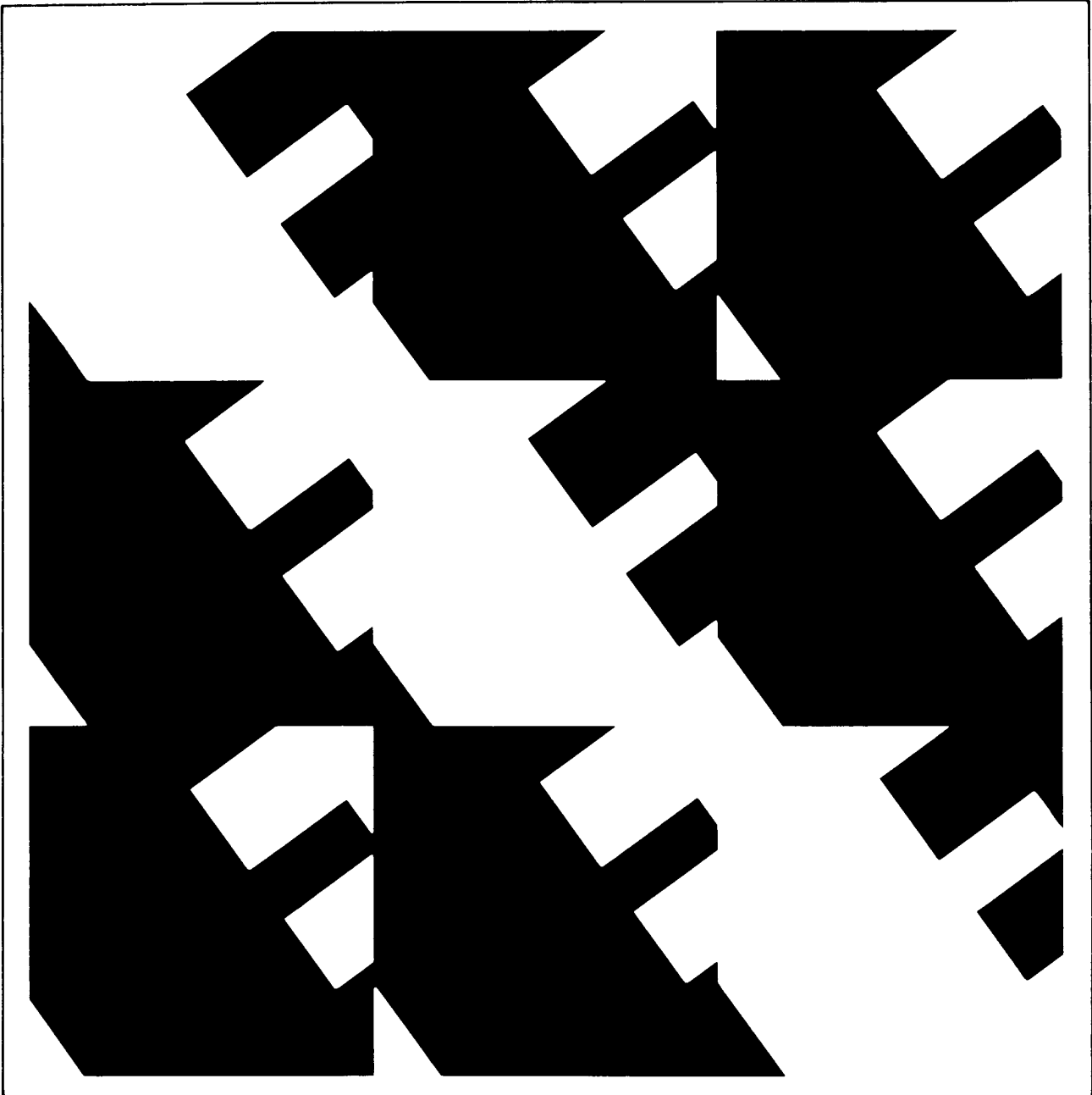


IEEE Standard Codes, Formats,
Protocols, and Common Commands
For Use with ANSI/IEEE Std 488.1-1987
IEEE Standard Digital Interface for
Programmable Instrumentation

ANSI/IEEE Std 488.2-1987



Published by The Institute of Electrical and Electronics Engineers, Inc. 345 East 47th Street, New York, NY 10017, USA
April 22, 1988

SH11353

An American National Standard

**IEEE Standard Codes, Formats,
Protocols, and Common Commands**

For Use with ANSI/IEEE Std 488.1-1987
IEEE Standard Digital Interface for
Programmable Instrumentation

Sponsor

**Automated Instrumentation Technical Committee
of the
IEEE Instrumentation and Measurement Society**

Approved June 11, 1987

IEEE Standards Board

Approved February 2, 1988

American National Standards Institute

ISBN 471-61871-3

© Copyright 1988 by

**The Institute of Electrical and Electronics Engineers, Inc
345 East 47th Street, New York, NY 10017**

*No part of this publication may be reproduced in any form,
in an electronic retrieval system or otherwise,
without the prior written permission of the publisher.*

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE which have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least once every five years for revision or reaffirmation. When a document is more than five years old, and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
345 East 47th Street
New York, NY 10017
USA

IEEE Standards documents are adopted by the Institute of Electrical and Electronics Engineers without regard to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standards documents.

Foreword

(This Foreword is not a part of ANSI/IEEE Std 488.2-1987, IEEE Standard Codes, Formats, Protocols, and Common Commands. For Use with ANSI/IEEE Std 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation.)

The IEEE Std 488 has been in existence for more than a decade during which time its power and versatility have been proven decisively. Probably its most important contribution to test system automation has been the standardization of system interconnection and communication. The extent, however, to which this standardization was developed was limited by the inability to clearly see the needs of the marketplace. Without experience to guide the decision making process, early efforts would have been detrimental to the acceptance of the standard.

Now the marketplace has matured, and many manufacturers, system integrators, and users have experienced the negative side of this limited standardization. Fortunately, a counterbalancing force of a large and growing base of design and application experience is available to further standardize the IEEE Std 488 bus. This standard is based upon that knowledge and will encourage a new level of growth and acceptance of the IEEE Std 488.

The concepts presented in this standard come from a variety of sources. The IEEE published a recommended practice (ANSI/IEEE Std 728-1982, IEEE Recommended Practice for Code and Format Conventions for use with ANSI/IEEE Std 488-1978). Since the introduction of ANSI/IEEE Std 488-1978, individual manufacturers of devices have developed internal standards that address the code, formats, protocol, syntax, and semantic concepts. System integrators have identified requirements useful in configuring systems. All these sources have been used and refined in developing this standard.

Minor changes were made to four interface functions in IEEE 488 when writing ANSI/IEEE Std 488.1-1987. The affected interface functions are Acceptor Handshake, Service Request, Device Clear, and Remote/Local. New, as well as experienced, designers should carefully review IEEE 488.1.

The additional standardization sought by ANSI/IEEE Std 488.2-1987 was founded on the premise that the existing investment in IEEE Std 488 resources must be protected. Thus, this document describes functionality that complements and is based upon the ANSI/IEEE Std 488.1-1987. This standard is intended to replace the ANSI/IEEE Std 728-1982 which will be withdrawn. Every effort was made to reap the benefits of standardization without limiting the freedom and creativity of the device designer.

At the time of approval of this standard the members of the working group were as follows:

Bob Cram, *Chairman*

Steven Barryte
Geoff Blyth
Anthony Bouwens
Bruce Choyce
Steve Coan
Gerry Cushing
Philip D'Angelo
Tonio Frühauf

Gary Gallagher
Stephen Greer
William Groseclose
Antal Hampel
Knud Johansen
Tod Johnson
Jeff Kodosky

Steve Lomas
Roger Oblad
Brian Pavlik
John Pieper
Armin Preuss
Larry Sollman
Stephen Tarr
Dana Trout

The following persons were on the balloting committee that approved this document for submission to the IEEE Standards Board:

David Ahlgren
John Barker
Steve Barryte
Richard Day
Ron Doss
Richard Drews
Gary Gallagher
Bernard Gollomp
Arnie Greenspan
Bill Gustafson

Larry Gross
Carl Hagerling
Chris Hancock
Faisal Imdad
B. Kowaluk
Robert Kurkjian
Thomas Leedy
Fred Liguri
Don Loughry
John McGlaughlin

Jerry Merritt
L. F. Moebus
Charles Osborn
Larry Ross
Eric Satcher
Milton Slade
J. R. Weger
Jim Weitenhagen
D. Williamson
Jim Zingg

When the IEEE Standards Board approved this standard on June 11, 1987, it had the following membership:

Donald C. Fleckenstein, *Chairman*

Marco W. Migliaro, *Vice Chairman*

Andrew G. Salem, *Secretary*

James H. Beall
Dennis Bodson
Marshall L. Cain
James M. Daly
Stephen R. Dillon
Eugene P. Fogarty
Jay Forster
Kenneth D. Hendrix
Irvin N. Howell

Leslie R. Kerr
Jack Kinn
Irving Kolodny
Joseph L. Koepfinger*
Edward Lohse
John May
Lawrence V. McCall
L. Bruce McClung
Donald T. Michael*

L. John Rankine
John P. Riganati
Gary S. Robinson
Frank L. Rose
Robert E. Rountree
Sava I. Sherr*
William R. Tackaberry
William B. Wilkens
Helen M. Wood

*Member emeritus



Contents

SECTION	PAGE
GENERAL CONCEPTS	
1. Introduction	19
1.1 Scope	19
1.2 Objectives	19
1.3 Notation	20
2. References	20
3. System Considerations	20
3.1 Definitions	20
3.2 System Message Traffic	21
3.2.1 Program and Response Message Relationships	21
3.2.2 Other Message Traffic	22
3.3 Functional Layers	22
DEVICE RELATED SPECIFICATIONS	
4. Device Compliance Criteria	24
4.1 IEEE 488.1 Requirements	24
4.2 Message Exchange Requirements	24
4.3 Syntax Requirements	24
4.3.1 Required Functional Elements	24
4.3.2 Optional Functional Elements	25
4.4 Status Reporting Requirements	26
4.4.1 Required Status Reporting Capability	26
4.4.2 Optional Status Reporting Capability	28
4.5 Common Commands	28
4.5.1 Required Common Commands	28
4.5.2 Optional Common Commands	28
4.6 Synchronization Requirements	31
4.7 System Configuration Capability	31
4.8 Controller Capability	31
4.8.1 Required Controller Capability	31
4.8.2 Optional Controller Capability	31
4.9 Device Documentation Requirements	32
5. Device Interface Function Requirements	33
5.1 Handshake Requirements	33
5.1.1 Source Handshake Requirements	33
5.1.2 Acceptor Handshake Requirements	33
5.2 Address Requirements	33
5.3 Talker Requirements	33
5.4 Listener Requirements	33
5.5 Service Request Requirements	34
5.6 Remote/Local Requirements	34
5.6.1 Control and Operation Definitions	34
5.6.2 IEEE 488.1 Subset Requirements	34
5.6.3 Local-to-Remote State Transition Requirements	34
5.6.4 Remote-to-Local State Transition Requirements	34
5.6.5 Local State Operation	35
5.6.6 Remote State Operation	35
5.6.7 Operation Independent of Remote/Local State	35
5.6.8 Remote/Local Indicator Requirements	35

SECTION	PAGE
5.7	Parallel Poll Requirements 35
5.8	Device Clear Requirements 35
5.9	Device Trigger Requirements 36
5.10	Controller Function Requirements 36
5.11	Electrical Requirements 36
5.12	Power-On Requirements 36
5.12.1	Items Not Affected by Power-On 36
5.12.2	Items Dependent Upon Power-On-Status-Clear Flag 36
5.12.3	Items That May be Affected by Power-On 36
6.	Message Exchange Control Protocol 37
6.1	Functional Elements 37
6.1.1	IEEE 488.1 Bus 38
6.1.2	Status Reporting 38
6.1.3	Message Exchange Interface 38
6.1.4	I/O Control 39
6.1.5	Input Buffer 41
6.1.6	Parser 41
6.1.7	Execution Control 43
6.1.8	Device Functions 44
6.1.9	Response Formatter 44
6.1.10	Output Queue 45
6.1.11	Trigger Control 45
6.2	Protocol Overview 45
6.2.1	Initialization 46
6.2.2	Command Processing 48
6.2.3	Query Processing 48
6.3	Message Exchange Control Operation 48
6.3.1	Message Exchange Control States 48
6.3.2	Message Exchange Control Transition Actions 50
6.4	Protocol Rules 51
6.4.1	Program Message Transfer 51
6.4.2	Message Source Independence 51
6.4.3	Message Exchange Sequence 51
6.4.4	Compound Queries 52
6.4.5	Message Order Requirements 53
6.5	Protocol Exceptions 54
6.5.1	Aborted Messages 54
6.5.2	Addressed to Talk with Nothing to Say 54
6.5.3	No Listener on Bus 54
6.5.4	Command Error 54
6.5.5	Execution Error 54
6.5.6	Device-specific Error 54
6.5.7	Query Error 54
7.	Device Listening Formats 55
7.1	Overview 55
7.1.1	Device Command Set Generation 55
7.1.2	Encoding Syntax 55
7.2	Notation 56
7.2.1	Diagramming Syntactic Flow 56
7.2.2	Synactic Elements 56
7.2.3	Special Symbols 58
7.3	Terminated Program Messages — Functional Syntax 58
7.3.1	Function 58

SECTION	PAGE
7.3.2	Syntax 58
7.3.3	Functional Element Summary 62
7.4	Separator Functional Elements 63
7.4.1	<PROGRAM MESSAGE UNIT SEPARATOR> 63
7.4.2	<PROGRAM DATA SEPARATOR> 63
7.4.3	<PROGRAM HEADER SEPARATOR> 64
7.5	<PROGRAM MESSAGE TERMINATOR> 64
7.5.1	Function 64
7.5.2	Encoding Syntax 64
7.5.3	Semantic Equivalence 65
7.6	Program Header Functional Elements 65
7.6.1	<COMMAND PROGRAM HEADER> 65
7.6.2	<QUERY PROGRAM HEADER> 67
7.7	<PROGRAM DATA> Functional Elements 69
7.7.1	<CHARACTER PROGRAM DATA> 69
7.7.2	<DECIMAL NUMERIC PROGRAM DATA> 69
7.7.3	<SUFFIX PROGRAM DATA> 71
7.7.4	<NON-DECIMAL NUMERIC PROGRAM DATA> 74
7.7.5	<STRING PROGRAM DATA> 76
7.7.6	<ARBITRARY BLOCK PROGRAM DATA> 77
7.7.7	<EXPRESSION PROGRAM DATA> 78
8.	Device Talking Formats 79
8.1	Overview 79
8.2	Notation 79
8.3	Terminated Response Messages—Functional Syntax 79
8.3.1	Function 79
8.3.2	Syntax 79
8.3.3	Functional Element Summary 82
8.4	Separator Functional Elements 83
8.4.1	<RESPONSE MESSAGE UNIT SEPARATOR> 83
8.4.2	<RESPONSE DATA SEPARATOR> 83
8.4.3	<RESPONSE HEADER SEPARATOR> 84
8.5	<RESPONSE MESSAGE TERMINATOR> 84
8.5.1	Function 84
8.5.2	Encoding Syntax 84
8.6	<RESPONSE HEADER> 84
8.6.1	Function 84
8.6.2	Encoding Syntax 84
8.6.3	Rules 85
8.7	<RESPONSE DATA> Functional Elements 85
8.7.1	<CHARACTER RESPONSE DATA> 86
8.7.2	<NR1 NUMERIC RESPONSE DATA> 86
8.7.3	<NR2 NUMERIC RESPONSE DATA> 86
8.7.4	<NR3 NUMERIC RESPONSE DATA> 87
8.7.5	<HEXADECIMAL NUMERIC RESPONSE DATA> 88
8.7.6	<OCTAL NUMERIC RESPONSE DATA> 88
8.7.7	<BINARY NUMERIC RESPONSE DATA> 89
8.7.8	<STRING RESPONSE DATA> 90
8.7.9	<DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA> 90
8.7.10	<INDEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA> 91
8.7.11	<ARBITRARY ASCII RESPONSE DATA> 91
9.	Message Data Coding 92
9.1	ASCII 7-bit Codes 92

SECTION	PAGE
9.2 Binary 8-bit Integer Codes	93
9.2.1 Byte Order and Data Line Relationships	93
9.2.2 Binary Integer Code	93
9.2.3 Binary Unsigned Integer Code	93
9.3 Binary Floating Point Code	93
9.3.1 Floating Point Code Fields.....	93
9.3.2 Basic Formats.....	94
9.3.3 Order of Transmission	94
9.3.4 Example of Single Precision Number	95
10. Common Commands and Queries	96
10.1 *AAD, Accept Address Command	98
10.1.1 Function and Requirements	98
10.1.2 Syntax	98
10.1.3 Semantics	98
10.1.4 Related Common Commands	98
10.1.5 Standard Compliance	98
10.1.6 Error Handling	98
10.2 *CAL?, Calibration Query	98
10.2.1 Function and Requirements	98
10.2.2 Query Structure	98
10.2.3 Response Syntax	98
10.2.4 Related Common Commands	98
10.2.5 Standard Compliance	98
10.3 *CLS, Clear Status Command	98
10.3.1 Function and Requirements	98
10.3.2 Syntax	98
10.3.3 Semantics	98
10.3.4 Related Common Commands	98
10.3.5 Standard Compliance	98
10.4 *DDT, Define Device Trigger Command	98
10.4.1 Function and Requirements	98
10.4.2 Syntax	99
10.4.3 Semantics	99
10.4.4 Related Common Commands	99
10.4.5 Standard Compliance	99
10.4.6 Error Handling	99
10.4.7 Example.....	99
10.5 *DDT?, Define Device Trigger Query	99
10.5.1 Function and Requirements	99
10.5.2 Query Structure	99
10.5.3 Response Semantics	99
10.5.4 Related Common Commands	99
10.5.5 Standard Compliance	99
10.6 *DLF, Disable Listener Function Command.....	99
10.6.1 Function and Requirements	99
10.6.2 Syntax	99
10.6.3 Semantics	99
10.6.4 Related Common Commands	100
10.6.5 Standard Compliance	100
10.6.6 Error Handling	100
10.7 *DMC, Define Macro Command	100
10.7.1 Function and Requirements	100
10.7.2 Syntax	100

SECTION	PAGE
10.7.3 Semantics	100
10.7.4 Related Common Commands	100
10.7.5 Standard Compliance	100
10.7.6 Error Handling	100
10.7.7 Examples	101
10.8 *EMC, Enable Macro Command	101
10.8.1 Function and Requirements	101
10.8.2 Syntax	101
10.8.3 Semantics	101
10.8.4 Related Common Commands	101
10.8.5 Standard Compliance	101
10.8.6 Error Handling	101
10.8.7 Example	101
10.9 *EMC?, Enable Macro Query	102
10.9.1 Function and Requirements	102
10.9.2 Query Structure	102
10.9.3 Response Semantics	102
10.9.4 Related Common Commands	102
10.9.5 Standard Compliance	102
10.10 *ESE, Standard Event Status Enable Command	102
10.10.1 Function and Requirements	102
10.10.2 Syntax	102
10.10.3 Semantics	102
10.10.4 Related Common Commands	102
10.10.5 Standard Compliance	102
10.10.6 Error Handling	102
10.11 *ESE?, Standard Event Status Enable Query	102
10.11.1 Function and Requirements	102
10.11.2 Query Structure	102
10.11.3 Response Semantics	102
10.11.4 Related Common Commands	102
10.11.5 Standard Compliance	102
10.12 *ESR?, Standard Event Status Register Query	102
10.12.1 Function and Requirements	102
10.12.2 Query Structure	103
10.12.3 Response Semantics	103
10.12.4 Related Common Commands	103
10.12.5 Standard Compliance	103
10.13 *GMC?, Get Macro Contents Query	103
10.13.1 Function and Requirements	103
10.13.2 Query Structure	103
10.13.3 Semantics	103
10.13.4 Related Common Commands	103
10.13.5 Standard Compliance	103
10.13.6 Error Handling	103
10.14 *IDN?, Identification Query	103
10.14.1 Function and Requirements	103
10.14.2 Query Structure	103
10.14.3 Response Semantics	103
10.14.4 Related Common Commands	104
10.14.5 Standard Compliance	104
10.14.6 Example	104
10.15 *IST?, Individual Status Query	104
10.15.1 Function and Requirements	104

SECTION	PAGE
10.15.2 Query Structure	104
10.15.3 Response Semantics	104
10.15.4 Related Common Commands	104
10.15.5 Standard Compliance	104
10.16 *LMC?, Learn Macro Query	104
10.16.1 Function and Requirements	104
10.16.2 Query Structure	104
10.16.3 Response Semantics	104
10.16.4 Related Common Commands	104
10.16.5 Standard Compliance	104
10.17 *LRN?, Learn Device Setup Query	104
10.17.1 Function and Requirements	104
10.17.2 Query Structure	105
10.17.3 Response Semantics	105
10.17.4 Related Common Commands	105
10.17.5 Standard Compliance	105
10.18 *OPC, Operation Complete Command	105
10.18.1 Function and Requirements	105
10.18.2 Syntax	105
10.18.3 Semantics	105
10.18.4 Related Common Commands	105
10.18.5 Standard Compliance	105
10.19 *OPC?, Operation Complete Query	105
10.19.1 Function and Requirements	105
10.19.2 Query Structure	105
10.19.3 Response Semantics	105
10.19.4 Related Common Commands	105
10.19.5 Standard Compliance	105
10.20 *OPT?, Option Identification Query	105
10.20.1 Function and Requirements	105
10.20.2 Query Structure	105
10.20.3 Response Semantics	105
10.20.4 Related Common Commands	106
10.20.5 Standard Compliance	106
10.21 *PCB, Pass Control Back	106
10.21.1 Function and Requirements	106
10.21.2 Syntax	106
10.21.3 Semantics	106
10.21.4 Related Common Commands	106
10.21.5 Standard Compliance	106
10.21.6 Error Handling	106
10.22 *PMC, Purge Macros Command	106
10.22.1 Function and Requirements	106
10.22.2 Syntax	106
10.22.3 Semantics	106
10.22.4 Related Common Commands	106
10.22.5 Standard Compliance	106
10.23 *PRE, Parallel Poll Enable Register Enable Command	106
10.23.1 Function and Requirements	106
10.23.2 Syntax	106
10.23.3 Semantics	107
10.23.4 Related Common Commands	107
10.23.5 Standard Compliance	107
10.23.6 Error Handling	107

SECTION	PAGE
10.24 *PRE?, Parallel Poll Enable Register Enable Query	107
10.24.1 Function and Requirements	107
10.24.2 Query Structure	107
10.24.3 Response Semantics	107
10.24.4 Related Common Commands	107
10.24.5 Standard Compliance	107
10.25 *PSC, Power-On Status Clear Command	107
10.25.1 Function and Requirements	107
10.25.2 Syntax	107
10.25.3 Semantics	107
10.25.4 Related Common Commands	107
10.25.5 Standard Compliance	107
10.25.6 Error Handling	107
10.25.7 Example	107
10.26 *PSC?, Power-On Status Clear Query	107
10.26.1 Function and Requirements	107
10.26.2 Query Structure	108
10.26.3 Response Semantics	108
10.26.4 Related Common Commands	108
10.26.5 Standard Compliance	108
10.27 *PUD, Protected User Data Command	108
10.27.1 Function and Requirements	108
10.27.2 Syntax	108
10.27.3 Semantics	108
10.27.4 Related Common Commands	108
10.27.5 Standard Compliance	108
10.27.6 Error Handling	108
10.28 *PUD?, Protected User Data Query	108
10.28.1 Function and Requirements	108
10.28.2 Query Structure	108
10.28.3 Response Semantics	108
10.28.4 Related Common Commands	108
10.28.5 Standard Compliance	108
10.29 *RCL, Recall Command	108
10.29.1 Function and Requirements	108
10.29.2 Syntax	109
10.29.3 Semantics	109
10.29.4 Related Common Commands	109
10.29.5 Standard Compliance	109
10.29.6 Error Handling	109
10.30 *RDT, Resource Description Transfer Command	109
10.30.1 Function and Requirements	109
10.30.2 Syntax	109
10.30.3 Semantics	109
10.30.4 Related Common Commands	109
10.30.5 Standard Compliance	109
10.30.6 Error Handling	109
10.31 *RDT?, Resource Description Transfer Query	109
10.31.1 Function and Requirements	109
10.31.2 Query Structure	109
10.31.3 Response Semantics	110
10.31.4 Related Common Commands	110
10.31.5 Standard Compliance	110
10.32 *RST, Reset Command	110

SECTION	PAGE
10.32.1 Function and Requirements	110
10.32.2 Syntax	110
10.32.3 Semantics	110
10.32.4 Related Common Commands	110
10.32.5 Standard Compliance	110
10.33 *SAV, Save Command	110
10.33.1 Function and Requirements	110
10.33.2 Syntax	110
10.33.3 Semantics	110
10.33.4 Related Common Commands	110
10.33.5 Standard Compliance	110
10.33.6 Error Handling	110
10.34 *SRE, Service Request Enable Command	111
10.34.1 Function and Requirements	111
10.34.2 Syntax	111
10.34.3 Semantics	111
10.34.4 Related Common Commands	111
10.34.5 Standard Compliance	111
10.34.6 Error Handling	111
10.35 *SRE?, Service Request Enable Query	111
10.35.1 Function and Requirements	111
10.35.2 Query Structure	111
10.35.3 Response Semantics	111
10.35.4 Related Common Commands	111
10.35.5 Standard Compliance	111
10.36 *STB?, Read Status Byte Query	111
10.36.1 Function and Requirements	111
10.36.2 Query Structure	111
10.36.3 Response Semantics	111
10.36.4 Related Common Commands	111
10.36.5 Standard Compliance	111
10.37 *TRG, Trigger Command	111
10.37.1 Function and Requirements	111
10.37.2 Syntax	112
10.37.3 Semantics	112
10.37.4 Related Common Commands	112
10.37.5 Standard Compliance	112
10.38 *TST?, Self-Test Query	112
10.38.1 Function and Requirements	112
10.38.2 Query Structure	112
10.38.3 Response Semantics	112
10.38.4 Related Common Commands	112
10.38.5 Standard Compliance	112
10.39 *WAI, Wait-to-Continue Command	112
10.39.1 Function and Requirements	112
10.39.2 Syntax	112
10.39.3 Semantics	112
10.39.4 Related Common Commands	112
10.39.5 Standard Compliance	112
11. Device Status Reporting	112
11.1 Overview	113
11.1.1 Operation	113
11.1.2 Summary of Related Common Commands	114

SECTION	PAGE
11.1.3	Related IEEE 488.1-Defined Operations..... 115
11.2	Status Byte Register..... 116
11.2.1	Definition..... 116
11.2.2	Reading the Status Byte Register..... 116
11.2.3	Writing the Status Byte Register..... 117
11.2.4	Clearing the Status Byte Register..... 117
11.3	Service Request Enabling..... 117
11.3.1	Operation..... 117
11.3.2	Service Request Enable Register..... 117
11.3.3	Service Request Generation..... 118
11.4	Status Data Structures..... 122
11.4.1	Overview..... 122
11.4.2	Status Data Structure — Register Model..... 122
11.4.3	Status Data Structure — Queue Model..... 125
11.5	Standard Status Data Structures..... 126
11.5.1	Standard Event Status Register Model..... 127
11.5.2	Standard Queue Model..... 130
11.6	Parallel Poll Response Handling..... 131
11.6.1	Parallel Poll Enable Register..... 131
11.6.2	Reading ist Without a Parallel Poll..... 132

SYSTEM RELATED SPECIFICATIONS

12.	Device/Controller Synchronization Techniques..... 132
12.1	Overview..... 132
12.2	Sequential and Overlapped Commands..... 132
12.2.1	Illustration of Sequential Commands..... 132
12.2.2	Illustration of Overlapped Commands..... 133
12.3	Pending-Operation Flag..... 133
12.4	No-Operation-Pending Flag..... 134
12.5	Controller/Device Synchronization Commands..... 134
12.5.1	The *WAI Common Command..... 134
12.5.2	The *OPC Common Command..... 135
12.5.3	The *OPC? Common Query..... 137
12.6	Synchronization with External-Control-Signals..... 138
12.7	Improper Usage of *OPC and *OPC?..... 138
12.8	Design Considerations..... 139
12.8.1	Overlapped Commands..... 139
12.8.2	Execution Error Handling..... 140
12.8.3	Operation Complete..... 140
13.	Automatic System Configuration..... 141
13.1	Introduction..... 141
13.2	Overview..... 141
13.3	Generic Approach to Automatic System Configuration..... 144
13.3.1	Address Assignment..... 144
13.3.2	Device Identification..... 146
13.4	Detailed Requirements of the Auto Configure Commands..... 147
13.4.1	*DLF Common Command Requirements..... 147
13.4.2	*AAD Common Command Requirements..... 147
13.5	Additional Automatic Configuration Techniques..... 152
13.6	Examples..... 152

CONTROLLER RELATED SPECIFICATIONS

14. Controller Compliance Criteria	152
14.1 IEEE 488.1 Requirements	152
14.2 Message Exchange Requirements	153
14.2.1 Required Control Sequences	153
14.2.2 Optional Control Sequences	153
14.3 Protocols	153
14.3.1 Required Protocols	153
14.3.2 Optional Protocols	154
14.4 Functional Element Handling	154
14.5 Controller Specification Requirements	154
15. IEEE 488.2 — Controller Requirements	154
15.1 Controller Interface Function Requirements	154
15.1.1 Controller Requirements	154
15.1.2 Talker Requirements	154
15.1.3 Listener Requirements	155
15.1.4 Passing Control Requirements	155
15.1.5 Electrical Requirements	155
15.2 Additional IEEE 488.2 — Controller Requirements	155
15.3 IEEE 488.2 — Controller Recommendations	155
15.3.1 Monitoring Bus Lines	155
15.3.2 Timeouts	155
15.3.3 SRQ Interrupts	155
16. Controller Message Exchange Protocols	156
16.1 Definitions	156
16.1.1 IEEE 488.1 Driver	156
16.1.2 Programming Environment	156
16.1.3 Application Program	156
16.1.4 IEEE 488.2 Controller	156
16.1.5 IEEE 488.1 Bus Signals	156
16.1.6 DAB	156
16.1.7 END	156
16.1.8 Control Sequence	156
16.1.9 Addresses	156
16.1.10 IEEE 488.1 State Conditions	157
16.1.11 Data Messages	157
16.1.12 Controller Errors	157
16.2 Control Sequences	157
16.2.1 SEND COMMAND	157
16.2.2 SEND SETUP	157
16.2.3 SEND DATA BYTES	158
16.2.4 SEND	158
16.2.5 RECEIVE SETUP	158
16.2.6 RECEIVE RESPONSE MESSAGE	159
16.2.7 RECEIVE	160
16.2.8 SEND IFC	160
16.2.9 DEVICE CLEAR	160
16.2.10 ENABLE LOCAL CONTROLS	160
16.2.11 ENABLE REMOTE	161
16.2.12 SET RWLS	161
16.2.13 SEND LLO	161
16.2.14 PASS CONTROL	162

SECTION	PAGE
16.2.15 PERFORM PARALLEL POLL.....	162
16.2.16 PARALLEL POLL CONFIGURE	162
16.2.17 PARALLEL POLL UNCONFIGURE.....	162
16.2.18 READ STATUS BYTE.....	162
16.2.19 TRIGGER	163
17. Common Controller Protocols.....	163
17.1 Reset Protocol	164
17.1.1 Keyword.....	164
17.1.2 Purpose	164
17.1.3 Information Requested by the Protocol	164
17.1.4 Information Supplied by the Protocol	164
17.1.5 Controller Algorithm.....	164
17.1.6 Additional Requirements and Guidelines	164
17.1.7 Standard Compliance	164
17.2 Find Device Requesting Service Protocol	165
17.2.1 Keyword.....	165
17.2.2 Purpose	165
17.2.3 Information Requested by the Protocol	165
17.2.4 Information Supplied by the Protocol	165
17.2.5 Controller Algorithm.....	165
17.2.6 Additional Requirements and Guidelines	165
17.2.7 Standard Compliance	166
17.3 Serial Poll All Devices Protocol	166
17.3.1 Keyword.....	166
17.3.2 Purpose	166
17.3.3 Information Requested by the Protocol	166
17.3.4 Information Supplied by the Protocol	166
17.3.5 Controller Algorithm.....	166
17.3.6 Additional Requirements and Guidelines	166
17.3.7 Standard Compliance	166
17.4 Pass Control Protocol	166
17.4.1 Keyword.....	166
17.4.2 Purpose	166
17.4.3 Information Requested by the Protocol	167
17.4.4 Information Supplied by the Protocol	167
17.4.5 Controller Algorithm.....	167
17.4.6 Additional Requirements and Guidelines	167
17.4.7 Standard Compliance	167
17.5 Requesting Control.....	167
17.5.1 Keyword.....	167
17.5.2 Purpose	167
17.5.3 Information Requested by the Protocol	167
17.5.4 Information Supplied by the Protocol	167
17.5.5 Controller Algorithm.....	167
17.5.6 Additional Requirements and Guidelines	168
17.5.7 Standard Compliance	186
17.6 Find Listeners Protocol.....	168
17.6.1 Keyword.....	168
17.6.2 Purpose	168
17.6.3 Information Supplied to the Protocol.....	168
17.6.4 Information Supplied by the Protocol	168
17.6.5 Controller Algorithm.....	168
17.6.6 Additional Requirements and Guidelines	169

SECTION	PAGE
17.6.7 Standard Compliance	169
17.7 Set Address Protocol	169
17.7.1 Keyword	169
17.7.2 Purpose	169
17.7.3 Information Requested by the Protocol	169
17.7.4 Information Supplied by the Protocol	169
17.7.5 Controller Algorithm	169
17.7.6 Additional Requirements and Guidelines	173
17.7.7 Standard Compliance	173
17.8 Test System Protocol	173
17.8.1 Keyword	173
17.8.2 Purpose	173
17.8.3 Information Requested by the Protocol	173
17.8.4 Information Supplied by the Protocol	173
17.8.5 Controller Algorithm	174
17.8.6 Additional Requirements and Guidelines	174
17.8.7 Standard Compliance	174

FIGURES

Fig 3-1 Usual Message Traffic	21
Fig 3-2 IEEE 488.1 and IEEE 488.2 Functional Protocol Layers	23
Fig 4-1 Required Status Reporting Capabilities	27
Fig 6-1 Device Status and Message Exchange Overview	37
Fig 6-2 Message Exchange Control Interface, Functional Blocks	38
Fig 6-3 Message Exchange Control State Diagram (Simplified)	46
Fig 6-4 Message Exchange Control State Diagram (Complete)	47
Fig 7-1 <TERMINATED PROGRAM MESSAGE> Functional Element Syntax	59
Fig 7-2 <PROGRAM MESSAGE> Functional Element Syntax	59
Fig 7-3 <PROGRAM MESSAGE UNIT> Functional Element Syntax	59
Fig 7-4 <COMMAND MESSAGE UNIT> Functional Element Syntax	60
Fig 7-5 <QUERY MESSAGE UNIT> Functional Element Syntax	60
Fig 7-6 <PROGRAM DATA> Functional Element Syntax	61
Fig 8-1 <TERMINATED RESPONSE MESSAGE> Functional Element Syntax	80
Fig 8-2 <RESPONSE MESSAGE> Functional Element Syntax	80
Fig 8-3 <RESPONSE MESSAGE UNIT> Functional Element Syntax	80
Fig 8-4 <RESPONSE DATA> Functional Element Syntax	81
Fig 11-1 IEEE 488.2 Status Reporting Structure Overview	113
Fig 11-2 Status Byte Register	114
Fig 11-3 Service Request Enabling	118
Fig 11-4 Service Request Generation	119
Fig 11-5 Set rsv State Diagram	120
Fig 11-6 Status Data Structure — Register Model	123
Fig 11-7 Status Data Structure — Queue Model	126
Fig 11-8 Standard Status Data Structures Overview	127
Fig 11-9 Standard Event Status Register Model	128
Fig 11-10 Standard Queue Model	130
Fig 11-11 Parallel Poll Response Handling Data Structure	131
Fig 12-1 Pending Operation Diagram of Selected Overlapped Commands	133
Fig 12-2 Timing Diagram — Sample Program Message not using *WAI Command	134
Fig 12-3 Timing Diagram — Sample Program Message using *WAI Command	135
Fig 12-4 Operation Complete Command Diagram	135
Fig 12-5 Timing Diagram — Sample Program Message Using the *OPC Command	136
Fig 12-6 Operation Complete Query Diagram	137
Fig 12-7 Timing Diagram — Improper Operation Example 1	138

FIGURES	PAGE
Fig 12-8 Timing Diagram — Improper Operation Example 2	139
Fig 13-1 Comprehensive Device Search	141
Fig 13-2 Non-Address-Configurable Device Search	142
Fig 13-3 Address-Configurable Device Search	142
Fig 13-4 Find Listeners Protocol	142
Fig 13-5 Set Address Protocol	143
Fig 13-6 Accept Address Common Command	144
Fig 16-1 Controller Model	156

TABLES	
Table 4-1 IEEE 488.1 Interface Requirements for Devices	24
Table 4-2 Required Functional Elements for Devices	25
Table 4-3 Optional Functional Elements for Devices	25
Table 4-4 Required Status Reporting Common Commands	26
Table 4-5 Optional Power-On Common Commands	28
Table 4-6 Optional Parallel Poll Common Commands	28
Table 4-7 Required Internal Operation Common Commands	28
Table 4-8 Optional Resource Description Common Commands	29
Table 4-9 Optional Protected User Data Commands	29
Table 4-10 Optional Calibration Command	29
Table 4-11 Optional Trigger Command	29
Table 4-12 Optional Trigger Macro Commands	30
Table 4-13 Optional Macro Commands	30
Table 4-14 Optional Option Identification Command	30
Table 4-15 Optional Stored Setting Commands	30
Table 4-16 Optional Learn Command	31
Table 4-17 Required Synchronization Commands	31
Table 4-18 Optional System Configuration Commands	31
Table 4-19 Optional Passing Control Command	32
Table 7-1 <suffix unit> Elements	72
Table 7-2 Allowed <suffix mult.> Mnemonics	74
Table 9-1 ASCII 7-bit Code	92
Table 9-2 ASCII 7-bit Code Chart	92
Table 10-1 IEEE 488.2 Common Command Headers	96
Table 10-2 IEEE 488.2 Common Command Groups	97
Table 11-1 Status Reporting Common Commands	114
Table 11-2 Status Reporting IEEE 488.1-defined Operations	115
Table 11-3 Set rsv State Diagram Mnemonics	121
Table 11-4 Set rsv State Diagram Message Output	121
Table 14-1 IEEE 488.1 Requirements	152
Table 14-2 Required Control Sequences	153
Table 14-3 Optional Control Sequences	153
Table 14-4 Required Protocols	153
Table 14-5 Optional Protocols	154
Table 17-1 IEEE 488.2 Common Controller Protocols	164

APPENDIXES

Appendix A Compound Headers Usage and Examples	175
A1. Compound Header Organization Example Using a Tree	175
A1.1 Use of the Compound Header Rules	176
A1.2 Enhanced Tree Walking Implementation	176
A2. Compound Header Organization Example Using a Graph	177

APPENDIXES	PAGE
Appendix B Device/Controller Synchronization Techniques	178
B1. Simple Device and Application Program Synchronization	178
B2. Types of Devices	179
B2.1 Hypothetical Stimulus-Device	179
B2.2 Hypothetical Response-Device	179
B2.3 Hypothetical Controller Language	180
B3. Synchronization with Stimulus-Devices	181
B3.1 Stimulus-Device Synchronization using a <RESPONSE MESSAGE>	181
B3.2 Stimulus-Device Synchronization with Service Request	181
B3.3 Stimulus-Device Synchronization Without <PROGRAM MESSAGE> Elements	182
B4. Synchronization with Response-Devices	183
B4.1 Simple Response-Device Synchronization	183
B4.2 Response-Device Synchronization with Service Request	184
B4.3 Response-Device Synchronization Without <PROGRAM MESSAGE> Elements	184
B4.4 Device Communications While Waiting for a Measurement	185
B4.5 Synchronization using an External Control Signal	185
B4.6 Example Involving Simultaneous Trigger of Two Response-Devices	185
B5. Generalized Synchronization Independent of Queries	187
B6. Device Synchronization using the *WAI command	188
B7. System Example Involving both a Response- and Stimulus-Device	188
Appendix C Automatic System Configuration Example	190
C1. Overall Flow of the Protocol	190
C2. Description of the Identifier Searches	191
C3. Description of the Character Search	192
C3.1 The Search Initiation Character	192
C3.2 The Search Body	192
C4. Detailed Description of the Search Process	193
C4.1 Manufacturer Search for the Three Sample Identifiers	193
C4.2 Detection of a Superset Field	193
APPENDIX FIGURES	
Fig A-1 Compound Header Organization Example Using a Tree	175
Fig A-2 Compound Header Organization Using a Graph Example 1	177
Fig A-3 Compound Header Organization Using a Graph Example 2	178
Fig B-1 System Topology for Response- and Stimulus-Device Example	189
Fig C-1 Overall Protocol Flow	190
Fig C-2 Identifier Searches	191
Fig C-3 Character Search Elements	192
Fig C-4 Search Body	193
Fig C-5 Character Search Action for a Manufacturer Character and an end-of-field <eof>	194
Fig C-6 Detection of the Superset field BB	195
APPENDIX TABLE	
Table C-1 Device Identifiers/Assigned Addresses	190
INDEX	
Index for IEEE 488.2	197

An American National Standard

IEEE Standard Codes, Formats, Protocols, and Common Commands

For Use with ANSI/IEEE Std 488.1-1988, IEEE Standard Digital
Interface for Programmable Instrumentation

1. Introduction

1.1 Scope. This standard specifies a set of codes and formats to be used by devices connected via the IEEE 488.1 bus. This standard also defines communication protocols necessary to effect application independent device-dependent message exchanges and further defines common commands and characteristics useful in instrument system applications.

This standard is intended to directly apply to small to medium-scale instrument systems. It applies to systems comprised mainly of measurement, stimulus, and interconnect devices with an instrumentation controller. The standard may also apply to certain devices outside the scope of the instrument system environment.

As well as defining a variety of device-dependent messages, this standard extends and further interprets certain interface functions contained in ANSI/IEEE Std 488.1-1987 [2]¹ while remaining compatible with that standard.²

This standard covers the following topics:

- (1) IEEE 488.1 subsets
- (2) Standard message handling protocols including error handling
- (3) Unambiguous program and response message syntactic structures
- (4) Common commands useful in a wide range of instrument system applications
- (5) Standard status reporting structures
- (6) System configuration and synchronization protocols

Use of this standard does not relieve the user from the burden of responsibility for system compatibility at the application level. The user must be familiar with the characteristics of all the system components in order to configure an optimum system.

The intended readers of this standard include both controller and device designers.

1.2 Objectives. The objectives of this standard are:

- (1) To provide a well-defined and unambiguous structure of codes, formats, protocols, and common commands.
- (2) To retain generality to accommodate the needs of a wide variety of applications within the scope of the standard.
- (3) To promote the degree to which devices from different manufacturers may be interconnected and used, without modification.
- (4) To enable the interconnection of instrumentation and related devices with both limited and extensive capability to generate, process and interpret a variety of different message types.

¹ The numbers in brackets correspond to those of the references, Section 2 of this standard.

² ANSI/IEEE Std 488.1-1987 shall be hereafter referred to in this document as IEEE 488.1.

(5) To define codes, formats, protocols, and common commands that will reduce the costs of generating application software and the costs of system integration.

(6) To permit direct communication among instrument system devices without extraordinary translation and conversion of special codes and formats.

1.3 Notation. This standard defines several common English words to have special meaning in the context of this standard. These words have different connotations in IEEE 488.1, but for readability and conciseness of text are the preferred words for use in IEEE 488.2. To avoid confusion, these words always appear in bold type when referenced in this document. Words which appear in bold type are: **system**, **device**, **controller**, **system bus**, and **system interface**. Local messages peculiar to IEEE 488.2 will also appear in lower case bold type.

This standard also defines syntactic elements used to describe messages transferred on the bus. Syntactic elements are enclosed by angle brackets, for example, <syntactic element>, to set them off from local messages, remote messages, and normal text.

2. References

This standard shall be used in conjunction with the following publications:

[1] ANSI/IEEE Std 260-1978, (R 1985). IEEE Standard Letter Symbols for Units of Measurement (SI Units, Customary Inch-Pound Units, and Certain Other Units).³

[2] ANSI/IEEE Std 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation.

[3] ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic.

[4] ANSI X3.4-1986, American National Standard Code for Information Interchange Coded Character Set—7-Bit.

[5] ANSI X3.42-1975, American National Standard Representation of Numeric Values in Character Strings for Information Interchange.

[6] ISO Std 2955-1983, Information processing—Representation of SI and other units in systems with limited character sets. Second edition, 1983-05-15.⁴

3. System Considerations

3.1 Definitions. The following definitions apply for the purpose of this standard. This section contains only general definitions. Detailed definitions are given in further sections, as appropriate. For definitions relating specifically to IEEE 488.1, see 1.3 of that standard.

³ ANSI/IEEE publications can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY 10018, or from the Service Center, The Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331.

⁴ ISO documents are available in the US from the Sales department, American National Standards Institute, 1430 Broadway, New York, NY 10018, USA, ISO documents also available from the ISO Office, 1, rue de Varembe, Case postale 56, CH-1211, Genève 20, Switzerland/Suisse.

controller. The component of a **system** that functions as the system-controller. A **controller** typically sends program messages to and receives response messages from **devices**. A **controller** may pass and receive control per the protocols in this standard. A **controller** meets all the requirements stated in Section 14 of this standard.

device. A component of a **system** that does not function as the system-controller but typically receives program messages from and sends response messages to the **controller**. A **device** may optionally have the capability to receive control from the **controller** and become the controller-in-charge of the **system**.⁵ A **device** meets all the requirements stated in Section 4 of this standard.

system. A group of **devices** and a **controller** interconnected with a **system interface**.

system bus. The IEEE 488.1 bus and protocols that interconnect the **devices** and **controllers** in a **system**. The content of this standard applies to device-dependent traffic over this bus.

system interface. An interface which connects a **device** or **controller** to the **system bus**.

A “non-IEEE 488.2 system interface” is any interface other than the **system interface** which may happen to be connected to a **device** or **controller**.

3.2 System Message Traffic. This standard is optimized for a **system** in which the **devices** do not become controller-in-charge. The usual message traffic is assumed to be from “controller-to-device” or from “device-to-controller” (see Fig 3-1).

The flexibility of the program message syntax (Section 7) also may allow the use of a IEEE 488.2-component in a system which contains non-IEEE 488.2 components.

In a system with a **device** and a noncompliant controller, the flexible listening formats allow the achievement of a higher degree of compatibility. This compatibility is realized by requiring minor variations in syntax to be accepted by the **device**. These variations are designed to be syntaxes that are easy to generate on a variety of controllers. They may even be the noncompliant controller's default syntax. Such systems are beyond the scope of this standard and may not perform their intended functions.

3.2.1 Program and Response Message Relationships. Subsequent sections of this standard will describe in detail the allowed syntax and semantics for system message traffic. The philosophy of this standard is that **devices** receive in a more flexible manner than they send.

Thus, a range of variations of syntax defined by this standard must be accepted when **devices** listen. Likewise, a precise syntax, also identified by this standard, is required when **devices** talk.

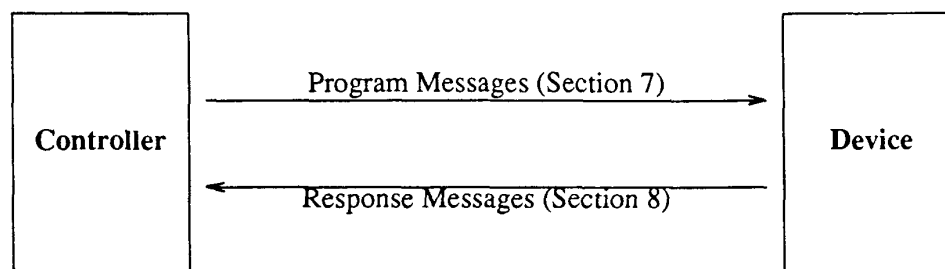


Fig 3-1
Usual Message Traffic

⁵ All IEEE 488.2 components, that is, **devices** and **controllers** in a system comprise what is defined in IEEE 488.1 as “device”.

These controller-to-device (program) and device-to-controller (response) messages are composed of syntactic elements described in Sections 7 and 8, respectively.

The following example demonstrates this relationship among program and response messages for a **device** that has a range that may be programmed and queried. The **device** has discrete ranges of 1.2, 12, and 120.

The following is an example of flexible reception of range programming with the **device** listening.

Example of Flexible Reception

Controller sends	Device Interprets:
RANGE 12.45	RANGE 12
or RANGE 12	RANGE 12
or RANGE 1.2E+1	RANGE 12

The syntactic rules of Section 7 provide for this flexibility of interpretation.

The following is an example of precise response to range status request with the **device** talking.

Example of Precise Response

Controller sends	Device Sends:
RANGE?	12

The syntactic rules of Section 8, ensure the **device** responds with a precise format.

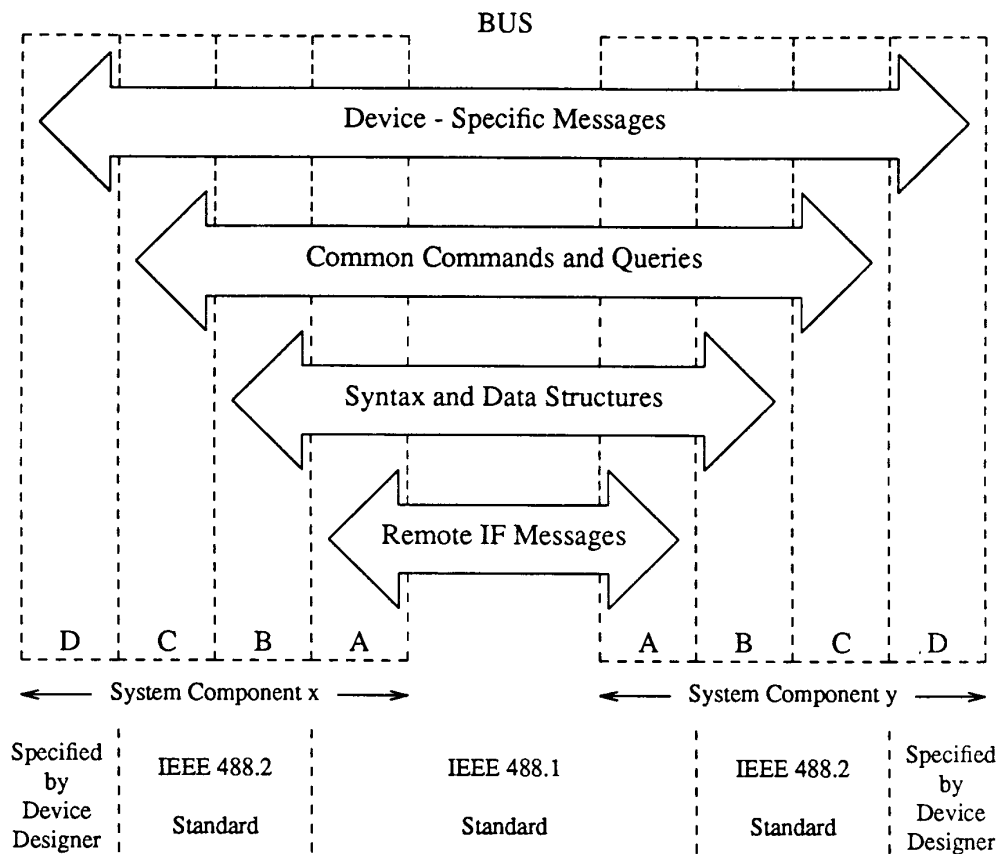
3.2.2 Other Message Traffic. In addition to controller-to-device and device-to-controller messages, protocols are defined to facilitate passing control, Section 17.

Device-to-device protocols are not explicitly defined in this document. However, message traffic between **devices** shall follow the syntax for <RESPONSE MESSAGES> defined in Section 8.

3.3 Functional Layers. The purpose of an interface system is to support the system application of the user. In the schema of which this standard is a part, the system contains successive, independent “levels” of communication protocol as shown in Fig 3-2.

Each layer shown has associated common “messages”, which form protocols to communicate between itself and the corresponding layer in the participating system components.

The protocols are designed for communication between nonpeer entities due to the requirements of IEEE 488.1. No attempt has been made to associate the protocol layering with the ISO Model for Open Systems Interconnection.



The layers have the following representations:

- Layer D represents Device Functions
- Layer C represents Common System Functions
- Layer B represents Message Communication Functions
- Layer A represents Interface Functions (IF)

When layers D, C, and B contain IEEE 488.1 device-dependent messages and layer A contains IEEE 488.1 interface messages.

Fig 3-2
IEEE 488.1 and IEEE 488.2 Functional Protocol Layers

4. Device Compliance Criteria

A **device** shall have certain capabilities. This section lists the capabilities which this standard requires in a **device**. A **device** may optionally contain additional capabilities. Any optional **device** capabilities which are described by this standard are also listed in this section.

Compliance for a **device** is divided into several areas which are considered separately. A **device** shall satisfy all the required functionality in each of the areas in order to comply with this standard.

This section summarizes requirements which are fully specified in the referenced section. The device designer shall use the requirements stated in later sections when actually designing a **device**.

4.1 IEEE 488.1 Requirements. A **device** shall contain these IEEE 488.1 subsets listed in Table 4-1 and no others.

Table 4-1
IEEE 488.1 Interface Requirements for Devices

IEEE 488.1 Interface Function	IEEE 488.1 Subsets	IEEE 488.2 Section
Source Handshake	SH1	5.1.1
Acceptor Handshake	AH1	5.1.2
Talker	T5, T6, TE5, or TE6	5.3
Listener	L3, L4, LE3, or LE4	5.4
Service Request	SR1	5.5
Remote Local	RL0 or RL1	5.6
Parallel Poll	PP0 or PP1	5.7
Device Clear	DC1	5.8
Device Trigger	DT0 or DT1	5.9
Controller	C0, or C4 with C5, C7, C9, or C11	5.10
Electrical Interface	E1 or E2	5.11

A **device** shall comply with IEEE 488.1. It shall also meet all requirements stated in Section 5 of this standard.

4.2 Message Exchange Requirements. A **device** shall follow all the requirements listed in Section 6 of this standard.

A **device's** Input Buffer may take on several forms. The Input Buffer's length may be a fixed number of bytes. It may contain a fixed number of complete <PROGRAM MESSAGE> elements. Its length in bytes or <PROGRAM MESSAGE> elements may vary with the **device's** state. See 6.1.5.

Certain query messages may generate the actual response message when the query is received. Other query messages may generate the actual response when the controller reads the response. See 6.4.5.4.

Execution of individual parsable elements may be done as they are received. The **device** may also wait until either a <PROGRAM MESSAGE UNIT SEPARATOR> or a <PROGRAM MESSAGE TERMINATOR> is parsed before executing preceding <PROGRAM MESSAGE UNIT> elements. A **device** may contain a mix of these types of commands. See 6.4.5.1.

4.3 Syntax Requirements. Sections 7 and 8 describe a set of functional elements which the device designer uses to describe the programming language for a specific **device**. The entire syntax described in Sections 7 and 8 is not required in every **device**. Some functional elements are required and some are optional.

4.3.1 Required Functional Elements. Table 4-2 lists the required functional elements.

Table 4-2
Required Functional Elements for Devices

Device Listening Functional Element	Section
<PROGRAM MESSAGE>	7.3.2
<PROGRAM MESSAGE TERMINATOR>	7.5
<PROGRAM MESSAGE UNIT>	7.3.2
<PROGRAM MESSAGE UNIT SEPARATOR>	7.4.1
<COMMAND MESSAGE UNIT>	7.3.2
<QUERY MESSAGE UNIT>	7.3.2
<COMMAND PROGRAM HEADER>*	7.6.1
<QUERY PROGRAM HEADER>*	7.6.2
<PROGRAM HEADER SEPARATOR>	7.4.3
<PROGRAM DATA SEPARATOR>	7.4.2
<PROGRAM DATA>	7.3.2
<DECIMAL NUMERIC PROGRAM DATA>	7.7.2
Device Talking Functional Element	Section
<RESPONSE MESSAGE>	8.3.2
<RESPONSE MESSAGE TERMINATOR>	8.5
<RESPONSE MESSAGE UNIT>	8.3.2
<RESPONSE MESSAGE UNIT SEPARATOR>	8.4.1
<RESPONSE DATA>	8.7
<RESPONSE DATA SEPARATOR>	8.4.2
<NR1 NUMERIC RESPONSE DATA>	8.7.2
<ARBITRARY ASCII RESPONSE DATA>	8.7.11

*NOTE: <compound command program header> and <compound query program header> are not required encoding elements.

4.3.2 Optional Functional Elements. Table 4-3 lists the optional functional elements.

Table 4-3
Optional Functional Elements for Devices

Device Listening Functional Element	Section
<COMMAND PROGRAM HEADER>*	7.6.1
<QUERY PROGRAM HEADER>*	7.6.2
<CHARACTER PROGRAM DATA>	7.7.1
<SUFFIX PROGRAM DATA>	7.7.3
<NON-DECIMAL NUMERIC PROGRAM DATA>	7.7.4
<STRING PROGRAM DATA>	7.7.5
<ARBITRARY BLOCK PROGRAM DATA>	7.7.6
<EXPRESSION PROGRAM DATA>	7.7.7
Device Talking Functional Element	Section
<RESPONSE HEADER SEPARATOR>	8.4.3
<RESPONSE HEADER>	8.6
<CHARACTER RESPONSE DATA>	8.7.1
<NR2 NUMERIC RESPONSE DATA>	8.7.3
<NR3 NUMERIC RESPONSE DATA>	8.7.4
<HEXADECIMAL NUMERIC RESPONSE DATA>	8.7.5
<OCTAL NUMERIC RESPONSE DATA>	8.7.6
<BINARY NUMERIC RESPONSE DATA>	8.7.7

Table 4-3 (Continued)
Optional Functional Elements for Devices

<STRING RESPONSE DATA>	8.7.8
<DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>	8.7.9
<INDEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>	8.7.10

*NOTE: <compound command program header> and <compound query program header> are optional encoding elements though other encoding elements are required.

4.4 Status Reporting Requirements

4.4.1 Required Status Reporting Capability. A device shall follow the status reporting model presented in Section 11. The device shall include those commands related to status reporting described in Section 10 and listed in Table 4-4.

Table 4-4
Required Status Reporting Common Commands

Status Reporting	
Command	Section
*CLS	10.3
*ESE	10.10
*ESE?	10.11
*ESR?	10.12
*SRE	10.34
*SRE?	10.35
*STB	10.36

The asterisk symbol (*) for example, preceding in the above table and throughout the text is meant to literally represent the 1st character of a common command or query (see also 7.6.1.2, 8.6.2, and Table 9-2).

A **device** shall implement the status byte register, the Service Request Enable Register, the Standard Event Status Register, and the Standard Event Status Enable Register illustrated in Fig 4-1.

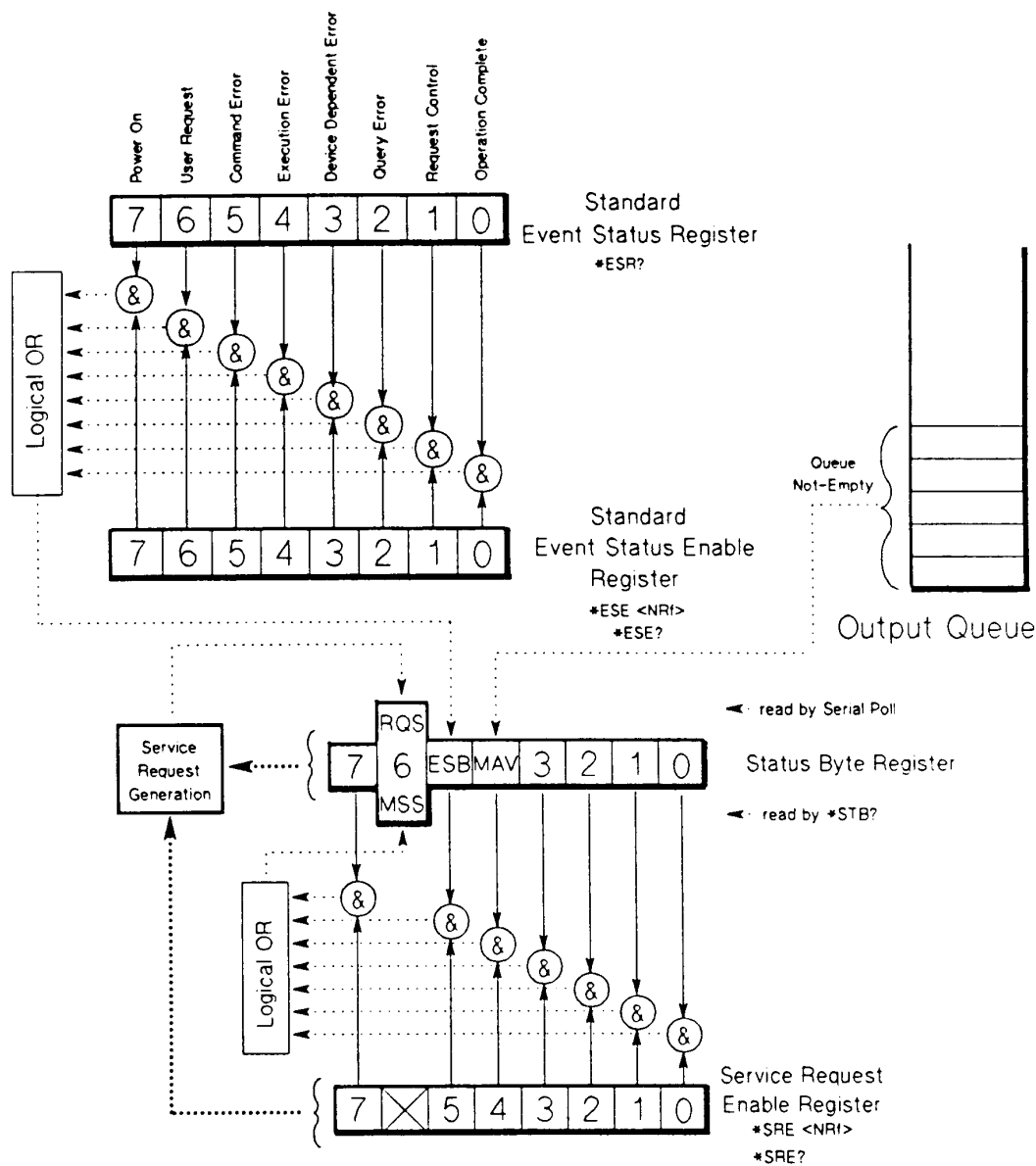


Fig 4-1
Required Status Reporting Capabilities

4.4.2 Optional Status Reporting Capability. A device may include any number of condition registers, event registers, enable registers, and queues provided they follow the model presented in Section 11.

The device may optionally include the ability to save enable registers when power is cycled. This capability also requires nonvolatile memory and all of the commands listed in Table 4-5.

Table 4-5
Optional Power-On Common Commands

Power On	
Command	Section
*PSC	10.25
*PSC?	10.26

The device may optionally include the ability to respond to a parallel poll. This capability also requires the IEEE 488.1 subset PP1 and all of the commands listed in Table 4-6.

Table 4-6
Optional Parallel Poll Common Commands

Parallel Poll	
Command	Section
*IST?	10.15
*PRE	10.23
*PRE?	10.24

4.5 Common Commands. This standard lists some reserved commands. Some of the commands are required, some are optional, and some shall be implemented in groups.

4.5.1 Required Common Commands. The common commands listed in 4.4.1 related to status reporting, and the common commands listed in 4.6 related to synchronization are required. The commands that relate to internal operations are listed in Table 4-7 and are required.

Table 4-7
Required Internal Operation Common Commands

Internal Operations	
Command	Section
*IDN?	10.14
*RST	10.32
*TST?	10.38

4.5.2 Optional Common Commands. In some cases, the implementation of a common command is independent of other common commands or device capabilities. In other cases, common commands shall be implemented in groups or in conjunction with some other device capability.

4.5.2.1 Resource Description Commands. The resource description commands, listed in Table 4-8, are optional and loosely coupled. If the resource description can be written into the device (*RDT), then the device shall also include the capability to read the resource description (*RDT?). The *RDT? query, however, may be included without the *RDT command.

Table 4-8
Optional Resource Description Common Commands

Resource Description	
Command	Section
*RDT	10.30
*RDT?	10.31

4.5.2.2 Protected User Data Commands. The protected user data commands, listed in Table 4-9, are optional. If any commands in this group are implemented, however, then all commands in this group shall be implemented.

Table 4-9
Optional Protected User Data Commands

Protected Data	
Command	Section
*PUD	10.27
*PUD?	10.28

4.5.2.3 Calibration Command. The self calibration command, listed in Table 4-10, is optional.

Table 4-10
Optional Calibration Command

Calibration	
Command	Section
*CAL?	10.2

4.5.2.4 Trigger Command. The trigger command, listed in Table 4-11, is optional. It is required, however, if the **device** has DT1 capability. If the trigger command is implemented the **device** shall also have DT1 capability.

Table 4-11
Optional Trigger Command

Trigger	
Command	Section
*TRG	10.37

4.5.2.5 Trigger Macro Commands. The trigger macro commands, listed in Table 4-12, are optional. If any commands in this group are implemented, however, then all commands in this group shall be implemented and the **device** shall have DT1 capability.

Table 4-12
Optional Trigger Macro Commands

Trigger Macro	
Command	Section
*DDT	10.4
*DDT?	10.5

If the trigger macro commands are implemented, the **device** shall also include the <ARBITRARY BLOCK PROGRAM DATA> and <DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA> functional elements.

4.5.2.6 Macro Commands. The macro commands, listed in Table 4-13, are optional. If any commands in this group are implemented, however, then all commands in this group shall be implemented.

Table 4-13
Optional Macro Commands

Macros	
Command	Section
*DMC	10.7
*EMC	10.8
*EMC?	10.9
*GMC?	10.13
*LMC?	10.16
*PMC	10.22

If the macro commands are implemented, the **device** shall also include the <STRING PROGRAM DATA>, <ARBITRARY BLOCK PROGRAM DATA>, <STRING RESPONSE DATA>, and <DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA> functional elements.

4.5.2.7 Option Identification Command. The option identification command, listed in Table 4-14, is optional.

Table 4-14
Optional Option Identification Command

Options	
Command	Section
*OPT?	10.20

4.5.2.8 Stored Setting Commands. The stored setting commands, listed in Table 4-15, are optional. If any commands in this group are implemented, however, then all commands in this group shall be implemented.

Table 4-15
Optional Stored Setting Commands

Stored Settings	
Command	Section
*RCL	10.29
*SAV	10.33

4.5.2.9 Learn Command. The learn command, listed in Table 4-16, is optional.

Table 4-16
Optional Learn Command

Learn	
Command	Section
*LRN?	10.17

4.6 Synchronization Requirements. Section 12 describes capability which is required in all **devices** for synchronization. A device designer may chose which operations have pending operation flags associated with them according to the rules in Section 12.

The required common commands related to synchronization are listed in Table 4-17.

Table 4-17
Required Synchronization Commands

Synchronization	
Command	Section
*OPC	10.18
*OPC?	10.19
*WAI	10.39

4.7 System Configuration Capability. Section 13 describes optional capability which is used to configure the IEEE 488.1 addresses of **devices**. This capability is implemented by including the common commands in Table 4-18.

Table 4-18
Optional System Configuration Commands

Auto Configure	
Command	Section
*AAD	10.1
*DLF	10.6

If either command is implemented then both commands shall be implemented.

These commands cause the **device** temporarily to suspend normal message exchange protocols and to use different protocols. To ensure compatibility with other devices, the device designer must carefully adhere to these different protocols.

4.8 Controller Capability

4.8.1 Required Controller Capability. A **device** is not required to have any controller capability.

4.8.2 Optional Controller Capability. If a **device** has any controller capability, it shall include the IEEE 488.1 C4 subset, the pass control back command listed in Table 4-19, it shall obey the passing control protocol described in 17.4, and it shall obey the requesting control protocol described in 17.5.

Table 4-19
Optional Passing Control Command

Passing Control	
Command	Section
*PCB	10.21

4.9 Device Documentation Requirements. All **devices** shall supply information to the user about how the **device** has implemented this standard. This information shall include:

- (1) A list of IEEE 488.1 Interface Functions subsets implemented, Section 5.
- (2) A description of **device** behavior when the address is set outside the range 0-30, see 5.2.
- (3) A description of when a user initiated address change is recognized by the **device**.
- (4) A description of the **device** setting at power-on, see 5.12. Any commands which modify the power-on settings shall also be included.
- (5) A description of message exchange options:
 - (a) The size and behavior of the Input Buffer, see 6.1.5
 - (b) Which queries return more than one <RESPONSE MESSAGE UNIT>, see 6.4.3
 - (c) Which queries generate a response when parsed, see 6.4.5.4
 - (d) Which queries generate a response when read, see 6.4.5.4
 - (e) Which commands are coupled, see 6.4.5.3
- (6) A list of functional elements used in constructing the device-specific commands. Whether <compound command program header> elements are used must also be included, see 7.1.1 and 7.3.3.
- (7) A description of any buffer size limitations related to block data, see 7.7.6.5.
- (8) A list of the <PROGRAM DATA> elements which may appear within an <expression> as well as the maximum sub-expression nesting depth. Any additional syntax restrictions which the **device** may place on the <expression> shall also be included.
- (9) A description of the response syntax for every query, Section 8.
- (10) A description of any device-to-device message transfer traffic which does not follow the rules for <RESPONSE MESSAGE> elements, see 8.1.
- (11) The size of any block data responses, see 8.7.9.4.
- (12) A list of common commands and queries which are implemented, Section 10.
- (13) A description of the state of the **device** after successful completion of the Calibration query, see 10.2.
- (14) The maximum length of the block used to define the trigger macro, if *DDT is implemented, see 10.4.
- (15) The maximum length of macro labels, the maximum length of the block used to define a macro, and how recursion is handled during macro expansion, if the macro commands are implemented, see 10.7.
- (16) A description of the response to the identification common query, *IDN?, see 10.14.
- (17) The size of the protected user data storage area, if the *PUD command & *PUD? are implemented, see 10.27.
- (18) The size of the resource description, if the *RDT command or *RDT? query are implemented, see 10.30 and 10.31.
- (19) A description of the states affected by *RST (see 10.32), *LRN? (see 10.17), *RCL (see 10.29), and *SAV (see 10.33).
- (20) A description of the scope of the self-test performed by the *TST? query, see 10.38.
- (21) A description of additional status data structures used in the **device's** status reporting, Section 11.
- (22) For each command, a statement describing whether it is overlapped or sequential.
- (23) For each command the device documentation shall specify the functional criteria that are met when an operation complete message is generated in response to that command, see 12.8.3.

5. Device Interface Function Requirements

This section describes the IEEE 488.1 interface function requirements of a **device**. It specifies the additional requirements of a **device** that are directly associated with the ten IEEE 488.1 interface functions. These requirements are intended to supplement the IEEE 488.1 specification for the IEEE 488.2-system environment as described in this standard.

5.1 Handshake Requirements

5.1.1 Source Handshake Requirements. A **device** shall contain the SH1 (complete capability) subset. It shall make the transition from Source Delay State (SDYS) to Source Transfer State (STRS) only if the ready for data (RFD) message is TRUE and the data accepted (DAC) message is FALSE, see 6.5.3.

5.1.2 Acceptor Handshake Requirements. A **device** shall contain the AH1 (complete capability) subset and shall additionally follow the requirement of the following paragraph.

A **device** shall enter Acceptor Idle State (AIDS) of the AH function within 1 ms after ATN assumes the FALSE state if the **device** was in Listener Idle State (LIDS) when ATN was TRUE. This requirement ensures that the **device** will operate reliably with the Find Listeners common controller protocol, see 17.6.

5.2 Address Requirements. A **device** shall have the same talk and listen addresses. The lower 5 bits of the MTA and MLA codes, the primary address of the **device**, shall be identical. (See IEEE 488.1, 6.3.) If the **device** utilizes extended addressing, the talk and listen secondary addresses (MSA's) shall also be identical.

A **device** shall have a single primary address which may be set by the user to any value within the range of 0 to 30. If the **device** utilizes extended addressing, the user shall also be able to set the single secondary address within the same range.

The address of a **device** shall be settable independent from any other device's address even if the devices share common physical resources such as the same enclosure.

A **device** shall have a local means of selecting its IEEE 488.1 primary address (and secondary address if extended addressing is used) that shall be retained during power-off. The device operator shall be capable of altering this address.

The physical configuration, labeling, and positioning of the address selection mechanism shall follow the guidelines of IEEE 488.1 Appendix I.

The device designer should avoid using address settings outside the range 0-30. Any exceptions to this recommendation shall be documented with the associated behavior.

If hard switches are used to set the address and the address is apparently set to 31, the **device** shall operate in a manner that does not disrupt **system bus** traffic.

Device documentation shall state when a user initiated address change is recognized by the **device**. A **device** shall update its address at power-on.

5.3 Talker Requirements. A **device** shall contain either the T5, T6, TE5, or TE6 subsets. These subsets require the basic talker with serial poll and unaddress if MLA.

This standard assumes that the controller performs all talker addressing via MTA and unaddressing via MLA, OTA, and UNT. Talk-only mode capability is, therefore, not a requirement of this specification. The presence of a talk-only mode of operation (for example, in a controller-less system) is allowable, but operation in this mode is considered beyond the scope of this standard.

5.4 Listener Requirements. A **device** shall contain either the L3, L4, LE3, or LE4 subsets. These subsets require the basic listener with unaddress if MTA.

This standard assumes the controller performs all listener addressing via MLA and unaddressing via MTA and UNL. Listen-only mode capability is, therefore, not a requirement of this specification. The presence of a listen-only mode of operation (for example, in a controller-less system) is allowable, but operation in this mode is considered beyond the scope of this standard.

Note the additional requirement for inactivating the Acceptor Handshake function on an ATN FALSE transition in Listener Idle State (LIDS). See 5.1.2.

5.5 Service Request Requirements. A device shall contain the SR1 (complete capability) subset and shall conform to the status handling requirements of Section 11.

5.6 Remote/Local Requirements

5.6.1 Control and Operation Definitions. IEEE 488.2 control functions and operations are classified using the source of control and method of annunciation.

5.6.1.1 IEEE 488.2 Remote Operation. An IEEE 488.2 remote operation is any operation of a device function in a **system** that is effected via program messages over the **system interface**.

5.6.1.2 IEEE 488.2 Local Operation. An IEEE 488.2 local operation is any operation of a device function that is not an IEEE 488.2 remote operation.

An example is the actuation of a user-accessible switch, knob, button, touch-screen location, etc that is, physically attached to a **device** and used to locally control or program the **device**.

Operation of a **device** via any other bus or interface connected to the **device** is considered to be local operation from the standpoint of the **system**.

5.6.1.3 Local Control. A local control effects IEEE 488.2 local operation of a **device**.

Local controls include **device** inputs which are designed for control.

Local controls and the accompanying Remote/Local requirements of this section are intended to apply only to local controls whose function can also be effected via IEEE 488.2 remote operation.

Local controls whose function cannot be effected via IEEE 488.2 remote operation are beyond the scope of this standard.

The following control functions are explicitly excluded from categorization as local controls.

- (1) Switching line power.
- (2) The generation of the user request (URQ) message.

5.6.1.4 External-Control-Signal. An external-control-signal invokes **device** actions using a device port other than the **system interface**. External-control-signals are beyond the scope of this standard, but may optionally function as a local control. External-control-signals, however, shall not violate the message exchange protocols of Section 6.

An example of an external-control-signal is a digital voltmeter's external trigger which initiates a measurement. The results may then be read from the **device**.

5.6.1.5 Hard Local Control. Hard local controls on a **device** have indicators (mechanical, positional, etc) which cannot be changed by IEEE 488.2 remote operation of the **device**.

For example, an instrument may have a mechanical rotary switch that is used to select among three different modes of operation. The state of this switch is indicated by printed labels on the front panel of the **device**, but remote messages cannot rotate the switch.

5.6.1.6 Soft Local Control. Any local control that is not a hard local control is a soft local control.

For example, a physical (momentary contact) key on a **device** may be used to turn a device function on or off with alternate pushes. The on/off state is indicated by a light located by the switch. The light is controlled by the actual state of the **device** function.

5.6.1.7 Programmable Local Control. A local control of a **device** function that can be affected by IEEE 488.2 remote operation as well as by IEEE 488.2 local operation is a programmable local control. Programmable local controls may be hard or soft local controls.

5.6.2 IEEE 488.1 Subset Requirements. A **device** that is capable of IEEE 488.2 local and remote operation and utilizes programmable local controls shall contain the RL1 (complete capability including local lockout) subset of the IEEE 488.1 Remote/Local function.

A **device** that is incapable of IEEE 488.2 local operation or does not utilize programmable local controls may contain the RL0 (no capability) subset of the IEEE 488.1 Remote/Local function. A **device** containing the RL0 subset shall ignore all IEEE 488.1 interface messages related to Remote/Local function state changes.

5.6.3 Local-to-Remote State Transition Requirements. The transition from Local State (LOCS) or Local with Lockout State (LWLS) to Remote State (REMS) or Remote with Lockout State (RWLS) shall disable local operation of all programmable local controls.

5.6.4 Remote-to-Local State Transition Requirements. The transition from REMS or RWLS to LOCS or LWLS shall enable local operation of all programmable local controls of the **device**.

The transition shall also alter any hard local control functions so they match the hard local control indicator. After the transition, the **device's** front panel and state of control shall agree.

The transition shall not alter any soft local control functions.

Setting the rtl local message TRUE with LLO false or Accept Data State (ACDS) inactive shall cause a **device** in REMS to change to LOCS (see IEEE 488.1, 2.8.3.3).

5.6.5 Local State Operation. In LOCS or LWLS, a **device** shall process all program messages so as to maintain consistency between the information presented to the local user and the actual state of the **device**.

To avoid an inconsistency, a device shall do one of the following:

- (1) Correct the inconsistent information presented to the local user
- (2) Remove the inconsistent information
- (3) Generate an execution error and discard the responsible <PROGRAM MESSAGE UNIT>

(See IEEE 488.2 Status reporting protocols in Section 11.)

Alternative 1 is the preferred choice.

In LOCS or LWLS a **device** shall not be inhibited from sending a <RESPONSE MESSAGE> in response to a query previously received by the **device**.

When a **device** is sent program messages while in local there are potential arbitration problems if front panel controls are manipulated. Application programs can avoid such problems by placing the device in remote.

5.6.6 Remote State Operation. In REMS or RWLS, a **device** shall process all program messages.

In REMS or RWLS, device-specific messages (not defined by this standard) may be sent to selectively enable specific local controls. Under this circumstance, the specified local controls are no longer locally inoperative. If specific hard local controls are enabled, the **device's** state of control and these hard local controls shall agree.

In REMS or RWLS a **device** shall not be inhibited from sending a <RESPONSE MESSAGE> in response to a query previously received by the **device**.

5.6.7 Operation Independent of Remote/Local State. The Remote/Local function shall be associated only with the enabling and disabling of local controls as defined in this section. Other device operation, such as updating the device state and associated local annunciation shall function independent of the Remote/Local state of the **device**.

5.6.8 Remote/Local Indicator Requirements. **Devices** with hard local controls shall incorporate a "Remote" indicator (IEEE 488.1, Appendix H).

For **devices** without hard local controls, the Remote indicator is optional. All indicators relating to Remote/Local transitions shall be labeled and function in accordance with the requirements of IEEE 488.1, Appendix H.

5.7 Parallel Poll Requirements. A **device** shall have either the PP0 (no capability) or the PP1 (remote configuration) subset. A **device** with the PP1 subset shall conform to the requirements of 11.6.

5.8 Device Clear Requirements. A **device** shall contain the DC1 (complete capability including selective device clear) subset. Specific IEEE 488.2 requirements for the **device's** response to entering DCAS are specified in Section 6 and summarized in the following:

A **device** entering Device Clear Active State (DCAS) via a DCL or SDC command
Shall:

- (1) Clear the Input Buffer and Output Queue.
- (2) Reset the Parser, Execution Control, and Response Formatter (see 6.1.4.2.6).
- (3) Clear any command that prevents processing a *RST or other device commands.
- (4) Discard all commands and queries deferred due to coupled parameters (see 6.4.5.3).
- (5) Put the **device** into Operation Complete Command Idle State (OCIS) and into Operation Complete Query Idle State (OQIS) (see 12.5.2.1.1 and 12.5.3.1.1).
- (6) Abort the *AAD and *DLF common commands (see Section 13).
- (7) Put the **device** in the message exchange IDLE state (see 6.3.1.1)

Shall not:

- (1) Change any settings or stored data in the **device** except as stated earlier.
- (2) Interrupt front panel I/O.
- (3) Interrupt or affect any **device** operation in progress except as stated earlier.
- (4) Change the status byte other than clearing the MAV bit as a consequence of clearing the Output Queue, (see 11.2.1.2).

5.9 Device Trigger Requirements. A **device** may contain either the DT0 (no capability) or the DT1 (complete capability) IEEE 488.1 device trigger function. A **device** containing the DT1 subset shall implement the associated common command, *TRG as defined in 10.37. Use of the optional *DDT common command, 10.4, affects the operation of *TRG and Group Execute Trigger (GET).

5.10 Controller Function Requirements. A **device** shall contain either no Controller function capability (C0) or contain a Controller function with the following subset options: C4 with C5, C7, C9, or C11. These subsets provide the capability to respond to SRQ, send IF messages, receive control, pass control, and take control synchronously. They specifically prohibit system controller capability, that is, sending the IFC and REN interface messages. They optionally permit the capability of passing control to self and conducting a parallel poll.

If a **device** contains any Controller function subset other than C0, then it shall also be able to pass and receive control by means of the protocols described in 17.4 and 17.5.

See Section 15 for **controller** (as contrasted to **device**) requirements.

5.11 Electrical Requirements. Implementation of the E2 electrical interface option (IEEE 488.1, Appendix C2) is recommended. Open collector drivers shall be used to drive the SRQ, NRFD, and NDAC signal lines. Three-state drivers should be used to drive the DAV, EOI, and ATN signal lines. A **device** is not allowed to drive REN or IFC. If the **device** is in Parallel Poll Active State (PPAS), the **device** shall use open collector drivers to drive the DIO1-8 signal lines. When not in PPAS, the **device** should use three-state drivers to drive the DIO1-8 signal lines. If a **device** needs to source handshake data bytes at a rate greater than 250 000 bytes per second, the E2 option is required, (IEEE 488.1, 5.2).

NOTE: Some integrated circuits may require external hardware to avoid driving the DIO lines during a parallel poll, even though the PP0 subset is implemented.

5.12 Power-On Requirements. At power-on, **device** settings shall be either restored to their values when the **device** was last powered off, set to known states which are explicitly stated in the **device** documentation, or set to a state defined by the user and stored in local nonvolatile memory.

Device-specific commands may be provided to select whether device settings are restored or set to a known state at power-on. This standard defines one such command, power-on status clear, (*PSC, 10.25) which sets and clears the power-on-status-clear flag.

5.12.1 Items Not Affected by Power-On. A **device** shall not alter the following due solely to a power-on (receipt of the IEEE 488.1 pon local message).

- (1) The bus address.
- (2) Pertinent calibration data.
- (3) Data or device states which cause a change in the response to the following common queries:
 - *IDN? (Identification Query, see 10.14)
 - *OPT? (Option Identification Query, see 10.20)
 - *PSC? (Power-on Status Clear Query, see 10.26)
 - *PUD? (Protected User Data Query, see 10.28)
 - *RDT? (Resource Description Transfer Query, see 10.31)

5.12.2 Items Dependent Upon Power-On-Status-Clear Flag. If the power-on-status-clear flag is FALSE, the Service Request Enable Register, the Standard Event Status Enable Register, and the Parallel Poll Enable Register shall not be affected by power-on. See *PSC command, refer to 10.25.

A **device** shall clear the Service Request Enable Register, the Standard Event Status Enable Register, and the Parallel Poll Enable Register at power-on if the power-on-status-clear flag is TRUE or if the *PSC command is not implemented.

5.12.3 Items That May be Affected by Power-On. At the discretion of the device designer, a **device** may alter the following at power-on.

- (1) Current device function states
- (2) Status information
- (3) *SAV/*RCL registers, see 10.33 and 10.29
- (4) Macro definition defined with the *DDT command, see 10.4
- (5) Macro definitions defined with the *DMC command, see 10.7
- (6) Macro enabling with the *EMC command, see 10.8
- (7) Address received with last *PCB command, see 10.21

6. Message Exchange Control Protocol

The device message exchange protocol specifies how a **device** handles program and response messages.

The term **controller** refers to the system element that is exchanging messages with the **device**. The requirements of this section also apply to device-to-device communication.

The phrase "... the **device** shall (not) ..." defines a requirement of this standard. The phrase "... the **controller** should (not) ..." refers to conditions which will (not) happen under normal conditions. The **device's** response to both normal and exceptional conditions is defined by this standard.

The term "Query Message", as used in this section, is a <PROGRAM MESSAGE> which contains one or more <QUERY MESSAGE UNIT> elements. Note that all bracketed syntactic elements are defined in Sections 7 and 8.

The terms "Command Error", "Execution Error", "Query Error", and "Standard Event Status Register" are defined in Section 11.

6.1 Functional Elements. Figure 6-1 presents the relationship among the IEEE 488.1 bus, the Message Exchange Interface described in this section, the Status Reporting functions described in Section 11, and the device-specific functions provided by the device designer.

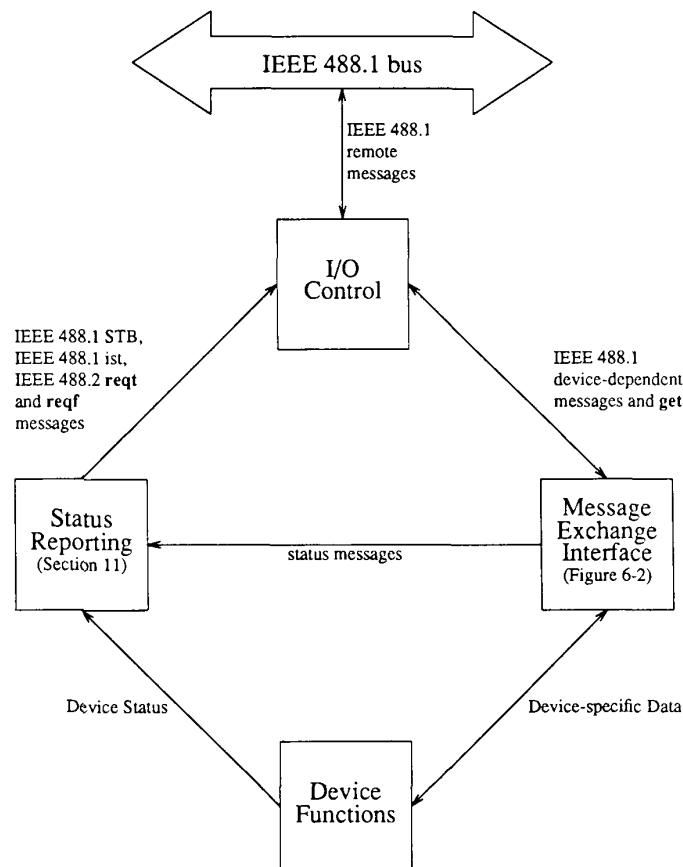


Fig 6-1
Device Status and Message Exchange Overview

6.1.1 IEEE 488.1 Bus. The IEEE 488.1 Bus, shown in both Figs 6-1 and 6-2, represents the physical IEEE 488.1 interface cable and drivers as defined in IEEE 488.1.

6.1.2 Status Reporting. The Status Reporting block appears only in Fig 6-1. Figure 6-2 does, however, display the status messages sent from the Message Exchange Interface and the messages sent to the I/O Control.

The Status Reporting block receives error and status messages from the Message Exchange Interface and Device Functions blocks. It also sends the STB, **reqt**, **reqf**, and IEEE 488.1 ist messages to the I/O Control, so that it can respond to IEEE 488.1 Serial Poll and Parallel Poll requests. The Status Reporting block is completely described in Section 11.

6.1.3 Message Exchange Interface. The Message Exchange Interface deals with all IEEE 488.1 device-dependent messages to and from the **device**.

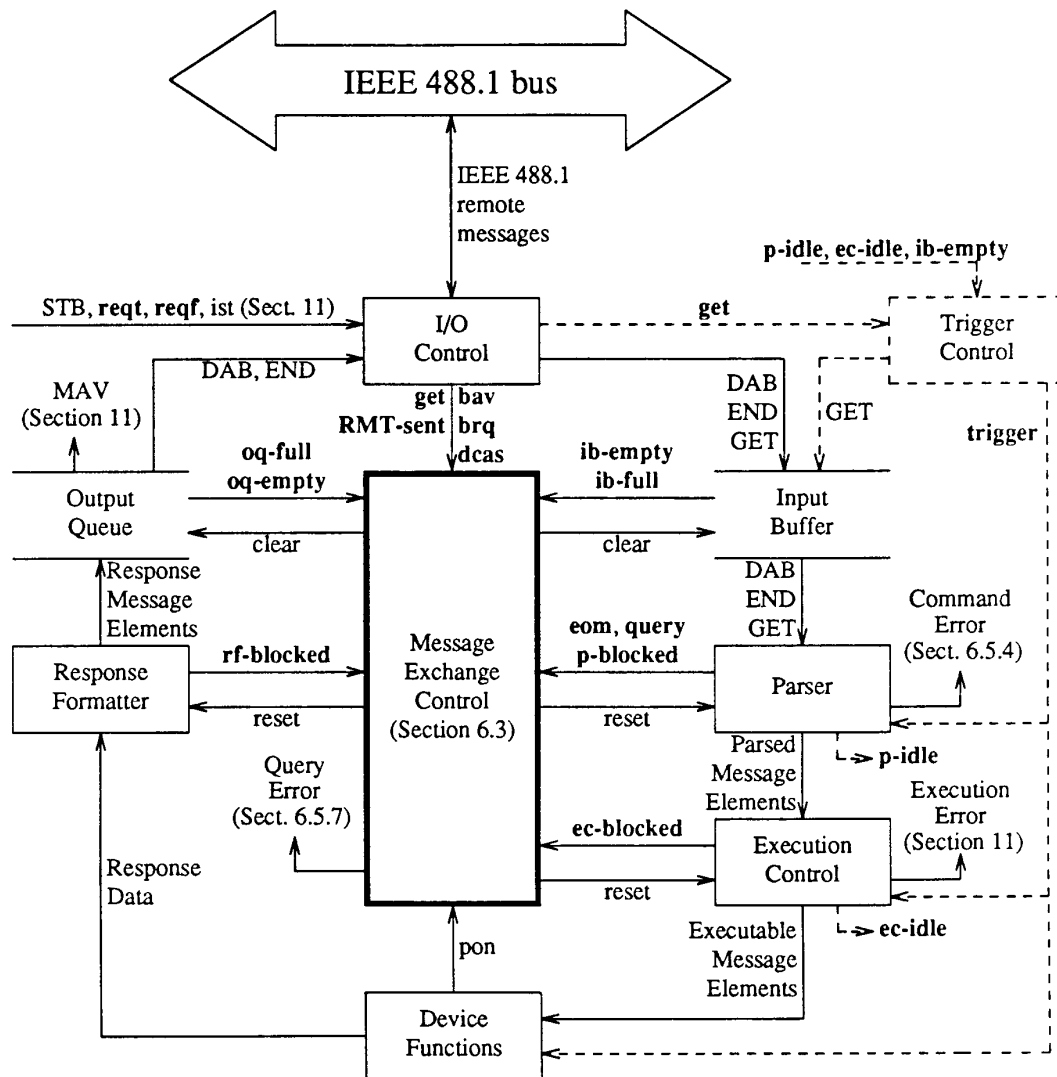


Fig 6-2
Message Exchange Control Interface Functional Blocks

Figure 6-2 presents a logical model which describes the operation of the **device's** Message Exchange Interface. It is not intended to imply a particular physical implementation. Some handshaking signals between blocks have been omitted to reduce the complexity of the diagram.

6.1.4 I/O Control. The I/O Control, shown in Figs 6-1 and 6-2, interprets and decodes IEEE 488.1 remote messages. Many of the functions of this block are provided by commercial integrated circuits that have been designed to implement the requirements of IEEE 488.1.

The I/O Control responds to IEEE 488.1 Serial Poll and Parallel Poll requests as directed by the STB, **reqt**, **reqf**, and **ist** messages from the Status Reporting block, Section 11. It generates the **dcas**, DAB, END, **get**, **bav**, and **brq** messages which are sent to the Message Exchange Interface, and receives the DAB and END messages from the Message Exchange Interface.

6.1.4.1 I/O Control Rules

6.1.4.1.1 I/O Control Sending Data Bytes. When **brq** is TRUE and **oq-empty** is FALSE, the I/O Control shall remove one byte from the Output Queue, set **brq** FALSE, and send the byte to the **controller** using the IEEE 488.1 Source Handshake (SH) function. See 6.1.4.2.3.

6.1.4.1.2 I/O Control Receiving Data Bytes. When **bav** is TRUE and **ib-full** is FALSE, the I/O Control shall set **bav** FALSE and then place the associated DAB and any accompanying END message into the Input Buffer using the IEEE 488.1 Acceptor Handshake (AH) function. See 6.1.4.2.4.

6.1.4.1.3 I/O Control Receiving IEEE 488.1 GET Message. When **get** is TRUE and **ib-full** is FALSE, the I/O Control shall set **get** FALSE and then place a GET message in the Input Buffer.

NOTE: If the optional Trigger Control is implemented, the **get** message shall be sent to the Trigger Control rather than the Input buffer. The **get** can be executed directly under certain conditions, see 6.1.1.1.

6.1.4.1.4 I/O Control Ordering of Received Messages. The I/O control shall preserve the order of received messages, including data bytes (DAB's), END messages, Group Execute Trigger, Device Clear, and Selected Device Clear.

The following example shows a design technique for preserving the order of received data bytes and the IEEE 488.1 remote interface messages: DCL, SDC, and GET.

Design the **device's** message interface for Device Trigger handling as follows:

Design the IEEE 488.1 AH function to wait in Accept Data State (ACDS) when the Device Trigger (DT) function enters the Device Trigger Active State (DTAS) after a GET is received and until the I/O Control sets the **get** message TRUE. This action is often called "NDAC bus holdoff".

Have the I/O Control set **bav** FALSE and place the data byte in the Input Buffer before accepting the GET message when the **bav** and **get** messages are both TRUE. Since the new data byte associated with the **bav** message must have been received first with the bus being in a holdoff state during the receipt of the GET message, proper sequencing is guaranteed.

Ensure the I/O Control accepts the GET message (sets **get** TRUE), before it releases the bus holdoff and allows the **device's** IEEE 488.1 AH function to accept a new data byte.

Design the **device's** message interface for Device Clear handling as follows:

Design the IEEE 488.1 AH function to wait in ACDS when the Device Clear (DC) function enters the Device Clear Active State (DCAS) until the I/O Control sets the **dcas** message TRUE.

Assure the I/O Control sets **bav** FALSE and discards the DAB before it releases the bus holdoff and allows the **device's** AH function to accept a new data byte. Since the new data byte associated with the **bav** message must have been received first with the bus being in a holdoff state during the receipt of the DCL or SDC messages, proper sequencing is guaranteed.

6.1.4.2 I/O Control Messages. This section describes the action of the I/O control IEEE 488.2 local messages.

6.1.4.2.1 Data Byte Message (DAB). The DAB message is defined in IEEE 488.1. It consists of eight bits that are either sent or received by the **device** while ATN is FALSE.

6.1.4.2.2 End Message (END). The END message is defined in IEEE 488.1. It is sent with a DAB by setting EOI TRUE and ATN FALSE.

6.1.4.2.3 Byte Requested Message (brq). The **brq** message indicates the readiness of the I/O control to send a data byte to the **controller**.

The I/O Control shall set **brq** TRUE after the IEEE 488.1 SH function enters the IEEE 488.1 Source Generate state (SGNS) and the T or TE function is in the Talker Active State (TACS). The **brq** message shall be set TRUE once and only once for each transition into SGNS.

The I/O Control shall set **brq** FALSE when any of the following conditions occur:

- (1) The I/O Control sends a data byte from the Output Queue to the **controller** (see 6.1.4.1.1).
- (2) The **device** performs the INTERRUPTED action described in 6.3.2.3.
- (3) The **device** performs the UNTERMINATED action described in 6.3.2.2.
- (4) The I/O Control sets the **dcas** message TRUE.
- (5) The Message Exchange Control enters the IDLE state from the DONE or DEADLOCK states.

No other device or interface condition shall cause **brq** to be set FALSE.

6.1.4.2.4 Byte Available Message (bav). The **bav** message indicates that the I/O Control has received a data byte and is ready to place it in the Input Buffer.

The I/O Control shall set **bav** TRUE after the IEEE 488.1 L or LE function is in the Listener Active State (LACS) and the IEEE 488.1 Acceptor Handshake function enters the Accept Data State (ACDS). The **bav** message shall be set TRUE once and only once for each transition from the Acceptor Ready State (ACRS) to ACDS.

The I/O Control shall set **bav** FALSE when any of the following conditions occur:

- (1) A data byte is placed in the Input Buffer. (If the **device** has a zero length Input Buffer, the I/O Control shall set **bav** FALSE when the data byte is sent to the Parser.) See 6.1.4.1.2.
- (2) The **dcas** message is set TRUE. The associated data byte shall be discarded.

No other device or interface condition shall cause **bav** to be set FALSE. See 6.1.4.1.3.

6.1.4.2.5 Group Execute Trigger Message (get). The IEEE 488.2 **get** message signals the transition of the IEEE 488.1 Device Trigger (DT) function to the Device Trigger Active State (DTAS). This transition occurs when a IEEE 488.1 Group Execute Trigger (GET) remote interface message is received while the **device** was addressed to listen. The **get** message can be used to initiate a device-defined action or the action defined by the *DDT common command, see 10.4.

The **device** shall process data bytes and GET messages in the order received, see 6.1.4.1.4.

The standard specifies two methods for device triggering, the common command *TRG, see 10.37, and the IEEE 488.1 interface message GET. Typically, device triggering is used for two distinct purposes:

- (1) For the properly sequenced initiation of a device-defined or user-defined action within a single addressed device
- (2) For the synchronized initiation of device-defined or user-defined actions within multiple addressed devices

For purpose 1, either GET or *TRG may be used for device triggering. The storage of either one in the Input Buffer ensures that the trigger will be executed in the correct sequence in relation to other commands.

For purpose 2, where the required synchronization could be compromised by the software/firmware processing overhead, the **get** message may be used in conjunction with the optional Trigger Control Block. **get** will generate a **trigger** local message for immediate execution by the **device** provided that the Input Buffer is empty and that the Parser and Execution Control are idle. If the Input Buffer is not empty, the GET message must be placed in the Input Buffer, and execution of the trigger deferred until previous commands have been executed.

If the optional Trigger Control block (see 6.1.11) is implemented, the I/O Control shall send the **get** message to it. Otherwise, the I/O Control shall place the GET message directly in the Input Buffer.

6.1.4.2.6 Device Clear Active State Message (dcas). The I/O Control shall set the **dcas** message TRUE after the **device** enters the Device Clear Active State (DCAS) defined by IEEE 488.1. The **device** enters DCAS whenever it receives the Device Clear (DCL) remote command or it receives the Selected Device Clear (SDC) remote command while it is addressed to listen.

When **dcas** is set TRUE, the Input Buffer and Output Queue shall be cleared, the Parser, Execution Control, and Response Formatter shall be reset, and the **device** shall enter the Message Exchange IDLE state. The **device** also enters Operation Complete Command Idle State (OCIS) and Operation Complete Query Idle State (OQIS), 12.5, and returns to normal operation after the *AAD or *DLF commands, Section 13.

The I/O Control shall set the **dcas** message FALSE when these actions are completed and the Message Exchange Control enters the IDLE state. Device Clear requirements are summarized in 5.8.

The **device** shall ensure that data bytes and DCL or SDC are processed in the order received. If the **device** receives a new data byte while **dcas** is TRUE, it shall not discard the data byte or set **bav** FALSE.

6.1.4.2.7 Response Message Terminator Sent Message (RMT-sent). The I/O Control block shall set the **RMT-sent** message TRUE when it has sent the <RESPONSE MESSAGE TERMINATOR>.

Receipt of TRUE **bav** or **brq** messages shall cause the I/O Control to set the **RMT-sent** message FALSE.

6.1.5 Input Buffer. The Input Buffer, shown in Fig 6-2, stores DAB's, GET messages, and END messages. The Input Buffer then delivers these messages to the Parser in the order they were received from the I/O Control. The **dcas** message shall not be stored in the Input Buffer as this could result in the **device** not being able to immediately respond to **dcas**.

The Input Buffer shall be implemented as a first-in first-out data structure. Data bytes, END, and GET messages are placed into the Input Buffer by the I/O Control as the **device** handshakes them from the IEEE 488.1 bus. The details of when and how items are placed into the Input Buffer are described in 6.1.4.1.2, 6.1.4.1.3, and 6.1.4.1.4 on I/O Control Receiving. DAB's, END, and GET messages are removed from the Input Buffer by the Parser at a rate consistent with the capabilities of the Parser and the associated execution of commands.

6.1.5.1 Input Buffer Rules

6.1.5.1.1 Input Buffer Length. The Input Buffer shall be equal to or greater than zero in length. The Input Buffer's length may be fixed at some number of bytes or program messages, or its length may vary. The Input Buffer length shall be specified in the user documentation.

The operation of the Input Buffer is transparent to application programs, except as a performance improvement. **Devices** which can accept program messages faster than they can execute them should have Input Buffers long enough to allow storing a reasonably sized program message. This buffering will allow application programs to send messages to the **device** and then continue to use the **system bus** for other purposes while the **device** is taking the time it needs to respond.

For example, a digital plotter responds to commands slowly due to the need to allow for pen movement. If a typical length program message for the **device** is 80 bytes, the **device** should have an Input Buffer of at least 80 bytes.

The device designer must consider the application of the **device** when choosing an Input Buffer length. A larger Input Buffer is useful in smart, highly independent test instruments which use complex <PROGRAM MESSAGE> elements. These complex <PROGRAM MESSAGE> elements may require a significant time to process. In some systems, this time can be used for tasks other than waiting for this **device** to finish processing a <PROGRAM MESSAGE>. Increasing buffer length, however, requires the user to be more careful about synchronization within the system.

If a zero length Input Buffer is used, the **system interface** provides a source of synchronization. With this very short Input Buffer, the **device** can never fall behind the **controller**. If a DAB or GET is sent it will be processed immediately by the **device**.

NOTE: Most commercial integrated circuits which implement the requirements of IEEE 488.1 provide at least a one-byte internal data buffer.

6.1.5.1.2 Input Buffer Overflow. The Input Buffer shall not overflow. If the buffer is full, the device shall not enter the IEEE 488.1 Acceptor Ready State (ACRS) while in Listener Active State (LACS) (NRFD holdoff). After the Parser removes an item from the Input Buffer, the I/O control may accept another item from the **system interface**. No error shall be reported when the Input Buffer fills unless the **device** becomes deadlocked, see 6.3.1.7 and 6.5.7.4.

6.1.5.1.3 Input Buffer Clearing. The Input Buffer shall be cleared when **pon** or **dcas** is TRUE. No device or interface condition is allowed to keep the clear operation from taking place. Except for the emptying during normal message processing and the actions of **pon** and **dcas**, no other interface or device condition shall be allowed to clear the Input Buffer. The Input Buffer is cleared to ensure that the **device** will be ready to receive and execute new <PROGRAM MESSAGE> elements following **dcas**.

6.1.5.2 Input Buffer Messages

6.1.5.2.1 Input Buffer Empty Message (ib-empty). The Input Buffer Empty message, **ib-empty**, shall be TRUE when the Input Buffer is empty. The **ib-empty** message shall be FALSE at all other times.

6.1.5.2.2 Input Buffer Full Message (ib-full). The Input Buffer Full message, **ib-full**, shall be TRUE when the Input Buffer is full. The **ib-full** message shall be FALSE at all other times.

6.1.6 Parser. The Parser, shown in Fig 6-2, is the logical portion of a **device** which takes DAB's, END messages, and GET messages from the Input Buffer and analyzes them by separating out the various IEEE 488.2 syntactic elements. It reports invalid syntax or headers to the Status Reporting block as

Command Errors. The Parser converts syntactic elements into an internal representation which is sent to the Execution Control. The Parser also generates the **eom** and **query** messages when it recognizes these syntactic elements.

A <PROGRAM MESSAGE> or <PROGRAM MESSAGE UNIT> is considered “parsed” when it has been analyzed by the Parser and the Parser is ready to continue analyzing other <PROGRAM MESSAGE UNIT> elements.

6.1.6.1 Parser Rules

6.1.6.1.1 Parser Errors. The Parser shall detect Command Errors and report them to the Status Reporting block by setting the Command Error bit in the Standard Event Status Register, see 11.5.1.1.4.

The Parser shall report a Command Error if it finds a syntax error in a <PROGRAM MESSAGE>, finds an unrecognized header, finds a parameter (<PROGRAM DATA> element) that is of the wrong type for its associated header, or encounters a GET message between the first byte of a <PROGRAM MESSAGE> and the <PROGRAM MESSAGE TERMINATOR>.

When a Command Error is detected the **device** shall determine what will be done with any prior parsable elements of the same <PROGRAM MESSAGE>. The **device** is allowed to either discard or attempt to execute any such parsable elements.

When a Command Error occurs the Parser shall discard all subsequent DAB's and GET messages until either:

- (1) **dcas** is TRUE, or
- (2) **pon** is TRUE, or
- (3) **eom** is TRUE, or
- (4) **brq** is TRUE and **ib-empty** is TRUE

The device designer may also choose to stop discarding DAB's and GET messages for other device defined conditions such as receipt of a comma, semicolon, or NL. The Parser is reset when any of these conditions is met.

NOTE: The **device** does not empty the Input Buffer after a Command Error. The Parser discards DABs and GET messages until it has detected one of the above conditions. The **device** will then resume normal parsing and executing of messages.

6.1.6.1.2 Parser Resetting. When the Parser is reset it shall interpret the next data byte it receives as the first byte of a <PROGRAM MESSAGE>. The Parser shall be reset when **dcas** or **pon** is TRUE. No device or interface condition is allowed to keep the reset operation from taking place. Other conditions that can reset the Parser are described in 6.1.6.1.1.

6.1.6.2 Parser Messages

6.1.6.2.1 Parsed Message Element. A Parsed Message Element represents the **device's** internal representation of all or part of a <PROGRAM MESSAGE>.

6.1.6.2.2 End Of Message Detected Message (eom). The Parser shall set the End Of Message Detected message, **eom**, TRUE when any of the following conditions occurs:

- (1) When it receives an END message or a sequence of data bytes making up a <PROGRAM MESSAGE TERMINATOR>, see 7.5, from the Input Buffer, or
 - (2) After it parses a GET message, except when the GET occurs between the first byte of a <PROGRAM MESSAGE> and the <PROGRAM MESSAGE TERMINATOR>, see 6.1.6.1.1, or
 - (3) When the Trigger Control block has passed the **trigger** message to the Device Functions block.
- The Parser shall set the **eom** message FALSE when any of the following conditions occurs:

- (1) When it is reset, or
- (2) When it receives any other DAB or GET message from the Input Buffer.

When the Parser has received a GET message from the Input Buffer or the Trigger Control block has passed a **trigger** message to the Device Functions block, any associated **query** message shall be set TRUE prior to setting the **eom** message TRUE. This ensures a proper sequence of execution of Message Exchange block transitions (Fig 6-4).

6.1.6.2.3 Query Detected Message (query). The Parser shall set the Query Detected message, **query**, TRUE when it receives any of the following:

- (1) A sequence of data bytes making up a valid <QUERY PROGRAM HEADER> which is not a macro label, or
- (2) A valid <COMMAND PROGRAM HEADER> or <QUERY PROGRAM HEADER> which is a macro label, where the macro is defined to include a <QUERY MESSAGE UNIT>, or

(3) A GET message or *TRG command (or, if implemented, the **trigger message**), where the device implements the *DDT common command and the device trigger action defined by the *DDT command includes a <QUERY MESSAGE UNIT>, or

(4) A GET message or *TRG command (or, if implemented, the **trigger message**), where the device does not implement the *DDT common command and the device trigger action specified by the device designer is to generate a <RESPONSE MESSAGE>.

“Valid <QUERY PROGRAM HEADER>” means that all characters up to and including the “?” represent a valid query mnemonic or macro label, and the characters immediately following the “?” form a legal separator or terminator.

“Valid <COMMAND PROGRAM HEADER>” means that the header represents a valid command mnemonic or macro label, and the characters immediately following the mnemonic form a legal separator or terminator.

For example,

*XYZ? is not a valid <QUERY PROGRAM HEADER> because there is no “*XYZ?” common query

*ESR?X is not a valid <QUERY PROGRAM HEADER> because “X” is not a separator or terminator

*IDN? 42 is a valid <QUERY PROGRAM HEADER> although the argument is incorrect.

A Command Error will be generated for all of the three conditions listed earlier.

The Parser shall set **query** FALSE when it is reset or when it receives any other DAB or GET message from the Input Buffer.

6.1.6.2.4 Parser Idle Message (p-idle). The Parser shall set the Parser Idle message, **p-idle**, TRUE when the Parser has parsed all prior messages and is not actively processing a valid <COMMAND PROGRAM MESSAGE> or a valid <QUERY PROGRAM MESSAGE>. The Parser shall set **p-idle** FALSE at all other times.

6.1.6.2.5 Parser Blocked message (p-blocked). The Parser shall set the Parser Blocked message, **p-blocked**, TRUE when the Parser is waiting for the Execution Control to finish processing the previous Parsed Message Element. The Parser shall set **p-blocked** FALSE at all other times.

6.1.7 Execution Control. The Execution Control block, shown in Fig 6-2, deals with coupled parameters (see 6.4.5.3), queries which require a device action before they can generate a <RESPONSE MESSAGE>, and the **device** synchronization commands described in Section 12. The Execution Control determines when the **device** has enough information to execute a command. When the Execution Control has gathered enough information to initiate a **device** action, it sends an Executable Message Element to the Device Functions block. The Execution Control may wait for the resulting **device** actions to complete (Sequential Command) or may continue processing commands (Overlapped Command), depending on the command. See 12.2. A <PROGRAM MESSAGE UNIT> is considered “executed” when it has been parsed and all corresponding device operations have been either completed (Sequential Command) or initiated (Overlapped Command).

6.1.7.1 Execution Control Rules

6.1.7.1.1 Execution Control Errors. The Execution Control shall detect Execution Errors and report them to the Status Reporting block, see 11.5.1.1.5.

6.1.7.1.2 Execution Control Resetting. When the Execution Control is reset prior history will not affect the execution of new <PROGRAM MESSAGE> elements received after **dcas** or **pon**. The Execution Control shall be reset when **dcas** or **pon** is TRUE. No **device** or interface condition is allowed to keep the reset operation from taking place. No other interface or **device** condition shall reset the Execution Control.

6.1.7.2 Execution Control Messages

6.1.7.2.1 Executable Message Element. An Executable Message Element is a directive to the Device Functions block to perform some device-specific action. These may take the form of Valid Query Requests or nonquery Executable Message Elements.

A Valid Query Request is an Executable Message Element which causes the Device Functions block to send data to the Response Formatter. A nonquery element performs device-specific action but does not send Response Data to the Response Formatter. A single <PROGRAM MESSAGE UNIT> may cause more than one Executable Message Element to be sent.

For example, a query which starts a measurement and returns the results when the measurement is complete might require two Executable Message Elements: a nonquery element to start the measurement and a Valid Query Request to send the results to the Response Formatter.

The Execution Control also shall maintain the order of creating <RESPONSE MESSAGE UNIT> elements (as required by 6.4.5.4), by sending the Valid Query Requests, associated with previous received <QUERY MESSAGE UNITS> that are already sent to the Device Functions block.

6.1.7.2.2 Execution Control Idle Message (ec-idle). The Execution Control shall set the Execution Control Idle Message, **ec-idle**, TRUE when the following conditions occur:

- (1) All deferred commands have been executed, and
- (2) Execution Control is ready to accept a new command or query from the Parser, and
 - (a) Execution Control is reset, or
 - (b) All previous commands have been executed

The Execution Control shall set **ec-idle** FALSE at all other times.

6.1.7.2.3 Execution Control Blocked Message (ec-blocked). The Execution Control shall set the Execution Control Blocked Message, **ec-blocked**, TRUE when it is ready to send a Valid Query Request to the Device Functions block but must wait for the Response Formatter and Device Functions block to process a prior Valid Query Request. The Execution Control shall set **ec-blocked** FALSE at all other times.

6.1.8 Device Functions. The Device Functions block, shown in both Figs 6-1 and 6-2, contains all device-specific features and functions. It accepts Executable Message Elements from the Execution Control and performs the associated operations.

6.1.8.1 Device Functions Rules. The Device Functions block shall accept Valid Query Requests from the Execution Control and shall send any required Response Data to the Response Formatter. All implemented IEEE 488.2 common query commands and all device-specific query commands shall be handled in this manner.

The Device Functions block shall report device status information to the Status Reporting block, Section 11.

6.1.8.2 Device Functions Messages

6.1.8.2.1 Power-On Message (pon). The Device Functions block shall generate the pon local message as described in IEEE 488.1.

When pon is set TRUE, the Input Buffer and Output Queue shall be cleared, the Parser, Execution Control, and Response Formatter shall be reset, and the device shall enter the Message Exchange IDLE state. The device also enters Operation Complete Command Idle State (OCIS) and Operation Complete Query Idle State (OQIS), see 12.5. The pon message may also affect device-specific settings, see 5.12. The Device Functions block shall set the pon message FALSE when these actions are completed and the Message Exchange Control enters the IDLE state (see 6.3.1.1).

6.1.8.2.2 Response Data. Response Data is data provided by the **device** in response to <QUERY MESSAGE UNIT> elements that have been parsed and sent to the **device** in the form of Executable Message Elements (Valid Query Requests). Response Data is unformatted and may only be tokens representing the actual data to be formatted.

6.1.9 Response Formatter. The Response Formatter, shown in Fig 6-2, builds a <RESPONSE MESSAGE> out of Response Message Elements from Valid Query Requests and response data. The <RESPONSE MESSAGE> is placed into the Output Queue. The Response Formatter's primary responsibility is to convert the internal representation of data elements into a sequence of data bytes according to the syntax rules in Section 8.

6.1.9.1 Response Formatter Rules. The Response Formatter shall delimit all but the last <RESPONSE MESSAGE UNIT> in the <RESPONSE MESSAGE> with a <RESPONSE MESSAGE UNIT SEPARATOR> (";", see 8.4.1), and the last <RESPONSE MESSAGE UNIT> in the <RESPONSE MESSAGE> with the <RESPONSE MESSAGE TERMINATOR>.

The Response Formatter shall be reset when **dcas** or **pon** is TRUE. No device or interface condition is allowed to keep the reset operation from taking place. The Response Formatter is reset so that any new <RESPONSE MESSAGE> elements will not be affected by conditions that existed before **dcas** or **pon**.

6.1.9.2 Response Formatter Messages

6.1.9.2.1 Response Message Element. A Response Message Element represents a **device's** internal representation of all or part of a <RESPONSE MESSAGE>.

6.1.9.2.2 Response Formatter Blocked Message (rf-blocked). The Response Formatter shall set the Response Formatter Blocked message, **rf-blocked**, TRUE when it has a <RESPONSE MESSAGE

UNIT> to be placed in the Output Queue and the Output Queue Full message, **oq-full**, is TRUE. The Response Formatter shall set **rf-blocked** FALSE at all other times.

6.1.10 Output Queue. The Output Queue, shown in Fig 6-2, stores device-to-controller messages until the **controller** reads them. The Response Formatter places DAB and END messages into the Output Queue in response to query commands. These bytes are removed from the Output Queue as they are read by the **controller**.

The Output Queue shown in Fig 6-2 is a logical model. The Output Queue in a real device may store the individual bytes of the <RESPONSE MESSAGE> or tokens which represent <RESPONSE MESSAGE> elements. The device designer may also use other means of determining the response to be generated.

The device designer should provide an Output Queue large enough to handle reasonably long <RESPONSE MESSAGE> elements. The actual length chosen will depend on the characteristics of the **device**. To avoid overflowing a physical buffer, the device designer may defer the formatting of lengthy <RESPONSE MESSAGE> elements until the **controller** requests output. The device designer should ensure that the size and operation of the Output Queue are adequate to make a deadlock unlikely, see 6.5.7.4.

6.1.10.1 Output Queue Rules. The Output Queue shall be cleared when **pon** or **dcas** is TRUE. Note that this is not a Query Error. A Query Error shall be reported if the contents of the Output Queue are discarded for any other reason, see 6.3.1.7, 6.3.2.2, and 6.3.2.3.

6.1.10.2 Output Queue Messages

6.1.10.2.1 Message Available Message (MAV). The Output Queue shall send the MAV message to the Status Reporting block. As long as the Output Queue contains one or more bytes, MAV shall be TRUE. MAV shall be FALSE when the Output Queue is empty except as indicated below. See 11.2.1.2.

When the **device** defers generation of Response Data until **brq** is TRUE, the MAV message shall be set TRUE when the device is ready to generate the Response Data. See 6.4.5.4.

6.1.10.2.2 Output Queue Full Message (oq-full). The Output Queue Full message, **oq-full**, shall be TRUE when the Output Queue is full. The **oq-full** message shall be FALSE at all other times.

6.1.11 Trigger Control. The Trigger Control block, shown in Fig 6-2, is optional. It allows for high speed hardware execution of GET messages without violating the rules of execution order.

The GET message is commonly used to trigger a device-specific action or set of actions within a single addressed device. It is also used to trigger synchronized actions across multiple addressed devices. In this and other cases where it may be necessary to avoid the software/firmware execution time overhead incurred in the Input Buffer, Parser, and Execution Control blocks, the device designer may choose to implement the Trigger Control block.

If the Trigger Control block is implemented, GET messages are not sent directly to the Input Buffer from the I/O Control. Instead **get** is sent to the Trigger Control Block which either sends a **trigger** message directly to the Device Functions, or places a GET in the Input Buffer.

6.1.11.1 Trigger Control Rules. The Trigger Control block shall detect the **get** message and check **ib-empty** (from the Input Buffer), **p-idle** (from the Parser), and **ec-idle** (from the Execution Control). If all four of these messages are TRUE, the Trigger Control block shall send the **trigger** message to the Device Functions. If either **ib-empty**, **p-idle**, or **ec-idle** is FALSE, the Trigger Control block shall place the GET message in the Input Buffer.

NOTE: The implementation of this optional Trigger Control block does not affect the **device's** response to GET nor relax the requirement of sequential processing of GET messages and DAB's. This block only facilitates high speed operation.

6.1.12 Message Exchange Control. The Message Exchange Control represents the interconnection of control messages between the Output Queue, Response Formatter, Input Buffer, Parser, Execution Control, I/O Control, and Device Functions blocks.

6.2 Protocol Overview. This protocol overview describes the normal operation of the device message exchange protocol. It is not a detailed specification of the protocol, nor does it cover any protocol exceptions. The detailed specification of the device message exchange protocol begins in 6.3.

Figure 6-3 shows the Message Exchange states and transitions encountered during normal operation of the device. It does not show certain required states and transitions associated with error recovery. In 6.3 and Fig 6-4 describe the complete Message Exchange State Diagram with the dashed lines indicating the exception or error cases.

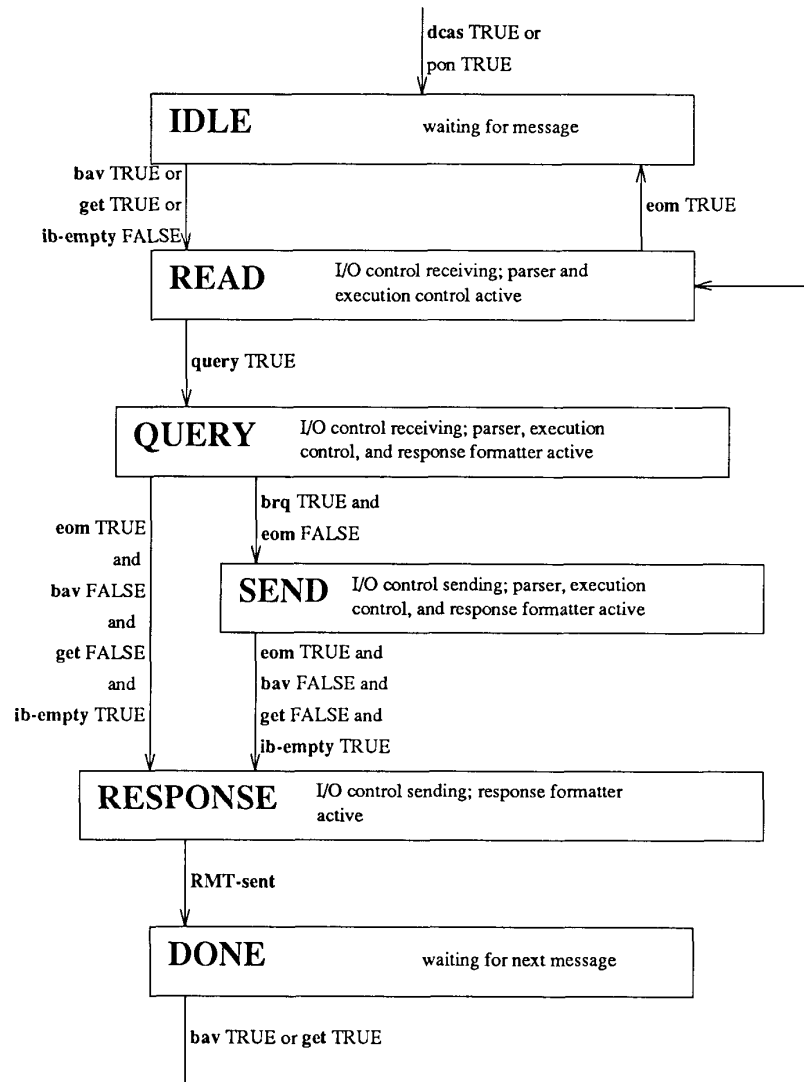
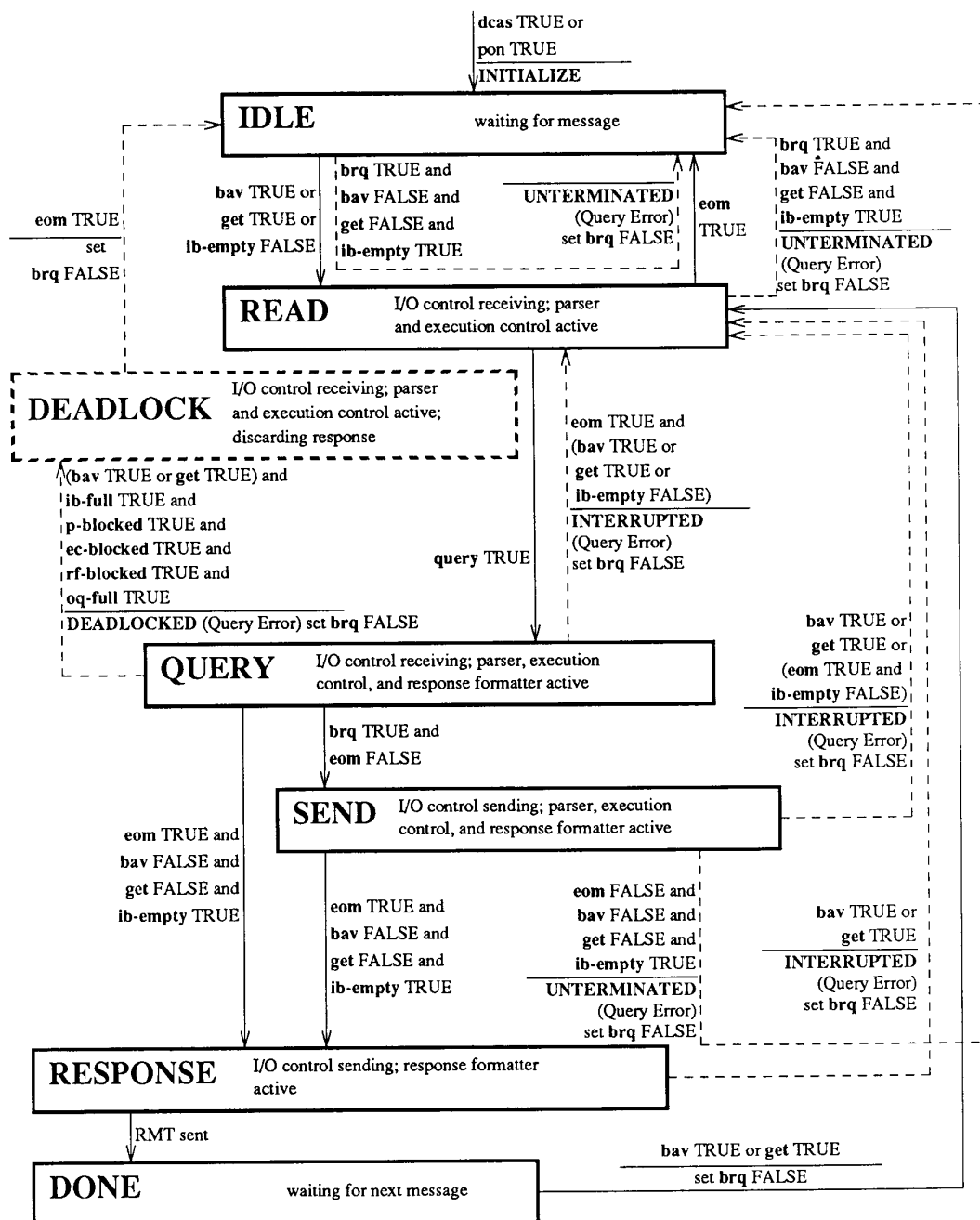


Fig 6-3
Message Exchange Control State Diagram (Simplified)

Figures 6-3 and 6-4 use a different format from IEEE 488.1 to reinforce that Message Exchange Control is done at a level higher than IEEE 488.1.

6.2.1 Initialization. After power on or **dcas**, the Message Exchange Control shall wait in the IDLE state for a <PROGRAM MESSAGE> or GET message from the **controller**. The device shall not send a <RESPONSE MESSAGE> to the **controller** until it receives a valid Query Message from the **controller**. The controller will not normally attempt to read data from the **device** until it has sent a Query Message to the device.



6.2.2 Command Processing. When the **device** receives a <PROGRAM MESSAGE> from the **controller** while in the READ state:

- (1) The I/O Control places the bytes of the message into the Input Buffer.
- (2) The Parser removes the bytes from the Input Buffer and identifies the syntactic components of the message according to the syntax rules in Section 7.
- (3) The Execution Control directs the Device Functions block to perform the actions associated with the message.

The **device** may accept the bytes of another <PROGRAM MESSAGE> and place them in the Input Buffer before it has finished processing prior <PROGRAM MESSAGE> elements.

6.2.3 Query Processing. Query Processing is handled in the QUERY, SEND, RESPONSE, and DONE states. If a <PROGRAM MESSAGE> contains one or more queries, the **device** prepares a <RESPONSE MESSAGE> for the **controller** and places it in the Output Queue. After the controller finishes sending the Query Message to the device, it will address the **device** to talk and start reading the <RESPONSE MESSAGE>. The controller will not normally send another <PROGRAM MESSAGE> to the **device** until it has finished reading the <RESPONSE MESSAGE>.

6.3 Message Exchange Control Operation. The Message Exchange Control State Diagram, Fig 6-4, precisely defines how messages, described in 6.1.4, from the I/O Control cause the **device** to either receive or send messages. The interactions between the blocks in the Message Exchange Control Interface, Fig 6-2, are shown in detail. Figure 6-3, shown before, includes only those states and transitions encountered during “normal” operation of the device. Figure 6-4 includes the additional states and transitions necessary to detect Message Exchange Protocol errors. (These additional states and transitions are shown with dashed lines.)

6.3.1 Message Exchange Control States. At any time, the **device** is in one of the Message Exchange states. Depending on the state, various functional elements from Fig 6-2 are active or inactive. Transitions are caused by the logical combination of messages from the functional elements.

6.3.1.1 IDLE State. In the IDLE state the **device** is waiting for a message from the **controller**. The next data byte received shall be interpreted as the beginning of a new <PROGRAM MESSAGE>. The Output Queue is empty.

The Message Exchange Control shall enter the READ state when either:

- (1) The I/O control sets **bav** TRUE, indicating that a data byte is available, or
- (2) The I/O Control sets **get** TRUE, or
- (3) **ib-empty** is FALSE. This condition can occur when the **device** had entered the IDLE state from the READ or DEADLOCK state.

The Message Exchange Control shall stay in the IDLE state when:

- (1) **brq** is TRUE, and
- (2) **bav** is FALSE, and
- (3) **get** is FALSE, and
- (4) **ib-empty** is TRUE

This occurs when the **device** has been addressed to talk and has nothing to say. The **device** shall perform the UNTERMINATED action described in 6.3.2.2.

The Message Exchange Control shall stay in the IDLE state when either the **dcas** or **pon** is TRUE. The **device** shall perform the INITIALIZE action described in 6.3.2.1.

6.3.1.2 READ State. In the READ state the I/O Control shall read data bytes, GET, and END messages from the IEEE 488.2 bus and place them in the Input Buffer as described in 6.1.4. The Parser and Execution Control are active and the Output Queue is empty.

The Message Exchange Control shall enter the IDLE state when either **dcas** or **pon** is TRUE. The **device** shall perform the INITIALIZE action described in 6.3.2.1.

The Message Exchange Control shall enter the IDLE state when each of the following conditions occur:

- (1) **brq** is TRUE, and
- (2) **bav** is FALSE, and
- (3) **get** is FALSE, and
- (4) **ib-empty** is TRUE

The **device** shall perform the UNTERMINATED action described in 6.3.2.2.

NOTE: Entering the IDLE state without terminating the <RESPONSE MESSAGE> could result in a controller time out when the expected data is not sent. When this happens an *ESR? (Standard Event Status Register Query, 10.12) can be used to determine that no data was sent because a Query Error had occurred.

The Message Exchange Control shall enter the IDLE state when the Parser sets **eom** TRUE.

The Message Exchange Control shall enter the QUERY state when the Parser sets **query** TRUE. See 6.1.6.2.3.

6.3.1.3 QUERY State. In the QUERY state the Parser has recognized a valid query within a <PROGRAM MESSAGE>. The I/O Control shall continue to read data bytes from the IEEE 488.1 bus and place them in the Input Buffer as described in 6.1.4.2.4. The Parser and Execution Control are active. The **device** may format its <RESPONSE MESSAGE> at this time, or may wait until the Message Exchange Control is in the SEND or RESPONSE states; see 6.4.5.4.

The Message Exchange Control shall enter the SEND state when **brq** is TRUE and an **eom** has not been parsed. This transition indicates that the **controller** has started to read the response.

The Message Exchange Control shall enter the RESPONSE state when the Parser sets **eom** TRUE, **bav** is FALSE, **get** is FALSE, and **ib-empty** is TRUE. This transition indicates that the **device** has finished processing the Query Message and is waiting for the **controller** to read the response.

The Message Exchange Control shall enter the DEADLOCK state when all of the following occur:

(1) The I/O Control sets **bav** or **get** TRUE, indicating that the **controller** is waiting to send data to the **device**, and

- (2) **ib-full** is TRUE, and
- (3) **p-blocked** is TRUE, and
- (4) **ec-blocked** is TRUE, and
- (5) **rf-blocked** is TRUE, and
- (6) **oq-full** is TRUE

This transition shall cause the **device** to set the Query Error bit in the Standard Event Status Register, clear the Output Queue, and reset the Response Formatter.

The Message Exchange Control shall enter the READ state when the following conditions occur:

- (1) **eom** is TRUE and either
- (2) **bav** is TRUE, or
- (3) **get** is TRUE, or
- (4) **ib-empty** is FALSE

The **device** shall perform the INTERRUPTED action described in 6.3.2.3.

The Message Exchange Control shall enter the IDLE state when either **dcas** or **pon** is TRUE. The **device** shall perform the INITIALIZE action described in 6.3.2.1.

6.3.1.4 SEND State. In the SEND state the **controller** has started to read the <RESPONSE MESSAGE> from the **device**. The **device** continues to parse and execute the Query Message stored in the Input Buffer. The Response Formatter prepares the <RESPONSE MESSAGE>, and places it in the Output Queue. The I/O Control shall send data bytes from the Output Queue to the **controller** as described in 6.1.4.2.3.

The Message Exchange Control shall enter the RESPONSE state when **eom** is TRUE, **bav** is FALSE, **get** is FALSE, and **ib-empty** is TRUE.

The Message Exchange Control shall enter the IDLE state when all of the following conditions occur:

- (1) **eom** is FALSE, and
- (2) **bav** is FALSE, and
- (3) **get** is FALSE, and
- (4) **ib-empty** is TRUE

The **device** shall perform the UNTERMINATED action described in 6.3.2.2.

The Message Exchange Control shall enter the READ state when any of the following conditions occur:

- (1) **bav** is TRUE, or
- (2) **get** is TRUE, or
- (3) The Parser sets **eom** TRUE and **ib-empty** is FALSE

This condition indicates that the **controller** started sending a new message before reading all of the response. The **device** shall perform the INTERRUPTED action described in 6.3.2.3.

The **device** shall enter the IDLE state when either **dcas** or **pon** is TRUE. The **device** shall perform the INITIALIZE action described in 6.3.2.1.