

Recognized as an
American National Standard (ANSI)

IEEE Std 488.2-1992(R2004)
(Revision of IEEE Std 488.2-1987)

IEEE Standard Codes, Formats, Protocols, and Common Commands for Use With IEEE Std 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation

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

Reaffirmed March 25, 2004
Approved June 18, 1992

IEEE Standards Board

Approved January 4, 1993
American National Standards Institute

Abstract: A set of codes and formats to be used by devices connected via the IEEE 488.1 bus is specified. This standard also defines communication protocols that are necessary to effect application-independent and device-dependent message exchanges, and further defines common commands and characteristics useful in instrument system applications. It is intended to apply to small-scale to medium-scale instrument systems comprised mainly of measurement, stimulus, and interconnect devices outside the scope of the instrument system environment. IEEE 488.1 subsets, standard message-handling protocols including error handling, unambiguous program and response-message syntactic structures, common commands useful in a wide range of instrument system applications, standard status reporting structures, and system configuration and synchronization protocols are covered.

Keywords: controller, device, system, system bus, system interface

Copyright © 1992 by The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA
ISBN 1-55937-238-9

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.

 **Documents**
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25, 2026 at 16:15:03 UTC from IEEE Xplore. Restrictions apply.

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 that 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 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
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
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 IEEE Std 488.2-1992, IEEE Standard Codes, Formats, Protocols, and Common Commands for Use With IEEE Std 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation.)

IEEE Std 488 has been in existence for seventeen years, 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. A large and growing base of design and application experience has allowed this further standardization of the IEEE Std 488 bus. The additional standardization sought by the original IEEE Std 488.2-1987 was founded on the premise that the existing investment in IEEE Std 488 resources must be protected. Thus, IEEE Std 488.2 describes functionality that complements and is based upon IEEE Std 488.1-1987. Every effort has been made to reap the benefits of standardization without limiting the freedom and creativity of the device designer.

Since the introduction of IEEE Std 488-1978, individual manufacturers of devices have developed internal standards that address the code, format, 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.

This revision of the original 1987 document contains many minor changes, too numerous to list, plus a few major changes. The changes are intended to be optional additions and clarifications to the 1987 standard. The goal of the working group was to allow devices and controllers designed in compliance with IEEE Std 488.2-1978 to remain in compliance with IEEE Std 488.2-1992.

Major new features include the addition of an <EXPRESSION RESPONSE DATA> element; the expanded use of <SUFFIX PROGRAM DATA>; the addition of the *RMC, Remove Individual Macro Command; the addition of the *SDS, Save Default Device Settings Command; the modification of *DDT, *DMC, *PUD, and *RDT commands to accept block or string data; and a new appendix on Reset Guidelines. There are many clarifications including a macro expansion algorithm; the removal of ambiguity in the definition of suffixes; and the scope of *LRN?, *SAV, *RCL, and *RST, to name a few.

The hope is that this new revision, incorporating five years of experience with IEEE Std 488.2-1987, will provide an even more solid foundation for the standardized communication of test and measurement systems.

At the time that this standard was completed, the working group had the following membership:

Bob Cram, Chair

L. Atchison	W. B. Groseclose	B. Nowlin
G. Blyth	C. Hagerling	R. Oblad
M. Brindle	M. G. Harrison	J. Pieper
R. Chojnacki	M. Hinds	A. Preuss
J. A. Colby	K. E. Johansen	L. Sollman
G. Cushing	S. G. Klaud	S. Tarr
P. D'Angelo	S. Lomas	D. Taylor
P. Davis	D. Lyons	A. Tracht
S. Greer	G. Meldrum	

At the time that it balloted and approved this standard for submission to the IEEE Standards Board, the balloting committee had the following membership:

L. Atchison	P. D'Angelo	W. B. Groseclose
S. E. Barryte	P. Davis	M. Harrison
D. Boyle	C. G. Gorss, Jr.	D. C. Hart
R. Cram	B. Gould	S. G. Klaud
G. Cushing	S. Greer	S. Lomas

D. Lyons
M. Maciejewski
A. Meloni
R. H. Nesson

W. Nowlin
R. Oishi
E. Sacher
J. M. Schachner

L. Sollman
P. Stuckert
C. Thomsen
A. Tracht

When the IEEE Standards Board approved this standard on June 18, 1992, it had the following membership:

Marco W. Migliaro, *Chair*
Donald C. Loughry, *Vice Chair*
Andrew G. Salem, *Secretary*

Dennis Bodson
Paul L. Borrill
Clyde Camp
Donald C. Fleckenstein
Jay Forster *
David F. Franklin
Ramiro Garcia
Thomas L. Hannan

Donald N. Heirman
Ben C. Johnson
Walter J. Karplus
Ivor N. Knight
Joseph Koepfinger *
Irving Kolodny
D. N. "Jim" Logothetis
Lawrence V. McCall

T. Don Michael *
John L. Rankine
Wallace S. Read
Ronald H. Reimer
Gary S. Robinson
Martin V. Schneider
Terrance R. Whittemore
Donald W. Zipse

* Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
James Beall

Richard B. Engelman
David E. Soffrin

Stanley Warshaw

Adam Sicker
IEEE Standards Project Editor

CLAUSE	PAGE
1. Introduction	1
1.1 Scope	1
1.2 Objectives.....	2
1.3 Notation.....	2
2. References	2
3. System Considerations	3
3.1 Definitions.....	3
3.2 System Message Traffic.....	4
3.3 Functional Layers.....	5
4. Device Compliance Criteria	6
4.1 IEEE 488.1 Requirements	7
4.2 Message Exchange Requirements.....	7
4.3 Syntax Requirements	7
4.4 Status Reporting Requirements.....	9
4.5 Common Commands.....	12
4.6 Synchronization Requirements	15
4.7 System Configuration Capability	15
4.8 Controller Capability.....	16
4.9 Device Documentation Requirements.....	16
5. Device Interface Function Requirements.....	17
5.1 Handshake Requirements.....	17
5.2 Address Requirements	18
5.3 Talker Requirements	18
5.4 Listener Requirements	18
5.5 Service Request Requirements.....	19
5.6 Remote/Local Requirements.....	19
5.7 Parallel Poll Requirements	21
5.8 Device Clear Requirements	22
5.9 Device Trigger Requirements	22
5.10 Controller Function Requirements.....	22
5.11 Electrical Requirements	22
5.12 Power-On Requirements	23
6. Message Exchange Control Protocol	24
6.1 Functional Elements.....	24
6.2 Protocol Overview	36
6.3 Message Exchange Control Operation.....	38
6.4 Protocol Rules	43
6.5 Protocol Exceptions	46
7. Device Listening Formats	48
7.1 Overview	48
7.2 Notation.....	49
7.3 Terminated Program Messages — Functional Syntax.....	52

CLAUSE	PAGE
7.4 Separator Functional Elements	56
7.5 <PROGRAM MESSAGE TERMINATOR>.....	58
7.6 Program Header Functional Elements	58
7.7 <PROGRAM DATA> Functional Elements	63
8. Device Talking Elements	78
8.1 Overview	78
8.2 Notation.....	79
8.3 Terminated Response Messages — Functional Syntax	79
8.4 Separator Functional Elements	83
8.5 <RESPONSE MESSAGE TERMINATOR>	84
8.6 <RESPONSE HEADER>	85
8.7 <RESPONSE DATA> Functional Elements	86
9. Message Data Coding	97
9.1 ASCII 7 bit Codes	98
9.2 Binary 8 Bit Integer Codes.....	100
9.3 Binary Floating Point Code.....	101
10. Common Commands and Queries	104
10.1 *AAD, Accept Address Command	107
10.2 *CAL?, Calibration Query	107
10.3 *CLS, Clear Status Command	108
10.4 *DDT, Define Device Trigger Command.....	109
10.5 *DDT?, Define Device Trigger Query.....	110
10.6 *DLF, Disable Listener Function Command.....	111
10.7 *DMC, Define Macro Command.....	111
10.8 *EMC, Enable Macro Command.....	114
10.9 *EMC?, Enable Macro Query	115
10.10 *ESE, Standard Event Status Enable Command	116
10.11 *ESE?, Standard Event Status Enable Query	116
10.12 *ESR?, Standard Event Status Register Query	117
10.13 *GMC?, Get Macro Contents Query	117
10.14 *IDN?., Identification Query 118	
10.15 *IST?, Individual Status Query.....	120
10.16 *LMC?, Learn Macro Query.....	120
10.17 *LRN?, Learn Device Setup Query	121
10.18 *OPC, Operation Complete Command.....	122
10.19 *OPC?, Operation Complete Query.....	122
10.20 *OPT?, Option Identification Query	123
10.21 *PCB, Pass Control Back.....	124
10.22 *PMC, Purge Macros Command	125
10.23 *PRE, Parallel Poll Enable Register Command.....	125
10.24 *PRE?, Parallel Poll Enable Register Query.....	126
10.25 *PSC, Power-On Status Clear Command	127
10.26 *PSC?, Power-On Status Clear Query	128
10.27 *PUD, Protected User Data Command.....	128
10.28 *PUD?, Protected User Data Query	129
10.29 *RCL, Recall Command	130
10.30 *RDT, Resource Description Transfer Command	131

CLAUSE	PAGE
10.31 *RDT?, Resource Description Transfer Query	132
10.32 *RST, Reset Command	133
10.33 *SAV, Save Command	134
10.34 *SRE, Service Request Enable Command	134
10.35 *SRE?, Service Request Enable Query	135
10.36 *STB?, Read Status Byte Query	136
10.37 *TRG, Trigger Command	136
10.38 *TST?, Self-Test Query	137
10.39 *WAI, Wait-to-Continue Command	138
10.40 *RMC, Remove Individual Macro Command	138
10.41 *SDS, Save Default Device Settings Command	139
11. Device Status Reporting	140
11.1 Overview	140
11.2 Status Byte Register	141
11.3 Service Request Enabling	146
11.4 Status Data Structures	152
11.5 Standard Status Data Structure	157
11.6 Parallel Poll Response Handling	164
12. Device/Controller Synchronization Techniques	165
12.1 Overview	165
12.2 Sequential and Overlapped Commands	166
12.3 Pending-Operation Flag	167
12.4 No-Operation-Pending Flag	167
12.5 Controller/Device Synchronization Commands	167
12.6 Synchronization With External-Control-Signals	171
12.7 Improper Usage of *OPC and *OPC?	172
12.8 Design Considerations	173
13. Automatic System Configuration	175
13.1 Introduction	175
13.2 Overview	176
13.3 Generic Approach to Automatic System Configuration	177
13.4 Detailed Requirements of the Auto Configuration Commands	182
13.5 Additional Automatic Configuration Techniques	188
13.6 Examples	188
14. Controller Compliance Criteria	188
14.1 IEEE 488.1 Requirements	188
14.2 Message Exchange Requirements	189
14.3 Protocols	190
14.4 Functional Element Handling	191
14.5 Controller Specification Requirements	191
15. IEEE 488.2 Controller Requirements	191
15.1 Controller Interface Function Requirements	191
15.2 Additional IEEE 488.2 Controller Requirements	192
15.3 IEEE 488.2 Controller Recommendations	193

CLAUSE	PAGE
16. Controller Message Exchange Protocols	193
16.1 Definitions	194
16.2 Control Sequences	196
17. Common Controller Protocols	204
17.1 Reset Protocol	205
17.2 Find Device Requesting Service Protocol	206
17.3 Serial Poll All Devices Protocol	207
17.4 Pass Control Protocol	209
17.5 Requesting Control	210
17.6 Find Listeners Protocol	211
17.7 Set Address Protocol	212
17.8 Test System Protocol	217
Annex A Compound Headers — Usage and Examples (Informative)	219
Annex B Device/Controller Synchronization Techniques (Informative)	223
Annex C Automatic System Configuration Example (Informative)	235
Annex D Reset Guidelines (Informative)	241

IEEE Standard Codes, Formats, Protocols, and Common Commands for Use With IEEE Std 488.1-1987, 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 apply directly to small-scale 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 IEEE Std 488.1-1987 [4]¹ while remaining compatible with that standard.

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

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

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
- 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 that have special meaning in the context of this standard. These words have different connotations in IEEE Std 488.1-1987 [4], but are, for reasons of readability and conciseness of text, the preferred words for use in this standard. To avoid confusion, these words always appear in bold type when referenced in this document. Words that appear in bold type are: **system**, **device**, **controller**, **system bus**, and **system interface**. Local messages peculiar to IEEE 488.2 will also appear in lowercase bold type.

This standard also defines syntactic elements that are 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 X3.4-1986, American National Standard Code for Information Interchange Coded Character Set—7-Bit.²
- [2] ANSI X3.42-1990, American National Standard Representation of Numeric Values in Character Strings for Information Interchange.
- [3] IEEE Std 260-1978, (Reaff 1985). IEEE Standard Letter Symbols for Units of Measurement (SI Units, Customary Inch-Pound Units, and Certain Other Units) (ANSI, DoD).³
- [4] IEEE Std 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation (ANSI).

²ANSI publications are available from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

³IEEE publications are available from the Institute of Electrical and Electronics Engineers, Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

- [5] IEEE Std 754-1985 (Reaff 1991), IEEE Standard for Binary Floating-Point Arithmetic.
- [6] ISO 31-1: 1992, Quantities and units — Part 1: Space and time.⁴
- [7] ISO 31-2: 1992, Quantities and units — Part 2: Periodic and related phenomena.
- [8] ISO 31-3: 1992, Quantities and units — Part 3: Mechanics.
- [9] ISO 31-4: 1992, Quantities and units — Part 4: Heat.
- [10] ISO 31-5: 1992, Quantities and units — Part 5: Electricity and magnetism.
- [11] ISO 31-6: 1992, Quantities and units — Part 6: Light and related electromagnetic radiations.
- [12] ISO 31-7: 1992, Quantities and units — Part 7: Acoustics.
- [13] ISO 31-8 : 1992, Quantities and units — Part 8: Physical chemistry and molecular physics.
- [14] ISO 31-9: 1992, Quantities and units — Part 9: Atomic and nuclear physics.
- [15] ISO 31-10: 1992, Quantities and units — Part 10: Nuclear reactions and ionizing radiations.
- [16] ISO 31-11: 1992, Quantities and units — Part 11: Mathematical signs and symbols for use in the physical sciences and technology.
- [17] ISO 31-12: 1992, Quantities and units — Part 12: Characteristic numbers.
- [18] ISO 1000: 1981, SI units and recommendations for the use of their multiples and certain other units.
- [19] ISO 2955: 1983, Information processing—Representation of SI and other units in systems with limited character sets.
- [20] Mueller, J. E., “Efficient Instrument Design Using IEEE 488.2,” *IEEE Transactions on Instrumentation and Measurement*, vol. IM-39, pp. 146–150, Feb. 1990.

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 Std 488.1-1987 [4], see 1.3 of that standard.

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.

default: The choice used when no specification is given.

⁴ISO publications are available from ISO, Case Postale 56, 1 rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse. ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

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**. 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 that connects a **device** or **controller** to the **system bus**. A “non-IEEE 488.2 system interface” is any interface other than the **system interface** that 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, see Section 7, also may allow the use of an IEEE 488.2 component in a **system** that 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 syntactically 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.

These controller-to-device (program) and device-to-controller (response) messages are composed of syntactic elements that are 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.

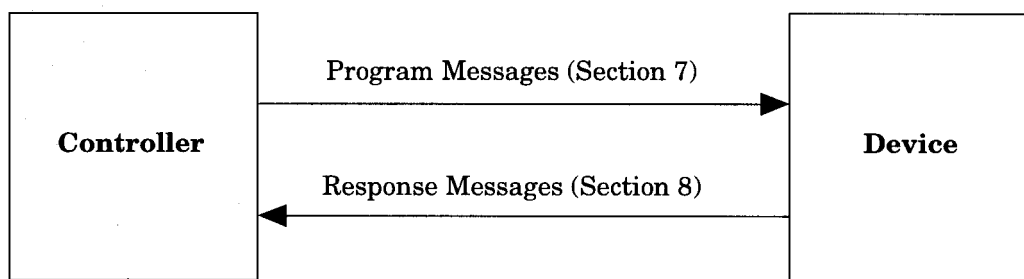


Figure 3-1 — Usual Message Traffic

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**

Controller Sends	Device Sends
RANGE?	12

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

3.2.2 Other Message Traffic

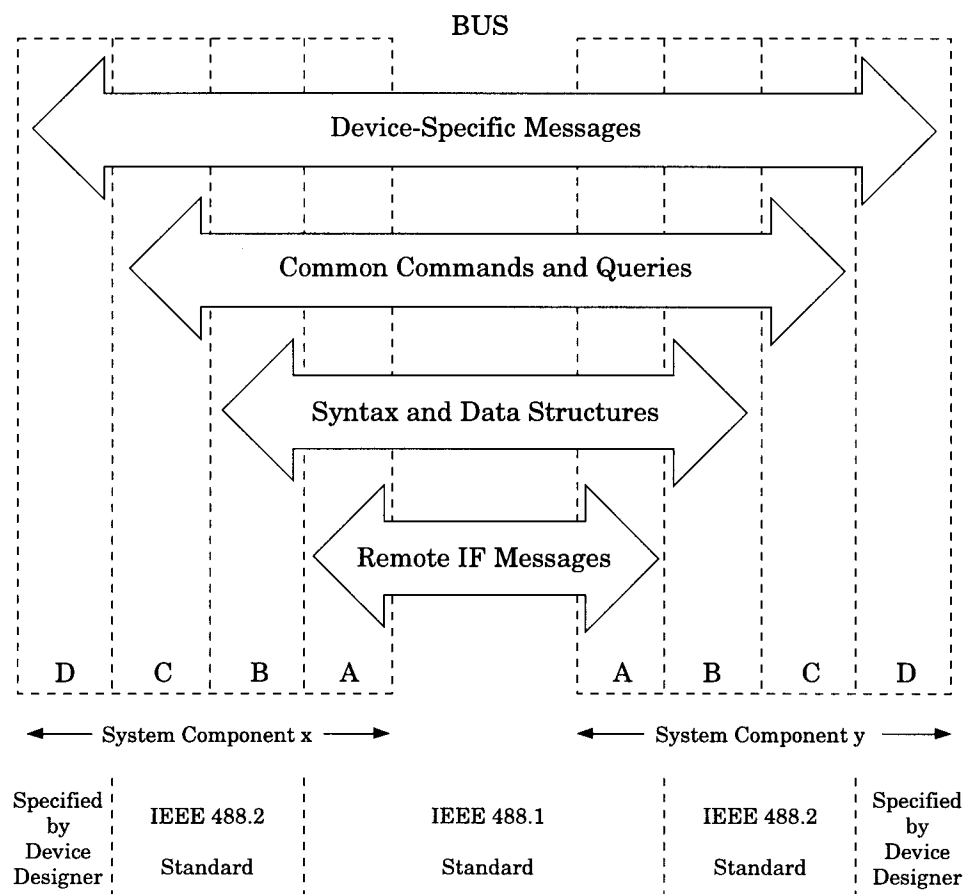
In addition to controller-to-device and device-to-controller messages, protocols are defined to facilitate passing control, see Section 17. Device-to-device protocols are not explicitly defined in this document. However, message traffic between **devices** shall follow the syntax for <RESPONSE MESSAGES> 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” that 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 488.1-1987 [4]. No attempt has been made to associate the protocol layering with the ISO Model for Open Systems Interconnection (OSI).



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)

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

Figure 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 that this standard requires in a **device**. A **device** may optionally contain additional capabilities. Any optional **device** capabilities that are described by this standard are also listed in this section.

Compliance for a **device** is divided into several areas that 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

that 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 the IEEE 488.1 subsets listed in Table 4-1 and no others.

Table 4-1—IEEE 488.1 Interface Requirement 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 that 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>	7.5.0
<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

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

Device Talking Functional Element	Section
<RESPONSE MESSAGE>	8.3.2
<RESPONSE MESSAGE TERMINATOR>	8.5.0
<RESPONSE MESSAGE UNIT>	8.3.2
<RESPONSE MESSAGE UNIT SEPARATOR>	8.4.1
<RESPONSE DATA>	8.7.0
<RESPONSE DATA SEPARATOR>	8.4.2
<NR1 NUMERIC RESPONSE DATA>	8.7.2
<ARBITRARY ASCII RESPONSE DATA>	8.7.11

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
<NONDECIMAL 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

* NOTE — If a **device** implements <compound command program header> and <compound query program header> elements, it does not implement <simple command program header> and <simple query program header> elements. See 7.6.1 and 7.6.2.

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
<STRING RESPONSE DATA>	8.7.8
<DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>	8.7.9
<INDEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>	8.7.10
<EXPRESSION RESPONSE DATA>	8.7.12

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.

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.

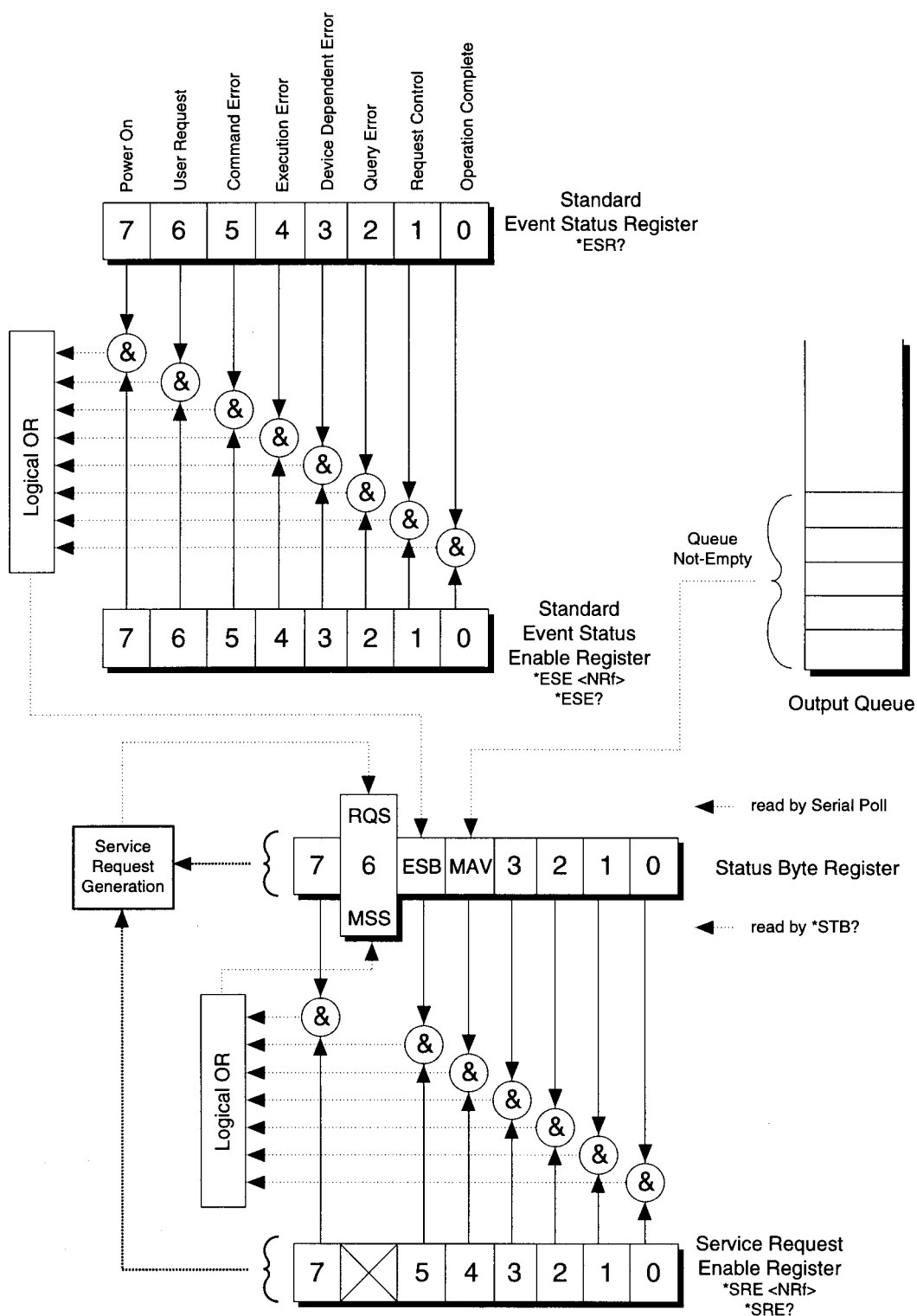


Figure 4-1—Required Status Reporting Capabilities

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(*), preceding CLS 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).

4.4.2 Optional Status Reporting Capability

A **device** may include any number of condition registers, event registers, enable registers, and queues, provided that 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 that are related to status reporting and the common commands listed in 4.6 that are 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 either command is implemented, then both commands 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. Implementing either command requires the **device** to have DT1 capability. Implementing *DDT requires the **device** to implement *DDT?.

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.

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 4-13—Optional Macro Commands

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

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 either command is implemented, then both commands 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 the capability that is required in all **devices** for synchronization. A **device** designer may choose which operations have pending-operation flags associated with them in accordance with 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 the optional capability that 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** to suspend temporarily 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 and 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

Stored Settings	
Command	Section
*PCB	10.21

4.9 Device Documentation Requirements

The documentation provided with a **device** shall describe how the **device** has implemented this standard. This documentation is required and shall include

- 1) A list of IEEE 488.1 Interface Functions subsets implemented, see Section 5.
- 2) A description of **device** behavior when the address is set outside the range of 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 that modify the power-on settings shall also be included. A list of device-specific enable registers affected by the *PSC common command shall 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 that may appear within an <expression> as well as the maximum subexpression nesting depth. Any additional syntax restrictions that the **device** may place on the <expression> shall also be included.
- 9) A description of the response syntax for every query, see Section 8.
- 10) A description of any device-to-device message transfer traffic that 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 that are implemented, including, where appropriate, indications of any optional functional elements implemented, see 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 (see 10.4.6.1) and the method of interpreting *TRG within a *DDT command sequence (see 10.4.6.3).
- 15) The maximum length and complexity 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) If the *DDT command is implemented,
 - a) The maximum length of the block used to define the trigger macro, see 10.4.
 - b) The command sequence sent with the *DDT command has the same effect as *RST on the action performed by GET, see 10.4.1 and 10.4.6.3.
- 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, see 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.
- 24) A description of the representations, if any, used for infinity and not-a-number, see 7.7.2.4.5 and 8.7.4.4.

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 11 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 Std 488.1-1987 [4], 6.3.) If the **device** utilizes extended addressing, the talk and listen secondary addresses (MSAs) shall also be identical.

A **device** shall have a single primary address that 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 Std 488.1-1987 [4], Appendix I.

The **device** designer should avoid using address settings outside the range of 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 locally to 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 that 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 or whose function is to protect the safety of the user or equipment 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 local controls. 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.) that 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 that 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 that 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 that 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 Std 488.1-1987 [4], 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 (see IEEE Std 488.1-1987 [4], Appendix H).

For **devices** without hard local controls, the Remote indicator is optional. All indicators relating to Remote/Local transitions shall be labeled and shall function in accordance with the requirements of IEEE Std 488.1-1987 [4], 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).

See Appendix D for a discussion of the resetting actions of **dcas** as they relate to other reset commands.

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, see 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 (CO) or shall 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 CO, 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 (see IEEE Std 488.1-1987 [4], 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 (see IEEE Std 488.1-1987 [4], 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 that 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, see 10.25.4), which sets and clears the power-on-status-clear flag.

See Appendix D for a discussion of the resetting actions of power-on and *PSC as they relate to other reset commands.

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 that cause a change in the response to the following common queries:
 - a) *IDN? (Identification Query, see 10.14)
 - b) *OPT? (Option Identification Query, see 10.20)
 - c) *PSC? (Power-On Status Clear Query, see 10.26)
 - d) *PUD? (Protected User Data Query, see 10.28)
 - e) *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 in 10.25.

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

The **device** designer may use the state of the power-on-status-clear flag to control whether other status registers are cleared at power-on.

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, see 5.12.2 and 11
- 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 that 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> that 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

Fig 6-1 presents the relationship between 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.

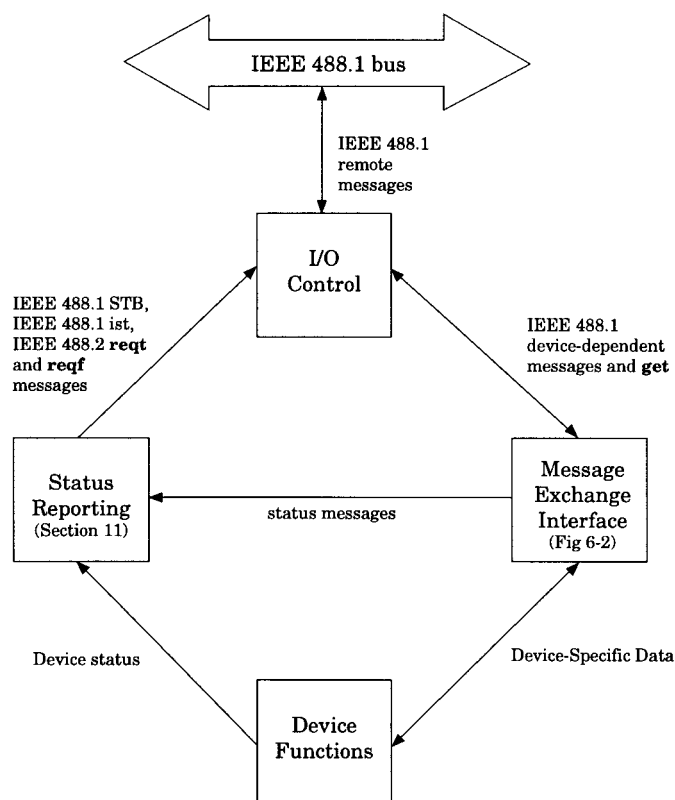


Figure 6-1—Device Status and Message Exchange Overview

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 Std 488.1-1987 [4].

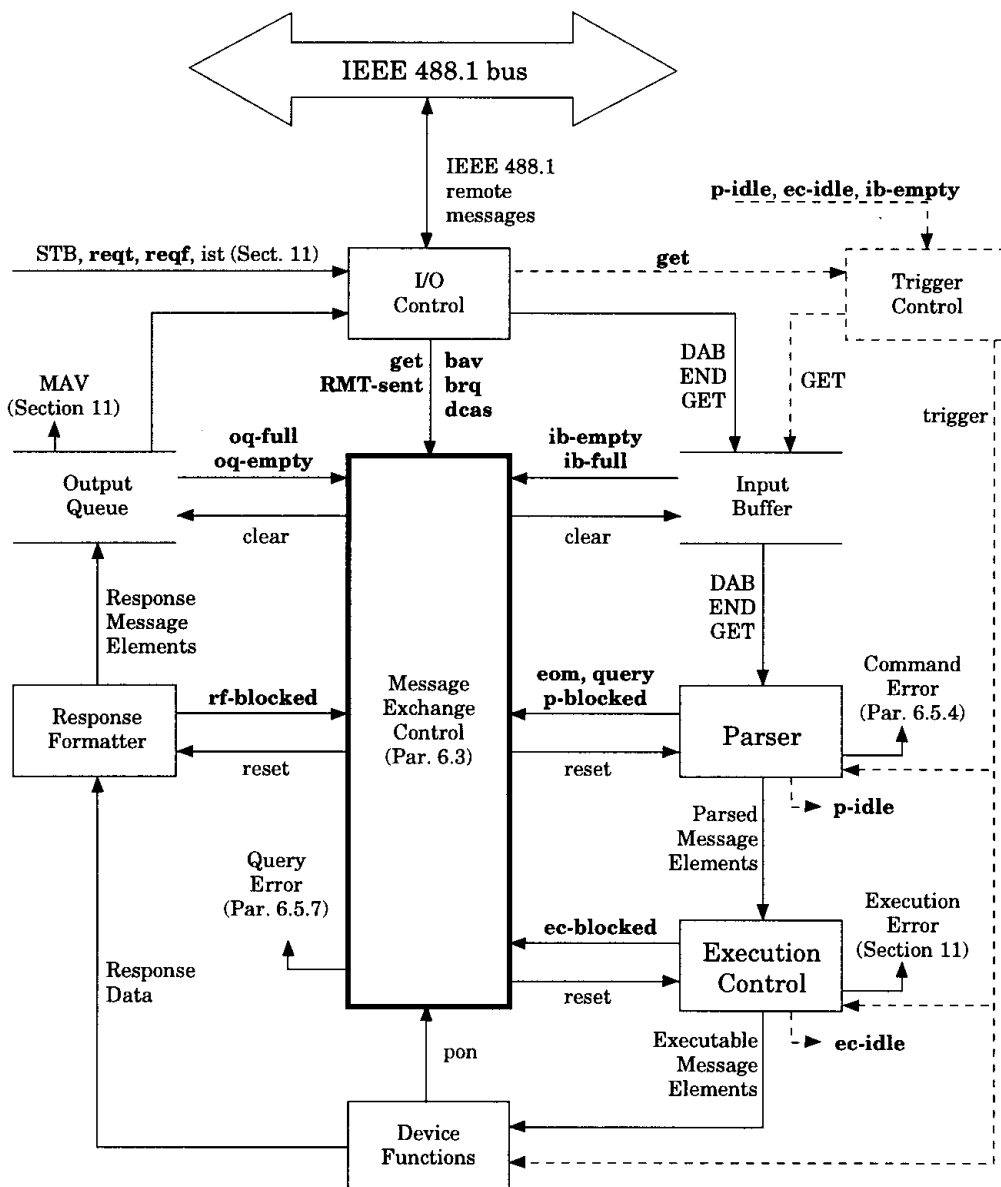


Figure 6-2—Message Exchange Control Interface Functional Blocks

6.1.2 Status Reporting

The Status Reporting block appears only in Fig 6-1. Fig 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 described completely 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 presents a logical model that 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.

NOTE — For an example of an implementation of the Message Exchange Interface, see [20].

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, see Section 11. It generates the **dcas**, DAB, END, **get**, **bav**, and **brq** messages that 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 6.1.11.

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 (DABs), 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 Clear handling as follows:

- 1) 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.
- 2) Ensure that 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.

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

- 1) 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."
- 2) 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.
- 3) Ensure that 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.

NOTE — The use of NDAC holdoff on GET allows the following situation when the Input Buffer is full. The **device** will assert NDAC as long as the Input Buffer is full. The **controller** cannot communicate with the **device**. A Device Clear cannot be sent. The only way to remove this condition is to have

- (1) The parser remove a DAB, END, or GET from the Input Buffer, or
- (2) The Device Function send the pon message to the Message Exchange Control

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.

This 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 optional Trigger Control Block may be used. When the Trigger Control Block receive a **get** message from the I/O Control Block, it generates a **trigger** local message for immediate execution by the **device**, provided that the Input Buffer is empty and the Parser and Execution Control are idle.

If the Input Buffer is not empty or if the Parser or Execution Control are not idle, the alternative GET message shall be placed in the Input Buffer, and execution of the trigger shall be 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), see 12.5, and returns to normal operation after the *AAD or *DLF commands, see 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 DABs, GET messages, and END messages. The Input Buffer then delivers these messages to the Parser in the order that 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. DABs, 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** that can accept program messages faster than they can execute them should have Input Buffers long enough to allow the storing of 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 that 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 that implement the requirements of IEEE 488.1 provide at least a 1 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 parser attempts to remove an item from the Input Buffer and the Input Buffer is empty. The **ib-empty** message shall be set FALSE when I/O Control Block places a DAB, END, or GET message into the Input Buffer.

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** that takes DABs, 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>, if it finds an unrecognized header, if it finds a parameter (<PROGRAM DATA> element) that is of the wrong type for its associated header, or if it 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 DABs and GET messages until either

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

The **device** designer may also choose to stop discarding DABs 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
- 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.
- 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
- 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, see 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> that is not a macro label
- 2) A valid <COMMAND PROGRAM HEADER> or <QUERY PROGRAM HEADER> that is a macro label, where the macro is defined to include a <QUERY MESSAGE UNIT>
- 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>
- 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 above.

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 that 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 **pen**. The Execution Control shall be reset when **INTERRUPTED**, or when **dcas** or **pen** 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. This may take the form of a Valid Query Request or a nonquery Executable Message Element.

A Valid Query Request is an Executable Message Element that 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 that 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 that are 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.
- 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 that 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.

NOTE — Certain integrated circuits contain a one byte buffer, which operates as part of the Output Queue when MAV is generated (see 6.1.10.2) and “Clear the Output Queue” is executed. A **device** designer cannot disregard this buffer. See 6.3.2.1(2), 6.3.2.2(2), and 6.3.2.3(2).

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.

6.1.10.2.3 Output Queue Empty Message (oq-empty)

The Output Queue Empty message, **oq-empty**, shall be TRUE when the Output Queue is empty. The **oq-empty** 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 in which 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 block, 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 block. 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 does it relax the requirement of sequential processing of GET messages and DABs. This block only facilitates highspeed 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.

Fig 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. Paragraph 6.3 and Fig 6-4 describe the complete Message Exchange State Diagram, with the dashed lines indicating the exception or error cases.

Figs 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>.

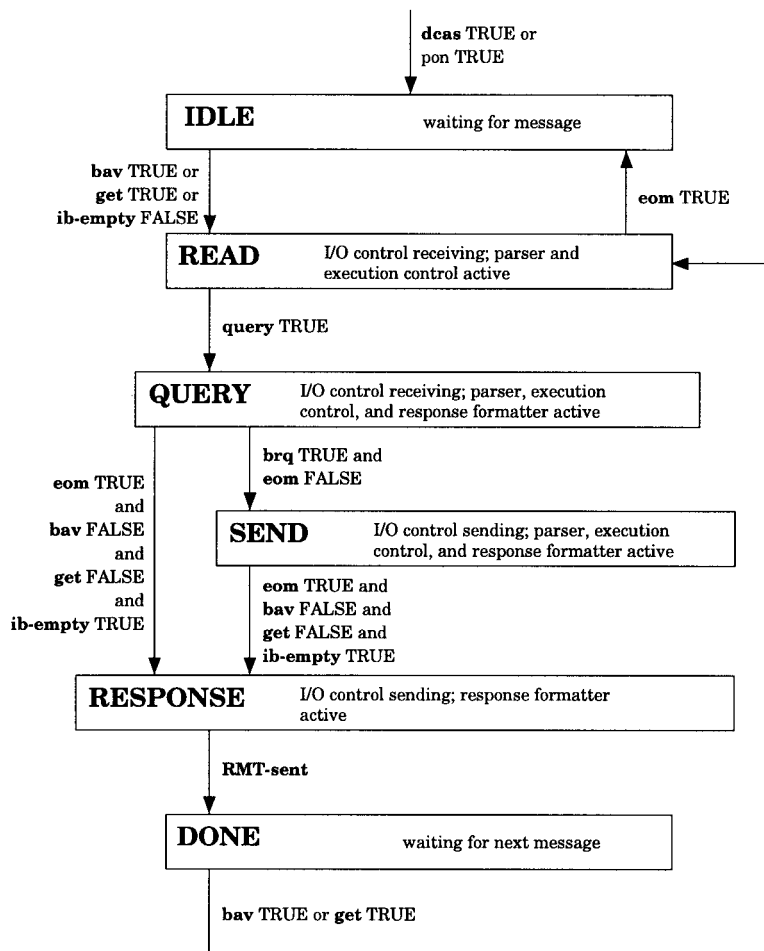


Figure 6-3—Message Exchange Control State Diagram (Simplified)

6.3 Message Exchange Control Operation

The Message Exchange Control State Diagram, Fig 6-4, precisely defines how messages from the I/O Control, described in 6.1.4, 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. Fig 6-3 includes only those states and transitions encountered during “normal” operation of the **device**. Fig 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.

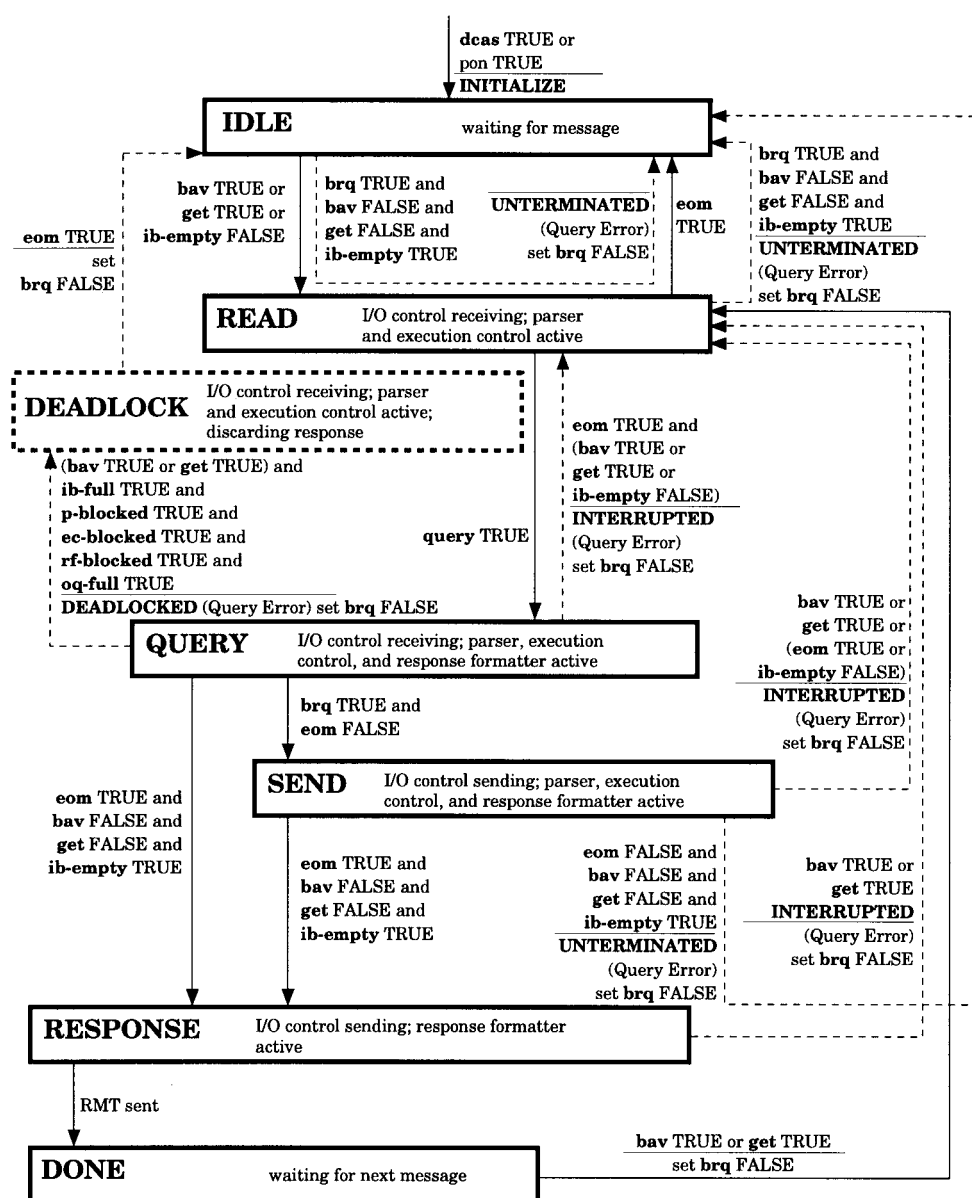


Figure 6-4—Message Exchange control State Diagram (Complete)

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.
- 2) The I/O Control sets **get** TRUE.
- 3) **ib-empty** is FALSE. This condition can occur when the **device** has 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.
- 2) **bav** is FALSE.
- 3) **get** is FALSE.
- 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 488.1 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.
- 2) **bav** is FALSE, and
- 3) **get** is FALSE.
- 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, see 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 it 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**.
- 2) **ib-full** is TRUE.
- 3) **p-blocked** is TRUE.
- 4) **ec-blocked** is TRUE.
- 5) **rf-blocked** is TRUE.
- 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.
- 2) **bav** is FALSE.
- 3) **get** is FALSE.
- 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.
- 2) **get** is TRUE.
- 3) The Parser sets **com** 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.

6.3.1.5 RESPONSE State

In the RESPONSE state, the Parser has received a <PROGRAM MESSAGE TERMINATOR> and the **device** is sending the complete <RESPONSE MESSAGE> to the **controller**. The Response Formatter formats the <RESPONSE MESSAGE> and the I/O Control sends data bytes from the Output Queue to the **controller** as described in 6.1.4.2.3.

The Message Exchange Control shall enter the DONE state when the complete <RESPONSE MESSAGE>, including the <RESPONSE MESSAGE TERMINATOR>, has been sent.

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

- 1) **bav** is TRUE.
- 2) **get** is TRUE.

This transition indicates that the **controller** attempted to send a new message before reading the complete <RESPONSE MESSAGE>. 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, see 6.3.2.1.

A **device** may have nothing to say while the Message Exchange Control is in the RESPONSE state if all of the queries in the Query Message failed to generate any <RESPONSE DATA> because of syntax (Command), semantic (Execution), or device-specific errors. The **device** shall not send a <RESPONSE MESSAGE TERMINATOR> to the **controller**. The Message Exchange Control shall remain in the RESPONSE state until it either receives a data byte or GET message from the **controller** or the **pon** or **dcas** message becomes TRUE.

6.3.1.6 DONE State

In the DONE state, the **device** has finished sending a <RESPONSE MESSAGE> to the **controller** and is waiting for a new <PROGRAM MESSAGE> or GET from the **controller**. The Parser, Execution Control, and Response Formatter are inactive and the Input Buffer and Output Queue are empty.

The Message Exchange Control shall enter the READ state and set **brq** FALSE when either

- 1) **bav** is TRUE, indicating that a new <PROGRAM MESSAGE> has been started, or
- 2) **get** is TRUE.

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.

6.3.1.7 DEADLOCK State

In the DEADLOCK state, the **device** has been asked to buffer more data than it has room to store. The Output Queue is full, blocking the Response Formatter, Execution Control, and Parser. The Input Buffer is full, and the **controller** is waiting to send more data bytes. to the **device**.

The Message Exchange Control breaks the deadlock by clearing the Output Queue and resetting the Response Formatter as it enters the DEADLOCK state (transition from QUERY to DEADLOCK, see 6.3.1.3). The Message Exchange Control also sets the Query Error bit in the Standard Event Status Register. While in the DEADLOCK state, the Message Exchange Control shall continue to parse and execute <PROGRAM MESSAGE UNIT> elements as usual, except that it shall discard query responses rather than place them into the Output Queue.

The Message Exchange Control shall enter the IDLE state and set **brq** FALSE when either of the following conditions occur:

- 1) *The Parser sets **com** TRUE.* When the **controller** attempts to read the <RESPONSE MESSAGE>, the **device** shall not send any data bytes. Note that not sending any data bytes may cause the **controller** to time out. When this happens, an *ESR? (Standard Event Status Register Query, see 10.12) can be used to determine that no data was sent because a Query Error had occurred.
- 2) *Either **dcas** or **pon** is TRUE.* The **device** shall perform the INITIALIZE action described in 6.3.2.1.

6.3.2 Message Exchange Control Transition Actions

Certain transitions among the states are important enough to be named. The actions caused by the transitions are essential for reliable error detection, reporting, and recovery.

6.3.2.1 INITIALIZE Action

The INITIALIZE action is executed when the IEEE 488.2 communication channel is to be cleared.

The **device** shall perform each of the following actions:

- 1) Clear the Input Buffer.
- 2) Clear the Output Queue.
- 3) Reset the Parser.
- 4) Reset the Execution Control.
- 5) Reset the Response Formatter.

No error condition shall be reported.

6.3.2.2 UNTERMINATED Action

The UNTERMINATED action is executed when the **controller** attempts to read a <RESPONSE MESSAGE> from the **device** without first having sent a complete Query Message, including the <PROGRAM MESSAGE TERMINATOR>, to the **device**.

The **device** shall perform each of the following actions:

- 1) Set the Query Error bit in the Standard Event Status Register.
- 2) Clear the Output Queue.
- 3) Optionally execute any <PROGRAM MESSAGE UNIT> elements from the incomplete message. If it executes any of the <PROGRAM MESSAGE UNIT> elements, it shall also execute all previous <PROGRAM MESSAGE UNIT> elements from the same message.