPI Goals

The goal of *Standard Commands for Programmable Instruments* (SCPI) is to reduce Automatic Test Equipment (ATE) program development time. SCPI accomplishes this goal by providing a consistent programming environment for instrument control and data usage. This consistent programming environment is achieved by the use of defined program

messages, instrument responses, and data formats across all SCPI instruments, regardless of manufacturer.

A consistent program environment uses the same commands and parameters to control instruments that have the same functionality. These program commands and parameters are sent from a controller to an instrument using *IEEE 488.1,* VXIbus, RS-232C, etc., interfaces. SCPI instruments are very flexible in accepting a range of command and parameter formats, which makes the instrument easier to program. The instrument resonses sent back to the controller can be either data or status information. SCPI instrument response format of a particular query is well-defined and reduces the programming effort to understand instrument data and status information. Data information can be formatted so that it is device- and measurement-independent.

SCPI programming consistency is both vertical and horizontal. Vertical programming consistency defines program messages within an instrument class. An example of vertical consistency is using the same command for reading DC voltage from several different multimeters. Horizontal consistency is using the same command to control similar functions across instrument classes. For example, the trigger command would be the same for an identical trigger function found among counters, oscilloscopes, function generators, etc.

A key to consistent programming is the reduction of multiple ways to control similar instrument functions. The philosophy of SCPI is for the same instrument functions to be controlled by the same SCPI commands. To simplify learning, SCPI uses industry-standard names and terms that are manufacturer and customer supported.

SCPI provides several different levels of instrument control. Simple Measure commands provide users easy and quick control of SCPI instrumentation, while more detailed commands provide traditional instrument control.

SCPI is designed to be expanded with new defined commands in the future without causing programming problems. As new instruments are introduced, the intent is to maintain program compatibility with existing SCPI instruments. SCPI ATE test programs designed to run with new instruments may not be compatible with existing instruments. In other words, the test programs are upward-compatible, but not downward-compatible.

To promote wide industry acceptance, SCPI is publicly available for implementation by anyone, whether or not they are a member of the SCPI Consortium.

The Consortium does not release working documents or provisional documentation. Only approved standards are released to the public.

1.4 SCPI Usage

The advantage of SCPI for the ATE system programmer is reducing the time learning how to program new SCPI instruments after programming their first SCPI instrument.

Programmers who use programming languages such as BASIC, C, FORTRAN, etc., to send instrument commands to instruments will benefit from SCPI. Also, programmers who implement instrument device drivers for ATE program generators and/or software instrument front panels will benefit by SCPI's advantages. SCPI defines instrument

commands, parameters, data, and status. It is not an application package, programming language, or software intended for instrument front panel control.

SCPI is designed to be layered on top of the hardware-independent portion of *IEEE 488.2*. SCPI can be used with controller-to-instrument interfaces such as *IEEE 488.1,* VXIbus, RS-232C, etc.

1.5 Instrument Interchangeability

By providing a consistent programming environment, replacing one SCPI instrument with another SCPI instrument in an ATE system will usually require less effort than with non-SCPI instruments. SCPI is not a standard which completely provides for interchangeable instrumentation. SCPI helps move toward interchangeability by defining instrument commands and responses, but it does not define instrument functionality, accuracy, resolution, connectors, etc., which is needed to provide true instrument interchangeability without affecting the ATE system hardware and software.

2 References

- *"ANSI S1.4"*
- *ANSI X3.4-1977*, American National Standard Code for Information Interchange; ISO Std.646-1983, ISO 7 bit Coded Character Set for Information Interchange
- *ANSI X3.42-1975*, American National Standard Representation of Numeric Values in Character Strings for Information Interchange; ISO Std. 6093-1985, Representation of Numeric Values in Character Strings for Information Interchange
- *ANSI/EIA/TIA-562-1989*, Electrical Characteristics for an Unbalanced Digital Interface
- *ANSI/IEEE Std 181-1977*, IEEE Standard on Pulse Measurement and Analysis by Objective Techniques
- *ANSI/IEEE Std 194-1977*, IEEE Standard Pulse Terms and Definitions
- *ANSI/IEEE Std. 260-1978*, An American National Standard IEEE Standard Letter Symbols for Units of Measurement (SI Units, Customary Inch-Pound Units, and Certain Other Units); ISO Std.1000-1981, SI Units and Recommendations for the Use of Their Multiples and Certain Other Units
- *ANSI/IEEE Std 488.1-1987*, IEEE Standard Digital Interface for Programmable Instrumentation
- *ANSI/IEEE Std 488.2-1992*, IEEE Standard Codes, Formats, Protocols, and Common Commands for use with ANSI/IEEE Std 488.1-1987
- *ANSI/IEEE Std 754-1985*, IEEE Standard for Binary Floating-Point Arithmetic
- *Bell Telephone* "Per BTSM 41004"
- *"CCIR Recommendation 468-2"*
- *"CCITT Recommendation P53"*
- *CCITT Recommendation V.42, Fascile V111.1 (Blue book, 1988)*, Error Correcting Procedures for DCE's Using Asynchronous-to-Synchronous Conversion
- *"Dolby Labs Bulletin No 19/4"*
- *EIA RS-232-D*, Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange
- *EIA RS-422*, Electrical Characteristics of Balanced Voltage Digital Interface Circuits
- *"Fields and Waves in Communication Electronics,"* Ramo, Whinnery, and Van Duzer
- *"IEC Recommendation 179"*

- *IEEE Micro, Volume 8, Number 4, August, 1988, pp 62-76*
- *ISO Std. 2955-1983*, Information processing–Representation of SI and other units in systems with limited character sets
- *"On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform,"* F.J. Harris, Proc. of the IEEE, Vol 66-1, January, 1978, pp 51-83
- *SAE J2264 1995-04* Chassis Dynamometer Simulation of Road Load Using Coast Down Techniques.
- *"Tuning a Control Loop Performance,"* Gregory K. McMillan, Instrument Society of America, ISBN 0-87664-694-1
- *VXI* Consortium INC, VMEbus Extensions for Instrumentation Systems Specification, Revision 2.0

3 Life Cycle

SCPI is a “living” standard. Additional commands will always be needed to meet the needs of new technologies and new instruments. The SCPI Consortium meets regularly to propose and review extensions to the command set. New commands may be proposed by members of the Consortium, and by other interested parties.

Proposals accepted by the Consortium are published and distributed to member companies and are available for immediate use. Approved proposals are reviewed annually. After the annual review, a new revision of the SCPI Standard is published, and the commands become a permenant part of the standard.

All copies of the SCPI Standard contain a version year of issue. All SCPI instruments can be queried to determine the version-year to which they conform.

3.1 Adding a Capability

In general, a command proposed as an extension to SCPI will be approved if it is well formed according to the rules set forth in “Syntax and Style”, if it conforms in style to commands in the subsystem where it is placed, and if there is not already a command to control the same functionality. The Consortium rejects commands only on the basis of objective criteria.

SCPI is designed to grow in an upwardly compatible way. For a control program, this means that the aditions to SCPI will not change the meanings of existing commands, and standard programs will not need to be rewritten except to add new functionality. For an instrument, upward compatibility means that the instrument functionality will not become inaccessable due to additions to the language; existing instruments will not become obsolete by the extension of SCPI.

3.2 Obsoleting a Capability

It is possible that a command may someday have to be altered or deleted in order to permit important new functionality to be implemented. Such an event represents the failure of the extension process because it breaks upward compatibility. Proposals for changes that break upward compatibility will only be accepted if there is overwhelming evidence that the benefits to users and the member companies of the Consortium outweigh the cost to existing users.

The following commands have been deleted from the SCPI Volume II, Command Reference as a part of the SCPI life cycle process. Reuse is discouraged. Any reinstatement is to be accompanied by a careful review of compatibility issues with all previous versions of the SCPI standard.

KEYWORD	PARAMETER FORM	NOTES	DELETION DATE
STATus			1996
:QUEue			1996
:ENABle	<event_queue_list>		1996
[:NEXT]?		[query only]	1996

3.3 Device Dependent Commands

From time to time manufacturers may elect to build instruments with capabilities that are not covered by a current version of this standard. Presumably the manufacturer will submit these new commands into the SCPI Life Cycle process. If the consortium adopts commands that are incompatible those originally used by the manufacturer, the manufacturer is permitted to provide his original commands in subsequent products for compatibility.

4 SCPI Compliance Criteria

This section indicates mandatory and optional capabilities of SCPI. These capabilities are fully specified elsewhere, in "Syntax and Style", "Command Reference", "Data Interchange Format" and *IEEE 488.2*, and an instrument designer shall fully comply with the referenced sections. Only designs conforming to these requirements shall be able to claim conformance to the SCPI specification.

4.1 IEEE 488.2 Requirements

All SCPI instruments shall conform to the specifications for **devices** in *IEEE 488.2*, except that section *4.1 IEEE 488.1 Requirements* shall be omitted where an instrument does not implement an *IEEE 488.1* interface.

Additionally, a SCPI instrument shall be able to parse <compound command program header> and <compound query program header>, to handle the tree structured commands in SCPI.

4.1.1 IEEE Mandated Commands

All SCPI instruments shall implement all the common commands declared mandatory by *IEEE 488.2*.

Mnemonic	Name	488.2 Section
*CLS	Clear Status Command	10.3
*ESE	Standard Event Status Enable Command	10.10
*ESE?	Standard Event Status Enable Query	10.11
*ESR?	Standard Event Status Register Query	10.12
*IDN?	Identification Query	10.14
*OPC	Operation Complete Command	10.18
*OPC?	Operation Complete Query	10.19
*RST	Reset Command	10.32
*SRE	Service Request Enable Command	10.34
*SRE?	Service Request Enable Query	10.35
*STB?	Read Status Byte Query	10.36
*TST?	Self-Test Query	10.38
*WAI	Wait-to-Continue Command	10.39

4.1.2 IEEE Optional Common Commands

IEEE Std. 488.2 describes several optional commands which may be implemented in a device. None of these commands are required by SCPI.

4.1.3 IEEE Common Command Implications

Proper implementation of some of the IEEE 488.2 common commands carry some implications which are not immediately obvious. Primarily this has to do with the synchronization commands.

4.1.3.1 Overlapped and Sequential Commands

IEEE 488.2 defines a distinction between overlapped and sequential commands. As defined in IEEE 488.2, a sequential command is one which finishes executing before the next command starts executing. An overlapped command is one which does not finish executing before the next command starts executing. These types of commands are described in IEEE 488.2, section 12. Examples are given in IEEE 488.2, appendix B.

IEEE 488.2 defines three common commands (*OPC, *WAI, *OPC?) which a device controller can use to synchronize its operation to the execution of overlapped commands. The definitions for these common commands appear in sections 10.18, 10.19, and 10.39 of IEEE 488.2. All three are required by IEEE 488.2, even if none of the device's commands are overlapped.

Implementing any device commands as overlapped will increase the complexity of the implementation of the three synchronization common commands.

Each overlapped command has associated with it a Pending Operation flag. The device sets this flag TRUE when it passes the corresponding command from the Execution Control block to the Device Action block. The device sets the flag false when the device operation is finished, or has been aborted.

IEEE 488.2 defines a No Operation Pending flag. This flag is FALSE whenever any Pending Operation flag is TRUE, and is TRUE whenever all Pending Operation flags are FALSE. IEEE 488.2 also permits device designers to decide which overlapped commands are included in the No Operation Pending flag. A device may implement commands which select which overlapped commands are included in the No Operation Pending flag, although the SCPI standard does not include such a command.

IEEE 488.2 requires user documentation to clearly indicate which commands, if any, are overlapped, and which overlapped commands are included and which are excluded from the No Operation Pending flag.

4.1.3.2 *CLS

This command clears all status data structures in a device. For a device which minimally complies with SCPI, these registers are:

SESR	(IEEE 488.2)
OPERation Status Register	(SCPI)
QUEStionable Status Register	(SCPI)
Error/Event Queue	(SCPI)

Execution of *CLS shall also clear any additional status data structures implemented in the device. The corresponding enable registers are unaffected. See the table in Command Reference, 20.7.

*CLS forces the device into OCIS and OQIS (see 4.1.3.3 and 4.1.3.4) without setting the No Operation Pending flag TRUE and without setting the OPC bit of the SESR TRUE and without placing a "1" into the Output Queue.

For example, suppose a device implements INITiate[:IMMediate] as an overlapped command. Assuming that the trigger model is programmed so that it will eventually return to the IDLE state, and that INITiate[:IMMediate] takes longer to execute than *OPC, sending these commands to this device:

```
INITiate;*OPC
```

results in initiating the trigger model and, after some time, setting the OPC bit in the SESR. However, sending these commands:

```
INITiate;*OPC;*CLS
```

still initiates the trigger model. Since the operation is still pending when the device executes *CLS, the device does not set the OPC bit until it executes another *OPC command.

4.1.3.3 *OPC and *WAI

A device is in the Operation Complete Command Active State (OCAS) after it has executed *OPC. The device returns to the Operation Complete Command Idle State (OCIS) whenever the No Operation Pending flag is TRUE, at the same time setting the OPC bit of the SESR TRUE. See IEEE 488.2 Fig 12-4. The following events force the device into OCIS without setting the No Operation Pending flag TRUE and without setting the OPC bit of the SESR:

- power on
- receipt of the dcas message (device clear)
- execution of *CLS
- execution of *RST

Implementation of the *OPC and *WAI commands is straightforward in devices which implement only sequential commands. When executing *OPC the device simply sets the OPC bit of SESR. Executing *WAI is a no-operation. The device is, in effect, always in OCIS.

In devices which implement overlapped commands the implementation of *OPC and *WAI is a little more complicated. After executing *OPC the device must not set the OPC bit of SESR until the device returns to OCIS, even though it continues to parse and execute commands. After executing *WAI the device must execute no further commands or queries until the No Operation Pending flag is TRUE, or receipt of a dcas message, or a power on.

4.1.3.4 *OPC?

SCPI adds the requirement that *OPC? is implemented as a sequential command.

A device is in the Operation Complete Query Active State (OQAS) after it has executed *OPC?. The device returns to the Operation Complete Query Idle State (OQIS) whenever the No Operation Pending flag is TRUE, at the same time placing a “1" in the Output Queue. See IEEE 488.2 Fig 12-6. The following events force the device into OQIS without setting the No Operation Pending flag TRUE and without placing a ”1" in the Output Queue:

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

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

The implementation of overlapped commands in a device complicates the implementation of *OPC? and places some restrictions on the implementation of the Message Exchange Protocol (MEP). IEEE 488.2 dictates that devices shall send query responses in the order that they receive the corresponding queries (IEEE 488.2 6.4.5.4). Although IEEE 488.2 recommends that *OPC? be the last query in a program message, there is nothing to prevent a controller program from ignoring this suggestion. This is why *OPC? must be sequential.

4.1.3.5 *RST

The “*RST Conditions” chapter of Volume 1 gives pretty a lucid description of the SCPI requirements of *RST. There are, however, some implications.

4.1.3.5.1 Interaction With the Synchronization Commands

*RST stops the execution of any overlapped commands. The natural outcome of this is that the No Operation Pending flag will go TRUE. Execution of *RST must place the device in OCIS before setting the No Operation Pending flag TRUE, so that the effect of any pending *OPC command is nullified (that is, the OPC bit of the SESR does NOT get set).

4.1.3.5.2 Implications For *SAV and *RCL

In devices which implement the optional *SAV and *RCL common commands, the scope of instrument settings affected by these commands shall be the same as that affected by *RST. This means that if *RST has any effect upon a command, then so must *RCL. Conversely, if the device designer wants *RCL to have an effect upon a command, then *RST must also have an effect upon that command. Inversely, and more importantly, if the device designer wants a command to be EXCLUDED from the scope of *SAV and *RCL, then it must also be excluded from the scope of *RST.

4.1.3.5.3 *RST and *RCL as Overlapped Commands

While *RST stops the execution of running overlapped commands, the instrument designer may find it useful to make the *RST command, itself, overlapped. For example, if resetting requires the return to an initial position of a slow-reacting, electro-mechanical device. In this case, performance may be better served with *RST overlapped as subsequent commands may be parsed and executed while the resetting is completing. Even while overlapped, the *RST command shall always complete the resetting of device state variables before subsequent commands are processed to insure the integrity of *RST exception programming. For similar reasons, *RCL may be an overlapped command.

If *RST or *RCL is overlapped, its Pending-Operation flag shall be reported in the No-Operation-Pending flag.

If execution of *RST or *RCL causes a device operation which is equivalent to executing an overlapped command, the device shall set TRUE the Pending-Operation flag associated with that command.

4.1.3.6 *IDN?

IEEE 488.2 is purposefully vague about the content of each of the four fields in the response syntax. SCPI adds no further requirement, but here are some suggestions:

- All devices produced by a company should implement the *IDN? response consistently.
- Field 1, the Manufacturer field, should be identical for all devices produced by a single company.
- Field 2, the Model field, should NOT contain the word “MODEL”.
- Field 4, the Firmware level field, should contain information about all separately revisable subsystems. This information can be contained in single or multiple revision codes.

4.2 SCPI Requirements

IEEE 488.2 describes the syntax of programming and device behavior to a certain level. Further refinement of the syntax and addition rules are imposed by SCPI in order to give instruments a common “look and feel”. The “Syntax and Style” volume of this standard describes the concepts behind SCPI and sets guidelines for originating new commands. A SCPI device shall follow these guidelines and style requirements.

4.2.1 Required Commands

The following commands are required in all SCPI instruments:

Mnemonic	Command Reference Section	Syntax and Style Section
:SYSTem		
:ERRor	21.8	
[:NEXT]?	21.8.3e (see Note 1)	1996
:VERSion?	19.16 (see Note 2)	1991
:STATus	18	5
:OPERation		
[:EVENt]?		
:CONDition?		
:ENABle		
:ENABle?		
:QUEStionable		
[:EVENt]?		
:CONDition?		
:ENABle		
:ENABle?		
:PRESet		

Note 1: The requirement changed from SYSTem:ERRor? to SYSTem:ERRor[:NEXT]? in version 1995.1.

Note 2: Requirement applies for all versions greater than 1990.0

4.2.2 Optional Commands

All other commands in the "Command Reference" are considered optional, depending on the capabilities of the instrument. That is, the control of any instrument capability that is described in SCPI shall be implemented exactly as specified by using the appropriate SCPI defined commands. For example, an instrument dedicated to voltage measurement would not implement commands for sensing frequency. Certain commands, if implemented, require that other commands also be implemented.

When a device does not support all alternative parameter values that are allowed for a SCPI command, it may implement a subset of these values unless otherwise stated. For example, if an instrument implements only RECTangular and UNIForm data shaping for its CALCulate:TRANSform:WINDow command, it may generate an error on receipt of any other SCPI defined parameter value (FLATtop,HAMMing, etc). However, a device must implement all parameters of a multi-parameter SCPI command.

Commands required to implement a SCPI-described capability may be omitted under special conditions. The special condition exists when a command to be implemented affects a part of the instrument in which the configuration is fixed, and that the fixed configuration for that command corresponds to the value it would take on when an *IEEE 488.2* *RST reset command is issued. For example, an instrument with a display that is configured to be permanently on does not need to implement the command that turns the display ON or OFF, since at *RST it is required by SCPI to be ON. If an instrument has a fixed configuration in which a setting does not correspond to the *RST value, then the instrument shall implement the command for that setting even though it has only a single legal value.

4.2.3 Documentation Requirements

The documentation for a SCPI instrument shall list the version number for which the instrument complies. This information shall appear on instrument specification sheets and related documents, as well as the programming manual.

The manual for a SCPI instrument shall have a separate section titled *"SCPI Conformance Information"*. This section shall list separately:

- The SCPI version to which the instrument complies
- The syntax of all SCPI confirmed commands implemented by the instrument. Confirmed commands are those commands which are published in the latest version of SCPI to which the instrument conforms. Commands in the DIAGnostic subsystem, and other commands which are not intended for end-user use need not be documented in this section.
- The syntax of all SCPI approved commands implemented by the instrument. Approved commands are those commands which have been approved by the SCPI Consortium, but are not contained in the version of SCPI to which the instrument conforms.
- The syntax of all commands implemented by the instrument, which are not part of

the SCPI definition.

5 Notation

Extensive use is made of syntax flow diagrams and tables throughout this document. Associated with these are notational styles, which are described in this chapter.

5.1 Interpreting Command Tables

Command tables are used to define a set of SCPI commands. A table shows the commands, their hierarchical relationships, related parameters (if any), and associated notes. The table is broken into 3 columns; the KEYWORD, the PARAMETER FORM, and any NOTES.

The KEYWORD column provides the name of the command. The actual name of the command consists of one or more keywords since SCPI commands are based on a hierarchical structure, also known as a **tree system.** In such a system, associated commands are grouped together under a common node in the hierarchy, analogous to the way leaves at a same level are connected at a common branch. This and similar branches are connected to fewer and thicker branches, until they meet at the root of the tree. The closer to the root, the higher a node is considered in the hierarchy. To obtain a particular leaf or a particular command, the full path to it must be specified. This path is represented in the tables by placing the highest node in the hierarchy in the left-most position. Lower nodes in the hierarchy are indented one position to the right, below the parent node.

Square brackets ([]) are used to enclose a keyword that is optional when programming the command; that is, the instrument shall process the command to have the same effect whether the option node is omitted by the programmer or not. Such a node is called a **default node.** Letter case in tables is used to differentiate between the accepted short form (the uppercase characters) and the long form (the whole keyword). The significance of the short and long form keywords is explained in the “Program Headers” chapter.

The PARAMETER FORM column indicates the number and order of parameters in a command and their legal values. A command may allow the use of a SCPI-defined parameter type, a literal, or a combination of the two. The SCPI-defined parameter types are described in the “Parameters” chapter, and are distinguished by enclosing the type name in angle brackets (<>). A literal is typically a word that enumerates a setting that cannot be described using the SCPI-defined parameter types.

In the PARAMETER FORM column, a number of characters have special significance. Square brackets ([]) are used to enclose one or more parameters that are optional when controlling the instrument. Braces ({}), or curly brackets, are used to enclose one or more parameters that may be included zero or more times. The vertical bar (|) can be read as “or” and is used to separate alternative parameter options.

The query form of a command is generated by appending a question mark to the last keyword. However, not all commands have a query form, and some commands exist only in the query form. The NOTES column is used to indicate this.

As an example, suppose the existance of the following table.

KEYWORD	PARAMETER FORM	COMMENTS
:FREQuency		
[:CW]	<numeric_value>	
:AUTO	<Boolean>	
:CENTer	<numeric_value>	
:SPAN	<numeric_value>	

To set the frequency value for a Continuous Wave signal, the following command would be sent:

```
FREQuency:CW  2000000
```


Alternatively, since the CW node is optional the following command is also valid:

```
FREQuency  2000000
```

To set the value of AUTO, either of the following commands are allowed, again due to the optional CW node:

```
FREQuency:CW:AUTO  OFF
```

```
FREQuency:AUTO  OFF
```

The query form of the AUTO command is either of the following:

```
FREQuency:CW:AUTO?
FREQuency:AUTO?
```

In the "Command Descriptions," the detailed descriptions of the individual commands are provided immediately after the command table, in the order in which the commands appear in the table, reading from top to bottom.

5.2 Interpreting Syntax Flow Diagrams

Syntax flow diagrams are used extensively throughout the SCPI document and *IEEE 488.2* to provide pictorial representations of syntax. These representations are also known as **railroad diagrams**.

The flow through a diagram is given by lines and arrows. These link together the various objects used to form a command. Objects exist in the diagram as either a circle or a box. Circles indicate literal characters and the boxes represent a syntactic structure that is defined elsewhere in this document or in *IEEE 488.2*. Flow through the diagrams generally proceeds left-to-right. Diagrams are entered on the left, and the syntax is satisfied when the diagram is exited on the right. When an element or group of elements in the diagram is repeatable, a reverse, right-to-left path will be shown around and above the element(s), and is marked with a left-facing arrow. When an element or group of elements in the diagram are optional, a left-to-right bypass path will be shown around and below the element(s). A branch in the path indicates a choice of elements.

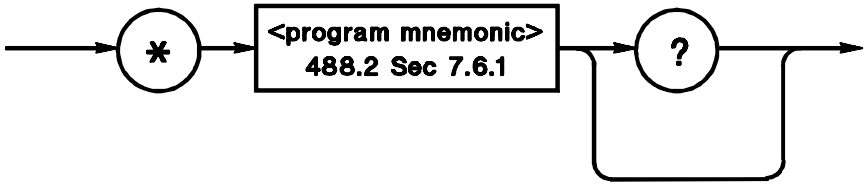
Lowercase and uppercase letters are considered equivalent; however, no attempt has been made to specify both alternatives in every instance. Whitespace characters (as defined in *IEEE 488.2*, section 7.4.1.2) are optional in many places, and no attempt has been made to specify "optional whitespace" everywhere it may exist.

6 Program Headers

Program headers are keywords that identify the command. The program headers follow the syntax described in section 7.6 of *IEEE 488.2*. Instruments shall accept both upper and lowercase characters without distinguishing between the cases. Program headers consist of two distinct types, common command headers and instrument-control headers.

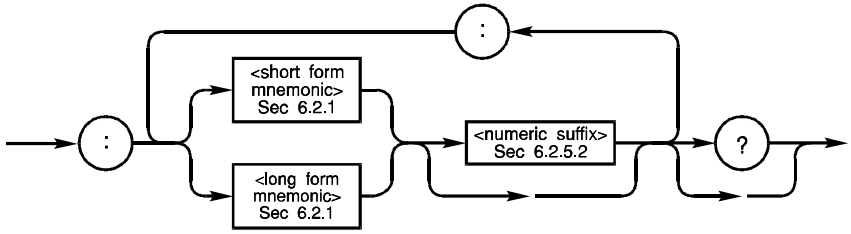
6.1 Common Command and Query Headers

The common command and query program header syntax is specified in *IEEE 488.2* for use with the *IEEE 488.2*-defined common commands and queries. Their syntax is:



6.2 Instrument-Control Headers

Instrument-control headers are used for all other instrument commands, typically those related to source and measurement control. The syntax for instrument-control headers is:



The definition of the effect of the colons in an instrument-control header is covered later in "Traversal of the Header Tree."

6.2.1 Mnemonic Generation Rules

Each instrument-control header or keyword has both a long and a short form. A SCPI instrument shall accept only the exact short and the exact long forms. Sending a header that is not the short form, nor the complete long form to a SCPI instrument shall cause it to generate an error. *IEEE 488.2* limits the length of a header to 12 characters, including any numeric suffix that may appear. The long form header is either a single word or an abbreviation of a phrase. The short form header is an abbreviation of the long form header. In order to maintain a consistent set of mnemonics, SCPI defines the rules to generate a mnemonic.

The long form mnemonic is generated from either a single word or a phrase. If a single word is used, then that word becomes the mnemonic. If a phrase is used, then the mnemonic is the

first letter of each word and the entire last word. For example the phrase “relative velocity” would generate RVELOCITY as the long form command header.

The short form mnemonic is usually the first four characters of the long form command header. The exception to this is when the long form consists of more than four characters and the fourth character is a vowel. In such cases, the vowel is dropped and the short form becomes the first three characters of the long form. For example, the short form of FREE is FREE, however, the short form of SWEEP is SWE. Note that elsewhere in this document a special notation is employed to differentiate the short form keyword from the longform of the same keyword. The long form of the keyword is shown, with the short form portion shown in uppercase characters, and the rest of the keyword is shown in lowercase characters. Thus Relative VELocity keyword would be shown as RVELocity.

The short form generation rules imply that phrases such as “Jump Start” and “Jump Stop” are not allowed. Although the long forms JSTART and JSTOP are unique, the short form is the same in both cases, “JST.” This can be overcome by changing the phrases to “Jump Begin” and “Jump End”, thus creating unique long and short forms. Alternatively, the mnemonic JUMP can become an additional level in the tree, allowing STARt and STOP to become their own mnemonics, giving JUMP:STARt and JUMP:STOP.

The mnemonic generation rules allow keywords such as “TIME” and “TIMer” at the same tree level, since these are unique in both forms. This is not recommended since an unfamiliar user may select the wrong function unintentionally. If the two functions have the same number and type of parameters, the instrument may not signal an error, further exacerbating the situation.

All instrument command headers are allowed a numeric suffix to differentiate multiple instances of the same structure, such as multi-channel instruments. The numeric suffix is applied to both the long and short forms. For example, TRIG1 is the short form of TRIGger1. A numeric suffix of 1 is implied on all instrument command headers that do not explicitly define a suffix; thus, TRIG is equivalent to TRIG1.

As a general rule, a mnemonic should contain no digits. Ending a short or long form mnemonic in a digit creates an ambiguity in separating the mnemonic from any added numeric suffix.

6.2.2 Building the Command Tree

SCPI commands are based on a hierarchical structure. This allows the same instrument-control header (keyword) to be used several times for different purposes, providing that the mnemonic occurs in a unique position in the hierarchy. That is, the mnemonic does not collide with any other mnemonics at the same level, or one level different where a default node exists.

Using hierarchical commands should eliminate the need for most multiword mnemonics. The use of multiword mnemonics is discouraged and they should only be used when:

A). The use of the hierarchical scheme would add several additional layers to the tree, and

B). No further growth at the level of those potentially intermediate nodes is anticipated.

These style guidelines should act as a starting point for creating new commands.

1. The lowest nodes should have the broadest base possible. For example, the center frequency command should be FREQuency:CENTer rather than CENTer:FREQuency. Since center is an adjective to the noun frequency, the tree will build with fewer duplicate nodes if FREQuency:CENTer is chosen.

2. If a function is broad-based and thought of as a separate subsystem, make it a separate subsystem even if it could properly be classified under another subsystem. For example, bandwidth is a separate subsystem, even though it could be classified under frequency. Bandwidth is generally thought of as a separate subsystem and not classified with other frequency commands by customers.

3. Keep the tree as shallow as possible, usually three levels or less. If the tree for a given implementation is considerably deeper than it is wide, something may be wrong with the command structure.

4. Some nodes may be made optional. When choosing to define a node as the default at a particular level, be sure that it is the most common choice at that level. Also, check carefully for conflicting keywords, since all nodes below a default are effectively promoted by one level when the default node is assumed.

5. Some root keywords are optional, depending on the instrument capabilities. In an instrument which is primarily a source, the root mnemonic [SOURce] is optional. [SENSe] is optional for instruments which are primarily sensors, and [ROUTe] is optional for scanners. This is at the discretion of the device designer. For horizontal compatibility purposes, the instrument should accept this node if it is sent as part of a program message.

6. A complete tree path shall be unique within SCPI. Thus, if a path is assigned a particular function in any instrument, it must perform the same function in all instruments which implement that path.

7. SCPI also discourages the use of different keywords performing the same function. This problem is more difficult because of industry-standard terminology. A good example is setting the output level on a source. Microwave sources call this power level, RF sources use amplitude level, and power supplies use voltage level, yet the function is the same in all three. One of the tools available is to provide aliases for the function in order to provide both sets of terminology. Whenever possible, SCPI shall implement as aliases those instances where duplicate names are necessary. Where aliases are used, one term shall be defined which is the primary keyword. This keyword shall appear in all instruments.

8. The tree structure shall follow the general form of the instrument model described in the "Command Reference."

9. Implementations shall use the language constructs described in the "Command Reference," if a construct exists for the desired feature. If the feature does not exist, the

designer is responsible for following the life cycle described in an earlier chapter in order to integrate the capability into SCPI.

10. The value or syntax of a parameter shall not change the type or number of other parameters in that command. If such a condition is encountered, additional nodes should be added to the tree.

For example, SCPI shall not include the structures such as:

```
:SYSTem
  :BEEPer      1,<frequency>
  :BEEPer      2,<time>
```

Rather, SCPI uses a structure such as:

```
:SYSTem
  :BEEPer
    :FREQuency <numeric_value>
    :TIME      <numeric_value>
```

11. In general, parameters should only appear at the leaf nodes of the tree. In the undesirable example below, the command to set a single frequency is placed at a different level to the commands that are used for setting a range (span) of frequencies. Conceptually equivalent level commands are being placed at different levels, leading to possible misconceptions in use. "Optional" nodes should be created as required to accomplish the same task. For example, suppose an instrument superficially wants the following set of commands:

```
FREQuency      <numeric_value>
  :CENTer      <numeric_value>
  :SPAN        <Boolean>
```

The SCPI implementation shall overcome this by creating a node under frequency that represents a command to set a single frequency, such as CW (Continuous Wave). Causing all the command parameters to be associated with leaf nodes. To retain simple operation as in the case of setting a single frequency, the CW node can be made optional, providing it does not create a keyword conflict. Thus our example becomes:

```
FREQuency
 [:CW]         <numeric_value>
  :CENTer      <numeric_value>
  :SPAN        <Boolean>
```

Exceptions to this guideline are the AUTO and STEP constructs defined in this document. The AUTO and STEP commands are formed from the leaf node for the value. Other exceptions may be defined. In these cases, the exception shall be noted in the command structure of the language document.

Nodes should not be added to a command tree simply to make the tree have a uniform depth. If the only usable names for a terminal seem not to have any meaning (for instance, VALue), the addition may not be helpful and the tree should be implemented unbalanced. An example

of this is the SOURce:FREQuency:SPAN node, which has the commands HOLD, LINK, and FULL beneath it.

12. If a particular device function is not alterable or does not exist, then the associated SCPI command need not be implemented unless the value of the function is different than the defined *RST value for that function. If the *RST value is device-dependent, then the function need not be implemented. For example:

A. A source that has no amplitude modulation does NOT have to implement AM:STATe OFF because AM:STATe OFF is specified at *RST.

B. A device that only sweeps in frequency (and has no CW function) must implement FREQ:MODE SWEep because FREQ:MODE is specified as CW at *RST. However, it is permissable to allow only SWEep as a parameter to FREQ:MODE.

C. A device that produces a single, nonalterable CW frequency of 100 MHZ need not accept FREQ[:CW] 100 MHZ since the *RST value of FREQ:CW is device-dependent.

The intent of this rule is to allow common features with varying complexities to be accessed. This rule specifies the minimum conditions of including a node. However, designers are encouraged to include additional degenerate nodes which are important to their market. In example A above, a designer may include AM:STATe OFF as a command, and is encouraged to do so if the instrument is being targeted into markets where AM modulation is a typical feature of instruments in this market.

13. A new command may not be added if there is already a command to control the same functionality. This rule insures two instruments will not be needlessly incompatible when designers choose arbitrarily among multiple commands available to control a particular function. (Notice that SCPI assures compatibility when aliased commands are implemented by making one path required). A new command may not be added to the standard unless it can be shown to control functions not controlled by any existing commands. The following guidelines may help in determining whether a command controls new functionality or represents illegal duplication, and guide selection of one command over another.

Does the setting of the new command affect the measurement result returned?

Signal preconditioning commands (like ATTenuator, FILTer, GAIN) appear in the INPut subsystem and in SENSe. These commands are not aliases; The setting of INPut:ATTenuator changes the reported signal level, while SENSe:ATTenuator would be compensated for so the reported signal level would not be affected. Since these commands affect the returned value differently, they control different functions and are not duplicate controls.

Does the new command control a function with a different purpose than any existing command?

Numerical post-processing commands (like WINDow, AVERage, SMOothing) appear in the sensor and also in CALC. The function properly belongs in the sensor when its purpose is to

enhance accuracy or correct for artifacts introduced by the sensor. The function belongs in CALCulate when its purpose is to change the presentation of the data or provide analytical tools not directly related to measurement.

Does the command control measurement hardware?

Some functions (like SMOothing) may be performed directly on the signal in hardware or in software after A/D conversion. While this rule can be difficult to apply, the fact that a command controls measurement hardware often indicates that it belongs in the sensor rather than CALCulate.

Does it make sense if the same operation may be done twice?

Another way to tell that a command controls new functionality is if it makes sense for the controlled function to appear twice in a single instrument. The AVERage subsystem is a good example; it is not unreasonable to build an instrument that averages several readings to improve the accuracy of the measurement, and also provides averaging as a statistical function in the CALCulate subsystem.

14. In general, SCPI commands which are only events without a query form do not have any parameters. Including a parameter might mislead a user into believing the value could be queried as most commands in SCPI have a query form, see 6.2.3.

An exception to this guideline is when the action is performed on a named object within the instrument and these objects cannot be enumerated. For example, <file_name>, <trace_name>, and <data_handle> are objects whose values cannot be entirely listed within SCPI. The user can create any number of <file_names> or <trace_names>. The signal routing concepts in SCPI allow a large number of possible <data_handles>.

6.2.3 Queries

All commands, unless otherwise noted, have an additional query form. As defined in *IEEE 488.2*, a query is a command header with a question mark symbol appended (for example, FREQ:CENT?). When a query form of a command is received, the current setting associated with the command is placed in the output buffer. Query responses do not include the command header. Execution of a query form has no side effects. It shall not cause any settings or couplings within the instrument to change. An exception is in MEASurement subsystem, where instrument settings may change as a result of the measurement.

Both the command and query forms shall be implemented unless a comment or note indicating otherwise appears in the keyword summary. Even when a command accepts only a single selection, both the command and query forms shall still be implemented. A note of the form “[event; no query]” or “[no query]” indicates that only the command form and not the query form shall be implemented. A note of the form “[query only]” indicates that only the query form and not the command form shall be implemented.

When character data is used for a parameter, both longform and shortform values shall be recognized. If the command has a query form with character response data, the shortform value is always returned.

When numeric parameters are queried, the result shall be returned in fundamental units unless otherwise specified or requested. When several different units may be considered fundamental (for example, dBuV or dBm), the units of the returned result shall be documented in the query response description for the command. A query may have a related :UNITs node to change the default units of the accepted or returned value, where this exists the coupling shall be clearly indicated in the command description.

6.2.4 Traversal of the Header Tree

IEEE 488.2 allows the designer some discretion on how compound headers are handled within the rules of section 7.6.1.5. SCPI imposes additional requirements regarding how a compound command is parsed.

Multiple <PROGRAM MESSAGE UNIT> elements may be sent in a <PROGRAM MESSAGE>. The first command is always referenced to the root node. Subsequent commands, however, are referenced to the same tree level as the previous command in a message unit. SCPI parsers shall follow the example presented in *IEEE 488.2,* section A.1.1. A SCPI parser shall NOT implement the enhanced tree walking implementation described in section A.1.2.

For example, if a hypothetical instrument had the command tree:

```
FREQuency
  :STARt          <numeric_value>
  :STOP           <numeric_value>
  :SLEW           <numeric_value>
    :AUTO         <Boolean> | ONCE
  :BANDwidth      <numeric_value>
POWer
  :STARt          <numeric_value>
  :STOP           <numeric_value>
BAND              A|B|C|D
```

Then the following commands shall behave as described:

- FREQ:STAR 3 MHZ;STOP 5 MHZ<nl> shall set the start frequency to 3 MHz and the stop frequency to 5 MHz.
- FREQ:STAR 3 MHZ;:FREQ:STOP 5 MHZ<nl> shall set the start frequency to 3 MHz and the stop frequency to 5 MHz.
- FREQ:STAR 3 MHZ;POW:STOP 5 DBM<nl> may set the start frequency to 3 MHz and shall generate an error because POW is not a node at the current parser level.
- FREQ:STAR 3 MHZ;SLEW:AUTO ON<nl> shall set the start frequency to 3 MHZ and couple the frequency slew rate.
- FREQ:SLEW:AUTO ON;STOP 5 MHZ<nl> may couple the slew rate and shall generate a command error since the coupling command is not at the same level as the stop command.

- FREQ:SLEW 3 MHZ/S;AUTO ON<nl> may set the frequency slew rate to 3 MHZ/S and shall generate a command error because :AUTO is not at the same level.
- FREQ:START 3 MHZ;BAND 1 MHZ<nl> shall set the start frequency to 3 MHz and the bandwidth to 1 MHz.
- FREQ:START 3 MHz;:BAND A<nl> shall set the start frequency to 3 MHz and select band A.
- FREQ:SLEW:AUTO ON;3 MHZ/S<nl> may couple the frequency slew rate and shall generate a syntax error, because *IEEE 488.2* specifies that headers must be sent with each command.

Default nodes in the tree shall not alter the header path of the parser in order to follow the *IEEE 488.2* rules for compound headers. *IEEE 488.2*, section 7.6.1.5 makes no mention of default nodes, so adding something to the header path is not allowed. For example, if a hypothetical instrument had the command tree:

```
DISPlay
  [:STATe]      <Boolean>
   :DATA        <string>
```

Then the following situations would have these results:

- DISP:STAT ON;DATA “Hello, world!”<nl> shall turn the display on and display “Hello, world!”.
- DISP ON;DATA “Hello, world!”<nl> may turn the display on and shall flag an error (assuming that DATA is not a root mnemonic).

6.2.5 Multiple Capabilities and Numeric Keyword Suffixes

Many instruments provide multiple capabilities, by duplicating internal functional blocks. One example is an instrument that can display more than one measurement at a time. Another example is an instrument that can make a particular measurement on one of several input channels. SCPI provides a mechanism to address this duplicate functionality while retaining the same hierarchy for each duplication.

6.2.5.1 Single Instrument with Many Electrical Ports

When only electrical ports (front panel terminals) are involved, the <channel_list> notation is sufficient to describe the measurement connection. An instrument which needs to select one or more of several input terminals can be logically modeled as a simple instrument with one set of inputs connected to a scanner (signal multiplexor).

6.2.5.2 Multiple Identical Capabilities

When an instrument has several independent measurement channels (or other capabilities, such as displays or sources), the selection of which to use is designated by a numeric suffix attached to a program mnemonic. If the suffix is absent, the command is treated as being designated for capability number one. In this way, channel one of a multiple channel instrument can accept the same commands as a single channel instrument to allow for

upward compatibility. Further, adding multiple channels or capabilities to an instrument class where multiple channels were not anticipated becomes possible.

The node that contains the numeric suffix is the one where the multiple capability occurs. This may be at any level of the tree: root, intermediate or terminal node. As an example, OUTP5:MOD3:FM2 would specify the second FM signal component of the third modulation signal on the fifth output channel.

Devices which implement multiple channels/capabilities shall recognize nodes which do not have numeric suffixes and treat them as channel/capability 1. Instruments with a single channel/capability are not required to accept a numeric suffix. This allows for upward compatibility.

6.2.5.3 Logical Instruments

A complex instrument such as a VXI card cage can be logically modeled as separate instruments, each with its own (secondary) bus address. For example, common functions for all instruments can be located at secondary address 00, and otherwise each instrument responds individually to *IEEE 488.2* commands. This requires that each logical instrument (card) implement its own status model and I/O buffers.

7 Parameters

SCPI uses the parameter forms described in *IEEE 488.2*, section 7.7, with some additional restrictions. Also note that SCPI specifies the values for all commands upon receipt of *RST. In some cases, these reset values are device-dependent, but all measurement parameters must be set to some deterministic value for that particular instrument, which in turn must be documented in the instrument manual.

7.1 Character Program Data

Where possible, the same truncation rules are used for parameters as for headers. However, in many cases industry standards take precedence. For example, IDC is a better choice for DC current than anything SCPI rules would define, simply because it is an existing standard with wide acceptance. In addition, several character program data forms are predefined as extensions of <DECIMAL NUMERIC PROGRAM DATA>.

7.2 Decimal Numeric Program Data

Numeric elements shall be used only for representing numeric quantities. They shall not be used for selecting functions on a “One of N” position switch.

Implementations shall accept numeric data as described in *IEEE 488.2*, section 7.7.2.4. However, any number which exceeds +-9.9 E 37 shall generate an execution error (-222, “Data out of range”). Numbers shall be rounded to the closest “correct” value that the instrument accepts without error. This document does not define the value a parameter is set to if an out-of-range value is received.

7.2.1 <numeric_value> Definition

The decimal numeric element is abbreviated as <numeric_value> throughout this document. This is different from the <NRf> described in section 7.7.2.1 of *IEEE 488.2* in several ways.

Several forms of <CHARACTER PROGRAM DATA> are defined as special forms of numbers. These are: MINimum, MAXimum, DEFault, UP, DOWN, Not A Number (NAN), INFinity, and Negative INFinity (NINF). Individual commands are required to accept MIN and MAX. DEFault, UP, DOWN, NAN, INFinity, and NINFinity may be implemented at the discretion of the designer, in which case it shall be noted in the instrument documentation. Where an optional form is accepted, it will be noted in the command description.

A <non-decimal numeric> (*IEEE 488.2*, section 7.7.4), a <numeric_expression>, or a <label> is part of <numeric_value> if an instrument implements these features.

7.2.1.1 DEFault

The special <numeric_value> parameter DEFault may be provided to allow the instrument to select a value for a parameter. When DEFault is sent, the instrument shall select a value which is deemed to be convenient to the customer. The setting may be device-dependent, or it may be defined as part of this standard. The use of DEFault is optional on a command-by-command basis. Individual commands shall document where DEFault is required.

For example, to use the SYST:TIME command to set a device’s clock ahead one hour (Daylight Savings Time), a program might send SYST:TIME UP,DEF,DEF<nl>.

Another example is found in the MEASure commands. The syntax of the command which measures DC voltage is:

```
MEASure:VOLTage:DC [<expected value>[,<resolution>]]
```

The MEASure command specifies that parameters are defaulted from the right and that any parameter may be defaulted by using DEFault in place of the parameter. The following command would measure DC voltage, defaulting the range to an instrument dependent value (possibly autorange), but specifying the resolution at 0.001 Volt:

```
MEASure:VOLTage:DC DEFault,0.001V
```

7.2.1.2 MINimum|MAXimum

The special form numeric parameters MINimum and MAXimum shall be provided which assume the limit values for the parameter. The maximum and minimum shall be queryable by sending <header>? MAXimum|MINimum. The MAXimum value refers to the largest value that the function can currently be set to, and MINimum refers to the value closest to negative infinity that the function can be currently set to.

Some commands have multiple parameters. The query form of these commands returns a list of values representing the current value of each of the parameters, in the order of their normal occurrence in a program message. If a MINimum/MAXimum query of multiple parameters is allowed, the keywords MINimum and MAXimum must occur as many times in the query as there are parameters. MINimum requests that the instrument return the legal value which is closest to negative infinity for the parameter; MAXimum requests the legal value which is closest to positive infinity.

For example, suppose an instrument implements the SYST:TIME command, which requires three parameters, and allows MIN/MAX queries on this command. The following queries shall have these results:

- SYST:TIME?<nl> shall return the current setting of the time-of-day clock in the instrument.
- SYST:TIME? MAX,MAX,MAX<nl> could return 23,59,59.
- SYST:TIME? MAX<nl> shall set an error (-109, “Missing parameter”), since three parameters are required and only one was sent.

7.2.1.3 UP/DOWN

An instrument may optionally allow the use of steps for some or all of its numeric entry. If steps are used, the keywords UP and DOWN shall be used as numeric parameters which perform stepping. Steps may be adjustable through the step node for each individual parameter.

The instrument may step a parameter when UP or DOWN is received in lieu of a numeric value. This capability is optional. However, if the capability is implemented, the device shall include a node for each command which accepts step parameters. This node will specify the step. The step may either be a fixed linear size or a logarithmic number representing number of decades/step.

7.2.1.3.1 STEP Subsystem Command Syntax

The form of the step subsystem is as follows:

```
:STEP
  [:INCRement]      <numeric_value>
   :PDECade         <numeric_value>
   :MODE             LINear|LOGarithmic|L125|L13
   :AUTO            <Boolean>|ONCE
```

7.2.1.3.2 [:INCRement] <numeric_value>

This command controls the step size in absolute units when STEP:MODE LINear is selected.

At *RST, this value is device-dependent.

7.2.1.3.3 :PDECade <numeric_value>

This command controls the number of steps per decade when STEP:MODE LOGarithmic is selected.

At *RST, this value is device-dependent.

7.2.1.3.4 :MODE LINear|LOGarithmic|L125|L13

This command controls the linearity of steps. The various parameters have the following meanings:

- LINear: Steps are a fixed value added to or subtracted from the current value of the parameter.
- LOGarithmic: Steps are placed at points spaced logarithmically. If the current value of the function is x, the value of the function after executing an up shall be determined by the following formula:

$$10^{\left\lfloor \frac{\lfloor \log_{10}(x) * STEP{:}PDECade \rfloor + 1}{STEP{:}PDECade} \right\rfloor}$$

  and the value of the function after executing a DOWN shall be determined by:

$$10^{\left\lfloor \frac{\lceil \log_{10}(x) * STEP{:}PDECade \rceil - 1}{STEP{:}PDECade} \right\rfloor}$$

  where $\lfloor$ is the symbol for *floor*, and $\lceil$ is the symbol for *ceiling*.

- L125 — Steps are determined by the sequence
  ... 0.1,0.2,0.5,1,2,5,10,20,50,100 ...

Executing UP will set the value of the function to the next higher value in the sequence. Executing DOWN will set the value of the function to the next lower value in the sequence.

- L13 — Steps are determined by the sequence
  ... 0.1,0.3,1,3,10,30,100 ...

Executing UP will set the value of the function to the next higher value in the sequence. Executing DOWN will set the value of the function to the next lower value in the sequence.

At *RST, this value is device-dependent.

7.2.1.3.5 :AUTO <Boolean>|ONCE

The step mode and increment is coupled to other instrument settings and physical inputs when AUTO is ON.

7.2.1.3.6 STEP Subsystem Examples

```
FREQ:CENT:STEP 5 MHZ<nl>
FREQ:CENT UP<nl>
```


sets the center frequency step to 5 MHz, and then increments the current value by 5 MHz.

```
BAND:RES 1MHZ<nl>
BAND:RES:STEP:MODE LOG;PDEC 3<nl>
BAND:RES UP<nl>
```

The above sequence would specify a logarithmic step, set three steps/decade, and then increment the resolution bandwidth by 1/3 decade (logarithmic) to a final value of 2.154 MHz.

```
BAND:RES 1MHZ<nl>
BAND:RES:STEP:MODE L125<nl>
BAND:RES UP<nl>
```

The above sequence would specify a 125 logarithmic sequence, and then increment the resolution bandwidth to the next step in the sequence to a final value of 2.0 MHz.

7.2.1.4 INFinity and Negative INFinity (NINF)

A special case is made for infinity. Positive infinity is represented as 9.9 E 37. Negative infinity is -9.9 E 37. Instruments shall accept numbers within this range as valid numeric data. They shall also limit response data to this range. If a valid instrument setting is infinity or negative infinity, the values 9.9 E 37 and -9.9 E 37 shall be accepted and returned to represent positive and negative infinity respectively, and on input, the implementation shall accept INFinity as an alias for positive infinity, and NINFinity as an alias for negative infinity.

The numeric values for positive and negative infinity were chosen so that they fit into a 32-bit *IEEE* 754 floating point number. This allows easy implementation using standard tools in all popular languages and operating systems. Note that this does not limit the numeric resolution. These values are larger than any quantities used or anticipated in instruments.

7.2.1.5 Not A Number (NAN)

Not a number is represented as 9.91 E 37. Not a number is defined in *IEEE 754*. Typical applications are dividing zero by zero or subtracting infinity from infinity. Not a number is

also used to represent missing data. A typical application is when a portion of a trace has not been acquired yet. On input, devices shall accept NAN as an alias for not a number.

The numeric value for NAN was chosen so that it can be represented as a 32-bit *IEEE* 754 floating point number. This allows easy implementation using standard tools in all popular languages and operating systems. Note that this does not limit the numeric resolution. This value is larger than any quantities used or anticipated in instruments.

7.2.2 Unit Suffixes

The <DECIMAL NUMERIC PROGRAM DATA> may be followed by an optional suffix. All parameters which have associated units shall accept a suffix. Only suffixes appropriate for the command should be accepted. All suffixes must include the associated unit. See *IEEE 488.2*, 7.7.3, <SUFFIX PROGRAM DATA>, for a complete discussion of this topic.

Suffixes must accept multipliers except in cases where the multiplier is illogical, such as dBm or PCT. Table 7-2 in *IEEE 488.2* lists the allowed multipliers. If any of the multipliers are allowed with a suffix, then all the multipliers shall be interpreted properly. Compound suffixes are allowed.

If a suffix is included, the suffix and associated multiplier, if implemented, are applied to the parameter. If the suffix is omitted, default units are used. The default unit is determined either from the :UNIT subsystem as described in 7.5 of Syntax & Style and in Chapter 23 of the Command Reference, or it is the fundamental unit associated with the parameter. Fundamental units are either described in the appropriate subsystem or in the one shown in *IEEE 488.2*, table 7-1.

7.3 Boolean Program Data

The form <Boolean> is used throughout this document as a shorthand for the form ON|OFF|<NRf>. Boolean parameters have a value of 0 or 1 and are unitless.

On input, an <NRf> is rounded to an integer. A nonzero result is interpreted as 1. This algorithm is the same as the one described in *IEEE 488.2*, section 10.25.3.

The <CHARACTER PROGRAM DATA> elements ON and OFF shall be accepted on input for increased readability. ON corresponds to 1 and OFF corresponds to 0.

Queries shall return 1 or 0, never ON or OFF.

7.4 Coupling of Functions

There are two forms of coupling: functional and value. A coupling occurs when sending a command changes the value associated with another command. In general, functional couplings are discouraged except in the MEASure subsystem. Value couplings are allowed if the coupling equations are well specified.

Some functions may have the ability to be decoupled. If functions can be decoupled, the ability to recouple them shall also exist. If a function cannot be decoupled, then a node to set its value shall not exist, or shall be implemented as query-only. If a node is settable under some conditions, then the node may exist.

Another level in the tree is used to control coupling. The keyword AUTO shall be used to control coupling. For example, the command to couple resolution bandwidth would be:

```
BAND:RES:AUTO ON
```

Setting a value explicitly for a function shall cause the function to be decoupled (:AUTO OFF), if decoupling is possible. Otherwise an error shall be generated.

Selecting :AUTO ONCE shall cause the function to select the most appropriate value for current conditions (range, signal level, etc), and then be decoupled. A subsequent query of an :AUTO? will always return 0, since ONCE is an event which leaves the function in AUTO OFF mode.

7.4.1 Functional Coupling

Functional coupling occurs when a command causes side effects in the basic operating mode of the device. An example is a command which sets the start frequency and also starts a sweep.

If a command which invokes a functional coupling is necessary, the instrument shall also implement primitive commands which do not have functional couplings. The instrument may then implement a complex command which is described as a combination of these primitive commands. This complex command may contain functional couplings, and must document the sequence of primitive commands used in its programming documentation.

For example, if a source needs a command which sets the AM modulation level, and which has the side effect of turning the modulation ON, the following could be implemented:

Primitives:
AM
:STATE <Boolean>
[:DEPTh] <numeric_value>

Complex command added:
AM
:SDEPth <numeric_value>

Where AM:SDEPth is defined in the manual as:
AM
:DEPth <numeric_value>;
:STATe ON

7.4.2 Value Coupling

The other form of coupling is called **value coupling**, which is allowed in SCPI. Values are coupled when a command changes the value of other numeric settings in the instrument. The coupling relation would follow device-dependent equations. For example, bandwidth in a receiver may be coupled to the frequency span. The individual command descriptions define these coupling equations.

Groups of functions may be coupled in complex ways which affect the values for each other. For example START, STOP, CENTER, and SPAN are all value-coupled. CENTER is the arithmetic average of START and STOP. SPAN is the difference between START and STOP. Changing any one affects the value of two others.

7.4.3 Automatic Coupling

A special type of coupling is the **automatic coupling**. This is when the device provides an algorithm to select the value of a parameter. This algorithm could be based upon physical inputs or other settings. When it is desirable to turn this algorithm on and off, the keyword AUTO is added. The AUTO command may accept parameters of ON|OFF|ONCE. ON enables the instrument algorithm. OFF disables the instrument algorithm. ONCE causes the algorithm to be employed once, changing the associated parameter, and then reverting to AUTO OFF.

7.5 Units of Measure and Suffixes

Most of the information on units and suffixes is specified in *IEEE 488.2*, section 7.7.3. However, further clarification is needed for units of power and amplitude, and for unitless quantities.

7.5.1 Units of Amplitude and Power

- The fundamental unit of linear power is the Watt (W).
- The fundamental unit of linear amplitude is the Volt (V).
- The fundamental unit of logarithmic power is the dBm.
- The fundamental unit of logarithmic amplitude is the dBV.

Industry practices have caused other units to become widely used in various disciplines, particularly the units of uV, dBuV, dBuW and dBmV. Furthermore, many instruments cover a broad spectrum of applications where different units are considered the standard. Therefore, instruments which implement commands whose parameters might be expressed in several different amplitude or power units shall provide the following means of setting and measuring amplitude and power functions:

If the unit is not specified, the default unit shall be that specified with a command in the UNIT subsystem, or a UNIT command in the appropriate sub-tree. The *RST values of settings in the UNIT subsystem shall be device-dependent. However, if a *RST unit is different than the specified default unit, or is a logarithmic unit, then the command in the UNIT subsystem for that unit must be implemented.

If a suffix is sent with an <NRf>, the unit corresponding to that suffix shall override the default unit for that parameter.

If an instrument accepts units from one of the classifications described above, it must accept all suffixes of that classification. For example, if an instrument accepts Watts, it must also accept uW, mW, etc. This includes all suffix multipliers described for that unit in *IEEE 488.2*. However, it may, but is not required, to accept Volts as a unit. In instrument which accepts both requires a defined impedance for the conversion

$$(Power = \frac{Voltage^2}{Impedance})$$

either explicitly through an input or output impedance command or implicitly if the impedance of the instrument is known and fixed. A further restriction, is that if an instrument accepts logarithmic units, it shall also accept the comparable linear unit. For example, an instrument which accepts dBms must also accept Watts, MW, UW, etc. For example:

```
POW:UNIT DBUV<nl>
POW:LEV 50<nl>
```

shall set the amplitude level to 50 dBuV.


```
POW:UNIT DBM<nl>
POW:LEV 5V<nl>
```

would set the power level to 5 Volts.

Since some instruments are used in applications where both power and voltage units are appropriate, the system's impedance may be important to the user. This impedance is needed to do the conversion between power and voltage (and maybe even current). All instruments are encouraged to provide a node for at least querying this impedance. The node shall be called :IMPedance, and will generally appear under the INPut or OUTPut subsystem. Open circuits shall be expressed by setting IMPedance INFinity. It is therefore possible to express output power in Volts:

```
POW:UNIT V<nl>
OUTP:IMP 50 OHM<nl>POW:LEV 5<nl>
```

would set the power level to 5 Volts into a 50 Ohm load, or 0.5 Watt.

It is also sometimes necessary to determine which amplitude measurement is being made. Therefore, a measurement qualification may be appended to the suffix. The suffix appendices are described as:

- PK: Peak amplitude
- PP: Peak-to-peak amplitude
- RMS: RMS amplitude

If no suffix appendix is specified, the default is device-dependent. However, all instruments shall accept the suffix appendix for all types they support.

For example, a voltmeter which measures RMS voltage shall accept the suffixes V and VRMS. A function generator which outputs peak voltage would accept the suffixes V and VPK. An RF signal source might accept dBUV, dBUVRMS and dBUVPK.

7.5.2 Expressing Unitless Quantities

The ratio of two linear quantities shall default to a unitless ratio (A/B). For example, if the input power for a system is 5 Watts, and the output power is 20 Watts, the power gain = 20W/5W = 4.

The difference between two logarithmic quantities shall be expressed in dB. For example if the input power of a system is 3 dBm and the output power is 5 dBm then the gain of the system is 5 dBm-3 dBm=2 dB.

Under exceptional conditions other units may be used for the ratio of two quantities. A valid exception would be if a standards-setting organization (Bell, CCITT, military, IEEE, IEC, etc.) specifies that a measurement shall be made using a special unit like PCT (percent).

8 Expressions

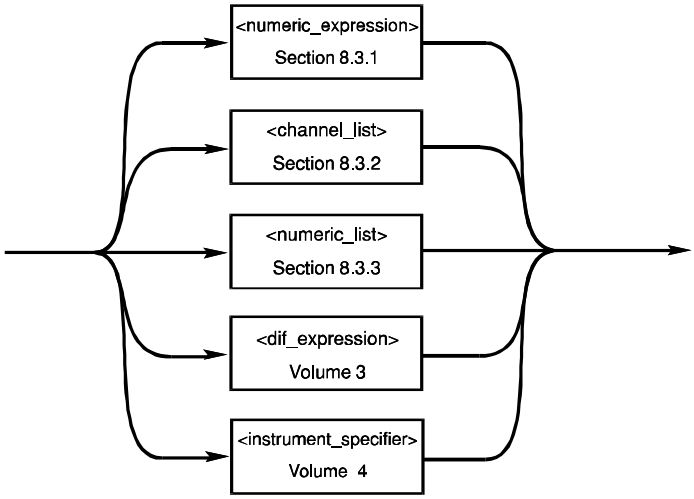
8.1 Function

The <EXPRESSION PROGRAM DATA> described in *IEEE 488.2* is more tightly defined in this section. SCPI defines five types of expressions, numeric expressions, channel lists, numeric lists, Data Interchange Format (dif) expressions, and instrument specifier expressions. The dif expressions are defined in Volume 3, "Data Interchange Format." Instrument specifiers are defined in Volume 4, "Instrument Class Applications."

Other expression types may be added in future releases of this document, including but not limited to logical expressions, trigger expressions, and sequence expressions. Other uses of *IEEE 488.2* expressions are legal, and may be documented in the individual command descriptions.

8.2 Usage

The use of expressions is optional in SCPI. For expression types defined in this section, it is permissible to use any subset. Only expression types which are required by a particular command may be recognized as parameters of that command.



8.3 Syntax

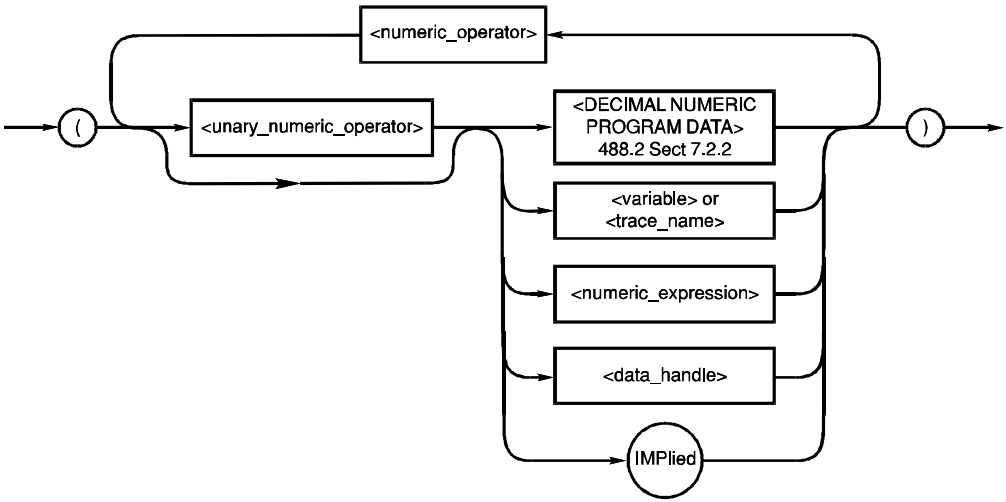
Inside expressions, white space as defined in *IEEE 488.2*, section 7.4.1.2 is allowed between any lexical elements. For expressions defined in this section, a **lexical element** is defined as any operator, punctuation, or *IEEE 488.2* syntactic element. For a <dif_expression>, spaces may appear around block names, block modifiers, keywords, or value types. See *Data Interchange Format,* Section 3.2. Note that explicit white space may be required to separate lexical elements. No semantic meaning should be attached to the case of alpha characters

used in operators. For example, the following are instances of the same operator:

AND
and
And

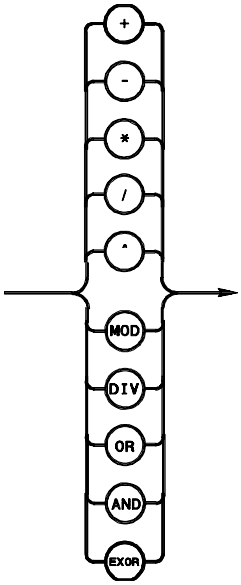
8.3.1 Numeric Expression

A numeric expression is a collection of terms which evaluates to a trace, number, array, or other data element.

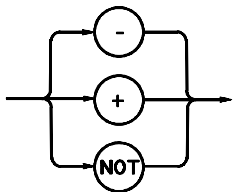


8.3.1.1 Syntax

Where <numeric_operator> is defined as:

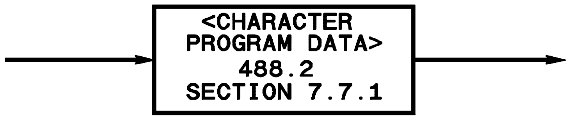


Where <unary_numeric_operator> is defined as:



Unary operators should not be used with signed numbers.

Where <variable> or <trace_name> is defined as the diagram shows:



8.3.1.2 Precedence Rules

Expressions shall be evaluated according to the following precedence rules:

1. Enclosed by parentheses
2. Unary operators (+ and -)
3. ^(exponentiation)
4. * (multiplication), / (division), MOD and DIV
5. + (addition) and - (subtraction)
6. NOT
7. AND
8. OR and EXOR
9. Left to right

8.3.1.3 Semantics

As stated in the syntax diagram, expressions may contain terms which are numbers, traces, variables, or expressions.

Elements in a numeric expression are promoted to the size and type of the most complex element, and the result of the expression is of that type. That is, an expression containing a <trace_name> will be evaluated according to the rules for arithmetic on traces, and the result will be a trace. If an expression contains both a <trace_name> and scalar data, a trace is created with all elements set to the value of the scalar data before arithmetic is performed. For example, if TREF is a trace_name, the expression (TREF-3) results in a trace the same size as TREF, with each data element lower by three.

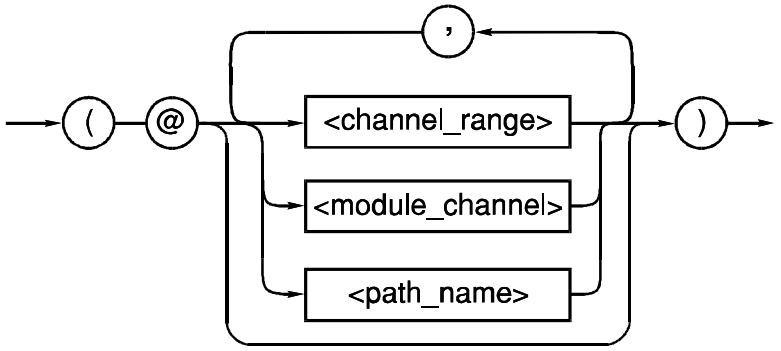
8.3.2 Channel Lists

Channel lists are used to specify electrical ports on an instrument. They are typically used for signal routing either in a standalone switch box or in an instrument with multiple input channels. An instrument with multiple channels may or may not do any signal switching as

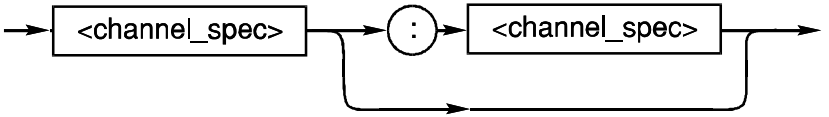
the result of a channel list. Completely separate sensing channels are allowed, but may appear to the language the same as channels which are switched.

A channel list may appear in measurement, configuration, and other such commands. For example, MEAS:VOLT? (@1,3,4:6) says measure the voltage on channels 1, 3, and 4 through 6. Whether the measurements are performed simultaneously or in the order in the list is unspecified. Channel lists are also used by the ROUTe subsystem, "Command Reference," 15.1.

The syntax for a <channel_list> is:

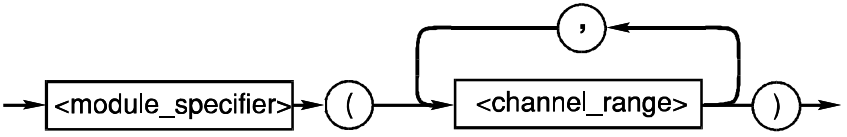


Where <channel_range> is defined as:

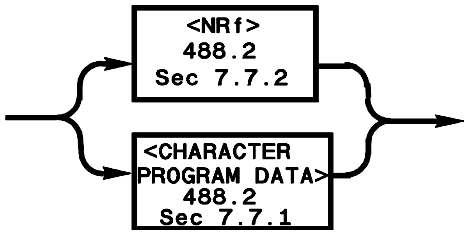


Where both instances of <channel_spec> must contain the same number of dimensions.

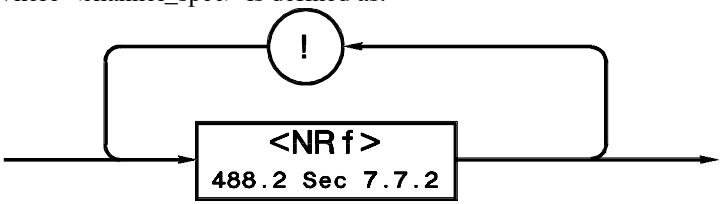
Where <module_channel> is defined as:



Where <channel_range> indicates all the channels from the first number through the second number and where <module_specifier> is defined as:



Where <channel_spec> is defined as:



The number of dimensions in a <channel_spec> is one greater than the number of occurrances of the '!' character in the <channel_spec>

When a <channel_range> of dimension greater than one is scanned, the right-most index of the <channel_spec> varies most rapidly.

The multidimensional channel range is to be considered as a list of single dimension channel ranges. For example, (@1!1:2!3) means dimension 1 (row) ranges from 1 to 2 and dimension 2 (column) ranges from 1 to 3. If the size of the matrix in this example is 2X8, then the range (@1!1:2!3) does *not* mean: row 1, column 1 through 8 and subsequently row 2, column 1 through 3, but row 1, column 1 through 3 and next row 2 column 1 through 3. This method provides a functional compatibility between commands sent to matrix switches with different row lengths.

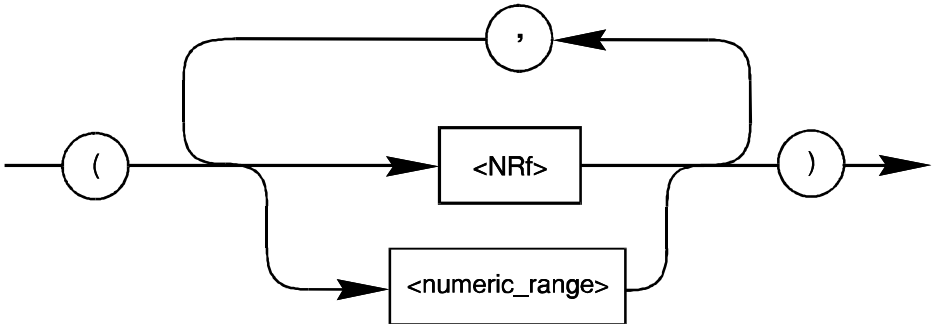
Channel lists are order sensitive. For example:

SCAN (@5:3) means close 5, 4, 3 in order.
SCAN (@1!1:2!3) means close 1!1, 1!2, 1!3, 2!1, 2!2, 2!3 in order.
SCAN (@1!3:2!1) means close 1!3, 1!2, 1!1, 2!3, 2!2, 2!1 in order.

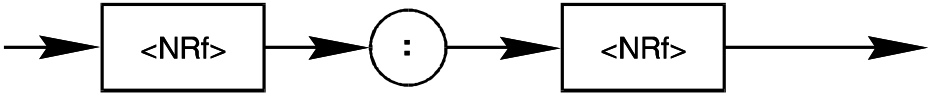
8.3.3 Numeric Lists

A numeric list is a an expression format for compactly expressing numbers and ranges of numbers in a single parameter.

The syntax for <numeric_list> is:



Where <numeric_range> is defined as:



The range is inclusive of the specified numbers.

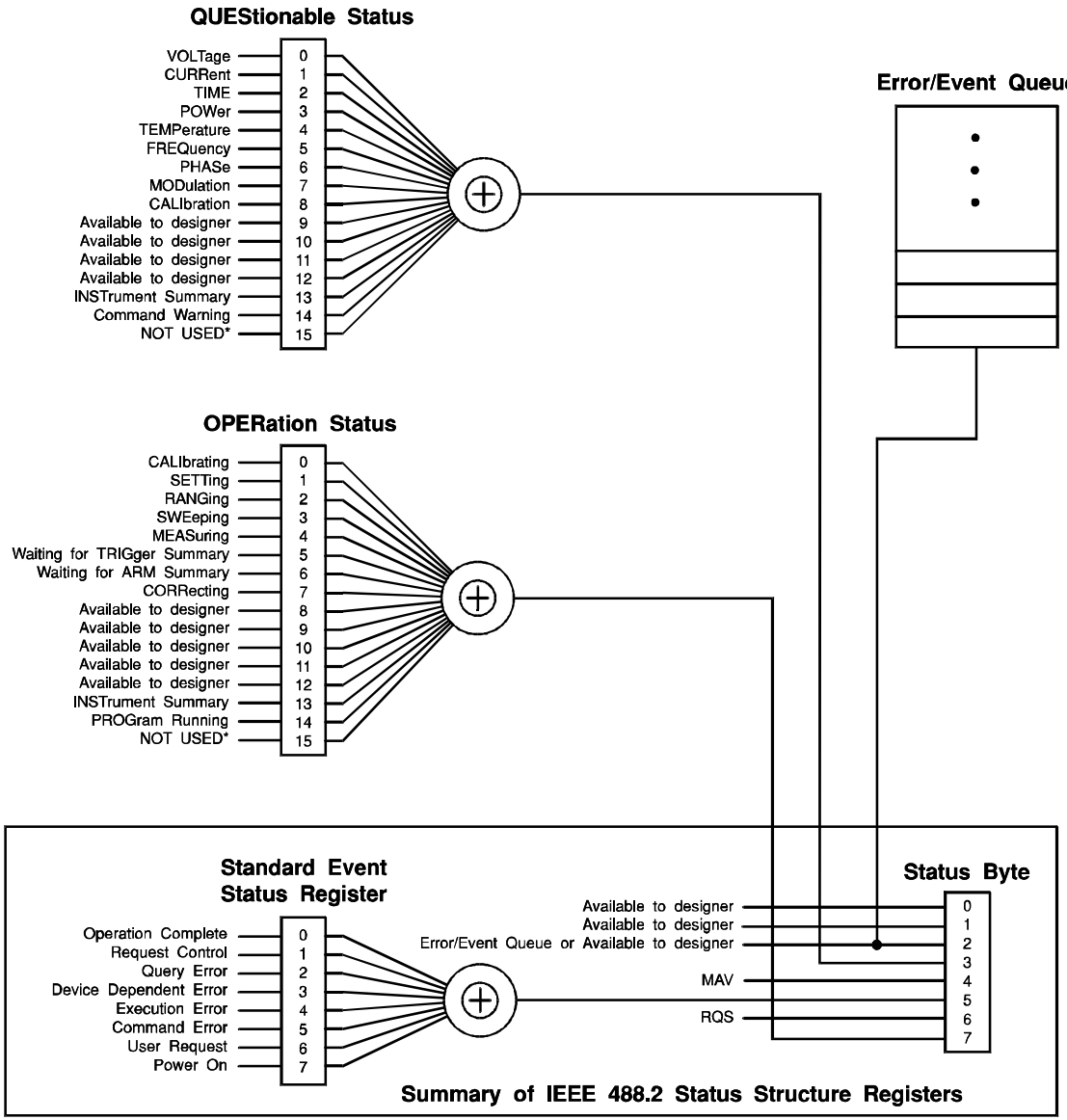
9 Status Reporting

SCPI requires the status mechanism described in chapter 11 of *IEEE 488.2*, including full implementation of the Event Status register structure.

A SCPI device shall include the SCPI-defined OPERation status register and QUEStionable data/signal status register with the associated condition, event, and enable commands. In Figure 9-1 on the following page, "Minimum Status Reporting Structure Required by SCPI", a pictorial representation is given that shows the core of the SCPI-required status reporting capability. Additional requirements are established for instruments that support either multiple logical INSTruments or the expanded capability TRIGger model.

In general, a status register should fit into a 16-bit integer with the most-significant bit always zero (positive logic).

In the figures in this chapter, an elongated box is used to represent the "Status Data Structure-Register Model," which is defined in *IEEE 488.2*. SCPI consists of condition, event, enable and optional transition registers. Two concentric circles with a cross through the center one indicates a logical summing.



* The use of Bit 15 is not allowed since some controllers may have difficulty reading a 16 bit unsigned integer. The value of this bit shall always be 0.

Figure 9-1 Minimum Status Reporting Structure Required by SCPI

9.1 The Device-Dependent Register Model

The Device-Dependent Register model follows the structure described in *IEEE 488.2*, section 11.4.2. The transition filter described in figure 11-6 is actually a pair of programmable transition filters which are described below.

The commands which access these registers are described in the "SCPI Language Description."

If the Error/Event Queue Summary is reported in the Status Model, then bit 2 of the Status Byte shall be used to reflect the Empty/Non-Empty status of the queue. A bit value of 1 indicates the queue is not empty. See the SYSTem:ERRor subsystem, Command Reference, 21.8.

9.2 Transition Filters

Transition filters are described in *IEEE 488.2*, section 11.4.2.2.1. SCPI allows the use of programmable transition filters. When transition filters are used, SCPI requires the use of separate positive and negative transition filters. A positive transition filter allows an event to be reported when a condition changes from false to true. A negative filter allows an event to be reported when a condition changes from true to false. Setting both positive and negative filters true allows an event to be reported anytime the condition changes. Clearing both filters disables event reporting.

The contents of transition filters are unchanged by *CLS and *RST.

9.3 Operation Status Register

The OPERation status register contains conditions which are part of the instrument's normal operation.

The definition of each of these bits (condition register) is as follows:

- **0-CALibrating** — The instrument is currently performing a calibration.
- **1-SETTling** — The instrument is waiting for signals it controls to stabilize enough to begin measurements.
- **2-RANGing** — The instrument is currently changing its range.
- **3-SWEeping** — A sweep is in progress.
- **4-MEASuring** — The instrument is actively measuring.
- **5-Waiting for TRIG** — The instrument is in a "wait for trigger" state of the trigger model.
- **6-Waiting for ARM** — The instrument is in a "wait for arm" state of the trigger model.
- **7-CORRecting** — The instrument is currently performing a correction.

- **8-12 available to designer**
- **13-INSTrument Summary Bit** One of n multiple logical instruments is reporting OPERational status.
- **14-PROGram running** — A user-defined programming is currently in the run state.
- **15 always zero**

9.4 QUEStionable Data/Signal Status Register

The QUEStionable status register set contains bits which give an indication of the quality of various aspects of the signal.

A bit set in the condition register indicates that the data currently being acquired or generated is of questionable quality due to some condition affecting the parameter associated with that bit. For example, if the FREQ bit were set, this would mean that the frequency accuracy of the signal was of questionable quality.

The frequency bit might, in turn, have a register set associated with it, further refining the error into device-dependent conditions such as loop unlocked, oven cold, or reference signal missing. This layering of registers is called “fan-out”. See Figure 9-2 for an illustration of this technique. The device designer should be aware that adding registers to the status model increases complexity. At the same time, it may be the only way to communicate time sensitive status information to the instrument controller.

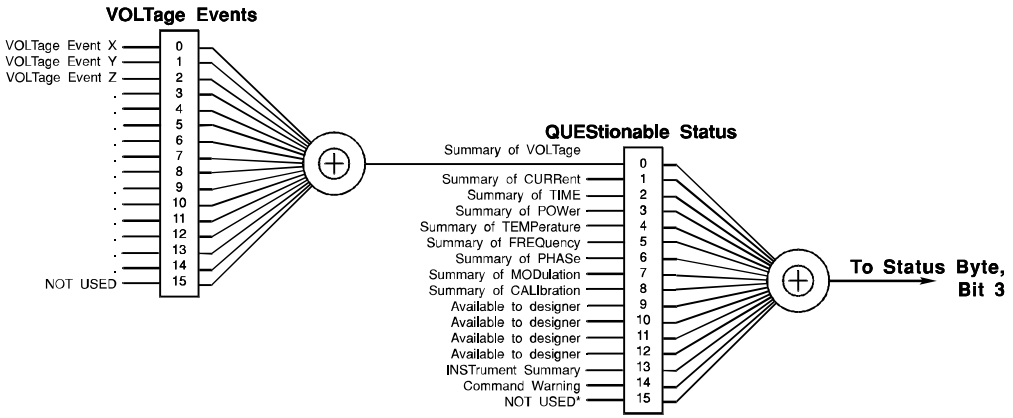


Figure 9-2 Hierarchical Expansion of the QUEStionable Status Register

Bit 14 is defined as the Command Warning bit. This bit indicates a non-fatal warning that relates to the instrument’s interpretation of a command, query, or one or more parameters of a specific command or query. Setting this bit is a warning to the application that the resultant instrument state or action is probably what was expected but may deviate in some manner.

For example, the Command Warning bit is set whenever a parameter in one of the Measurement Instruction commands or queries is ignored during execution. Such a parameter may be ignored because it cannot be specified by a particular instrument.

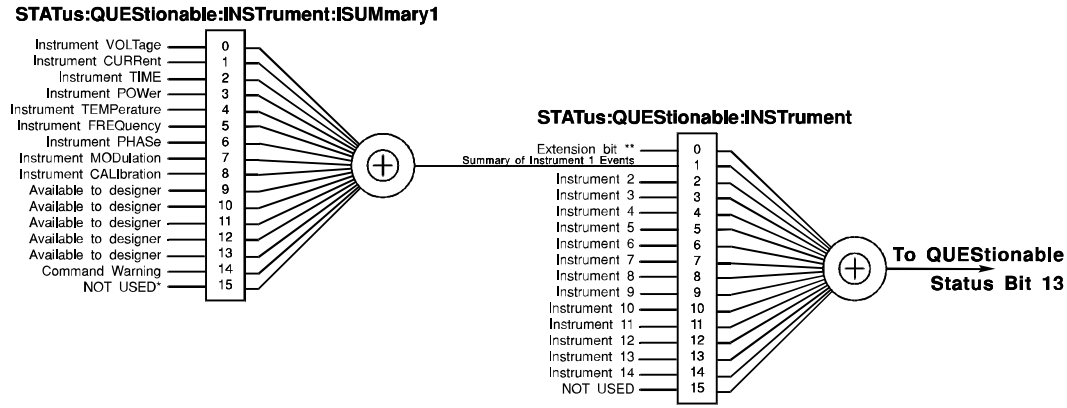
Bit 13, INSTrument summary, is described later in this chapter in association with multiple logical instruments.

Bits in both OPERation and QUEStionable status registers may be redefined by the application programmer. This optional feature allows an application program to configure two registers in whatever form it likes. Any error or event number from the entire pool of errors/events the instrument can generate may be mapped into any bit of the OPERation or QUEStionable register.

For example, an application program controlling a voltmeter may choose to map VOLTage event X, VOLTage event Y and VOLTage event Z directly into the QUEStionable register so that it has more direct access to them. Another purpose for this feature is to minimize the need for adding complex hierarchical registers to the status model. The mapping is controlled by the STATus:OPERation:MAP and STATus:QUEStionable:MAP commands.

9.5 Multiple Logical Instruments

A SCPI device that supports multiple logical instruments may include an INSTrument summary status register and an individual instrument ISUMmary for each logical instrument.



* The use of Bit 15 is not allowed since some controllers may have difficulty reading a 16 bit unsigned integer. The value of this bit shall always be 0.

** The extension bit is used, where required, to summarize the events in Instrument15 and up. The extension of the status structure shall conform to IEEE 488.2

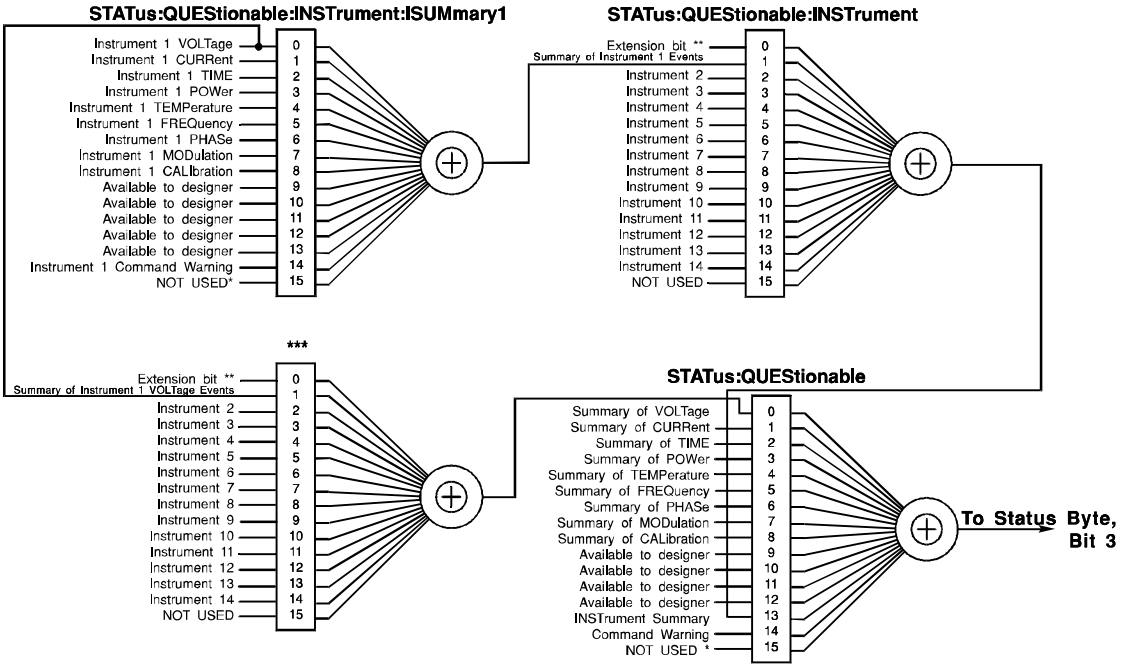
Note that the OPERation Register is expanded in the same manner for multiple logical instruments.

Figure 9-3 Expansion of the INSTrument Summary Bit for Multiple Logical Instruments

The ISUMmary registers shall report to the INSTrument register, which in turn shall report to bit 13 of the QUEStionable or OPERation status register. This is shown pictorially in Figure 9-3, “Expansion of the INSTrument Summary Bit for Multiple Logical Instruments.” Such a multilogical instrument may also expand the QUEStionable and OPERation register in the manner shown in Figure 9-4, “Expansion of QUEStionable Register for Multiple Logical Instruments.”

Using such a status register configuration allows a status event to be cross-referenced by instrument and type of event. Further, when using a single logical instrument, the status structure is seen to behave in a manner that is directly compatible with a single physical instrument of the same capability. This affords upward compatibility.

For multiple logical instruments, the INSTrument register indicates which instrument(s) have generated an event. The ISUMmary register is a pseudo-questionable status register for



* The use of Bit 15 is not allowed since some controllers may have difficulty reading a 16 bit unsigned integer. The value of this bit shall always be 0.

** The extension bit is used, where required, to summarize the events in Instrument15 and up. The extension of the status structure shall conform to IEEE 488.2

*** The name of the register will be specified in a later version of SCPI

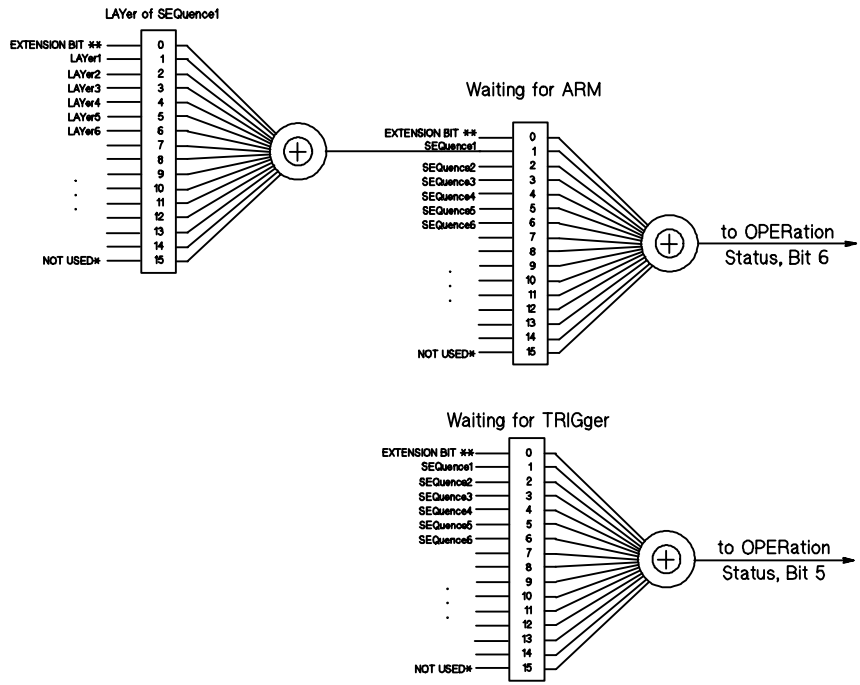
Note that the OPERation Register is expanded in the same manner for multiple logical instruments.

Figure 9-4 Expansion of QUEStionable Register for Multiple Logical Instruments

a particular logical instrument. There may be two INSTrument registers for each logical instrument in the device. For each event type, such as VOLTage, all the events may be ORed together from each logical instrument to provide a summary by event type to the QUEStionable status register.

9.6 Status Structure for the Expanded Capability Trigger Model

For instruments that implement the expanded capability trigger model, those instruments may implement status structures to indicate the exact point in which the wait for ARM or TRIGger is occurring. If multiple SEQuences are employed, then the status structure shall exist to indicate in which SEQuences the wait for ARM or TRIGger is occurring. If multiple ARM LAYers are employed, then the status structure shall exist to indicate in which LAYer the wait for ARM or TRIGger is occurring. Figure 9-5, "Expansion of the OPERational Register for the Expanded Capability Trigger Model," shows the structure for an instrument that employs more than one TRIGger sequence, more than one ARM sequence, and multiple ARM LAYers.



* The use of Bit 15 is not allowed since some controllers may have difficulty reading a 16 bit unsigned integer. The value of this bit shall always be 0.

** The extension bit is used, where required, to summarize the events in INSTrument15 and up. The extension of the status structure shall conform to IEEE 488.2

Figure 9-5 Expansion of OPERation Register for the Expanded Capability Trigger Model

10 *RST Conditions

All instruments shall return to a known configuration when the *IEEE 488.2* *RST command is received. The “Command Reference” specifies the *RST condition for every command. In some cases, the condition may be instrument-dependent. **Instrument-dependent** means that the designer may determine the setting at *RST. However, the setting shall be known for the instrument. Leaving a setting undefined at *RST is not allowed, unless specifically permitted in the command description.

Consideration shall be given to safety if the instrument is capable of producing hazardous conditions.

Operation after a *RST is optimized for remote operation.

A *RST command returns the instrument to a state where it is waiting for a command to initiate measurements or other instrument actions.

All instruments shall place themselves in the trigger idle state. Source instruments should not be sourcing power at any output port. This condition may be reached by lowering the power/voltage level or by turning the output state to OFF. Input ranges should normally be set to AUTORANGE or minimum sensitivity. Settings which are normally coupled should be coupled. Special modes, complex modulations, postprocessing, or similar functions which are generally application-dependent should be disabled leaving the instrument in its most fundamental mode of operation.

Finally, thought should be given to implementing *RST conditions so that the incremental programming necessary after *RST for the applications be minimized.

The SYSTem:PRESet command performs the same action as the front panel preset key. This typically sets all instrument parameters to values for good local/human interaction. In manual operation, it is often desirable to have SYSTem:PRESet enable continuous measurements. This may be different from the *RST state and the power-on state.

Status structures are not affected by *RST. The status event bits are cleared by *CLS and by reading the event register. The error/event queue is also cleared by *CLS. Device-dependent and SCPI status registers and queues are preset with the SCPI required STATus:PRESet command.

If a device is unable to implement the *RST condition of a function, it may choose a different *RST condition. However, under these circumstances, the command for this function must be implemented, even if only one selection is available. If the device implements the stated *RST setting, the function must assume that value at *RST.

11 Naming Conventions

In many SCPI subsystems there is the need for assigning names to structures or expressions or arbitrary data memory.

This includes naming of windows in the DISPlay subsystem, traces in TRACe, expressions to specify events and sequences in TRIGger, expressions to specify cards and channel lists in ROUTe, and various items elsewhere. If possible, these naming operations should all have the same form so that these operations in the various subsystems can be consistent. A user can therefore transfer knowledge gained in one area to a similar task in another subsystem.

For this proposed general-purpose naming facility, there is no "enable" command; all currently defined names will be associated with their data at define/assign time and exist until deleted. There are two forms of the "purge" command: one which operates on individual names as well as one for the whole name space. Names have implicit types and memory is allocated based on the subsystem in which the name is created. Names must be unique within the system to allow a designer to implement these commands using one global name space.

The same keywords for naming will be used in all subsystems which allow names. Even though TRACe:DEFine and DISPlay:WINDow:DEFine require different parameters, the operations are similar and the instrument programmer can expect similar keywords.

The name space is not affected by *RST, and names remain associated with their data after this operation.

KEYWORD	PARAMETER FORM	NOTES
Naming Sub-subsystem:		
:DEFine	<name>,<data>	
:DEFine?	<name>	
:DELete		
[:NAME]	<name>	
:ALL		[event; no query]
:CATalog?		[query only]

Command Descriptions

Differences among these commands and differences from the macro definition facility are noted in the descriptions.

The name is sent as <character data> rather than as a string, because that is the simplest *IEEE 488.2* type that meets the requirements. Macro definitions needed the <string> type because macro names could contain colons and queries, and their total length could be more than 12 characters. Names do not have those extensions. When a name is returned (for example, in a "CATalog?" query), it must be sent as <string response data> so that a null string ("") can indicate that nothing is defined.

Names defined using these commands are not purged at *RST. The only way of deleting them is through a DELete or DELete:ALL command.

11.1 :DEFine <name>,<data>

This command associates a user-specified <name> with data <data>. The type of <data> is dependent on the subsystem in which the command occurs, and must be specified in the syntax for that tree. The <data> field may be optional for some subsystems.

11.2 :DEFine? <name>

This query requests the instrument to return the definition of <name>. The type of data returned depends on the specific subsystem in which the command occurs. If possible (that is, if *IEEE 488.2* allows), it should be returned in the same form it was sent.

11.3 :DELete[:NAME] <name>

This command undefines the name, disassociates it from any data, and frees the name and its data memory for use by other definitions. There is no query associated with this action.

11.4 :DELete:ALL

This command undefines all names in this subsystem, and frees any associated data memory. There is no query associated with this action.

11.5 :CATalog?

This query requests a list of defined names in this subsystem. The instrument must return a list of one or more strings, each containing one name and separated by commas. If no names are defined, a single null string is returned.

A Programming Tips

By following a set of simple guidelines, a programmer can reduce the number of errors encountered when using SCPI instruments. All references in this appendix are to Volume I: Syntax and Style, unless otherwise specified.

1. Avoid sending default nodes. See 5.1 and 6.2.2 item 5. Default nodes are used within SCPI to allow the language to grow. Thus, older instruments will not have implemented the default node. A command is more likely to work with more instruments if the default nodes are omitted. For example, use:

```
INP:FILT ON
```

rather than:

```
INP:FILT:LPAS:STAT ON
```

2. Avoid sending a numeric suffix of 1 in applications that only use the default capability. See 6.2.5.2. An instrument with multiple capabilities is required to interpret a header without the numeric suffix as if a numeric suffix of one had been used. Leaving off the numeric suffix means the same commands will work with an instrument that has multiple capabilities and an instrument that does not. For example, use:

```
OUTP ON
```

rather than

```
OUTP1 ON
```

3. Be careful when sending coupled commands. See 7.4.2. Many instruments adhere to the suggestion in IEEE 488.2 section 6.4.5.3. If the coupled commands are contiguous in the same program message, the instrument is more likely to resolve any conflict between the current settings and the new settings. For example, use:

```
:FREQ:STAR 100;SPAN 100
```

which sets START to 100, STOP to 200, CENTER to 150, and SPAN to 100. Assume that START was 200 and STOP was 500 before sending;

```
:FREQ:STAR 100
```

Now, START is 100, STOP is 500, CENTER is 300, and SPAN is 400. Then send:

```
:FREQ:SPAN 100
```

which sets START to 250, STOP to 350, SPAN to 100, and leaves CENTER at 300 . These states are very different and the second is probably not the intended one.

4. Verify instrument settings when knowing their exact value is required. If the actual value of a setting is important, query the setting after programming it. An instrument is required to round parameters. See 7.2. For example, sending:

```
:VOLT 1.68
:VOLT?
```

may return 1.68, 1.7, or 2 depending on the capability of the instrument.

5. Send commands and queries in different program messages.

The response from a query combined in a program message with commands that affect the queried value is not predictable. Sending:

```
:FREQ:STAR 100;SPAN 100
:FREQ:STAR?
```

always returns 100. When:

```
:FREQ:STAR 100;STAR?;SPAN 100
```

is sent, however, the result is not specified by SCPI. The result could be the value of START before the command was sent since the instrument might defer executing the individual commands until a program message terminator is received. The result could also be 100 if the instrument executes commands as they are received.

6. Use the highest level commands possible for compatibility across instruments. Whenever feasible, use the MEASure, CONFigure, READ?, FETCh?, and INITiate commands. Use the lower level commands only when a special characteristic of an instrument must be manipulated.

Index

F

H

I

K

L

M

N

O

P

Q

R

S

T

U

V