1451.0™

IEEE Standard for a Smart Transducer Interface for Sensors and Actuators—Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats

IEEE Instrumentation and Measurement Society

Sponsored by the
Technical Committee on Sensor Technology (TC-9)

IEEE
3 Park Avenue
New York, NY 10016-5997, USA

21 September 2007

IEEE Std 1451.0™-2007

Recognized as an
American National Standard (ANSI)

IEEE Std 1451.0™-2007

IEEE Standard for a Smart Transducer Interface for Sensors and Actuators—Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats

Sponsor

Technical Committee on Sensor Technology (TC-9)
of the
IEEE Instrumentation and Measurement Society

Approved 9 August 2007

American National Standards Institute

Approved 22 March 2007

IEEE-SA Standards Board

Abstract: This standard provides a common basis for members of the IEEE 1451 family of standards to be interoperable. It defines the functions that are to be performed by a transducer interface module (TIM) and the common characteristics for all devices that implement the TIM. It specifies the formats for Transducer Electronic Data Sheets (TEDS). It defines a set of commands to facilitate the setup and control of the TIM as well as reading and writing the data used by the system. Application programming interfaces (APIs) are defined to facilitate communications with the TIM and with applications.

Keywords: actuator, application programming interface, communication protocol, network-capable application processor, sensor, smart transducer, transducer electronic data sheet, transducer interface module

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2007 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 21 September 2007. Printed in the United States of America.
2nd Printing 2 November 2007. A printing error for Equation (15) has been corrected.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Incorporated.

Print: ISBN 0-7381-5597-7 SH95684
PDF: ISBN 0-7381-5598-5 SS95684

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

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "**AS IS**."

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.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

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 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 societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

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

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854
USA

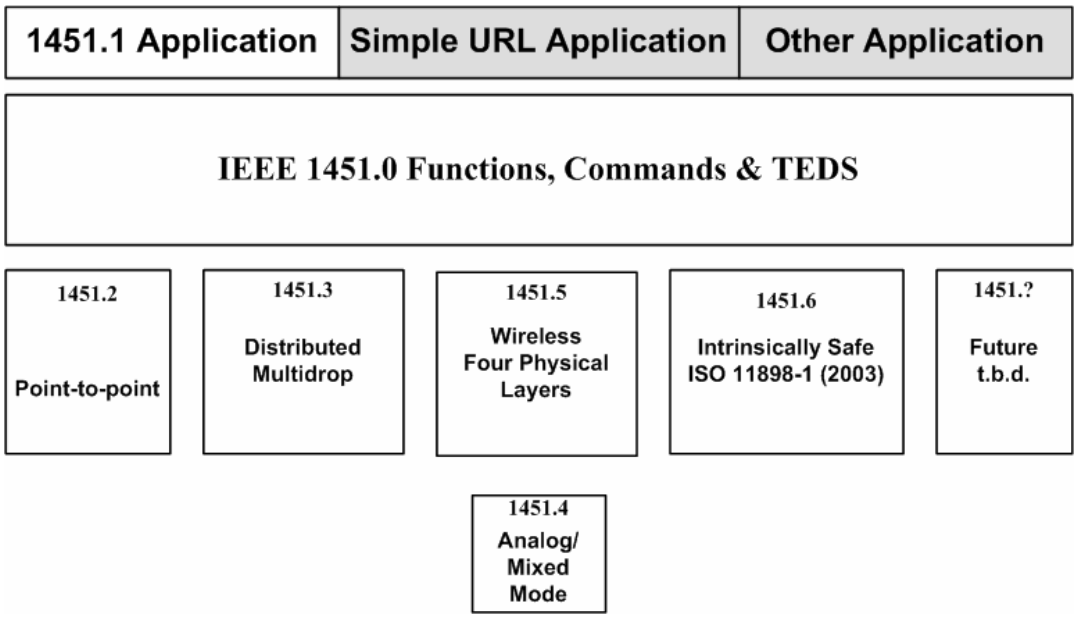
Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

This introduction is not part of IEEE Std 1451.0-2007, IEEE Standard for a Smart Transducer Interface for Sensors and Actuators—Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats.

This standard is intended to provide a basis for all future members of the IEEE 1451 family of standards that use digital interfaces. It should also be adopted by the existing members of the IEEE 1451 family of standards as they are revised in the future in order to provide the highest degree of compatibility among the members of the family. This standard does not apply to IEEE Std 1451.4™-2004, which only provides a size-constrained TEDS and an analog interface.

The relationships between this standard and the other members of the family are shown in the following diagram. Three of these standards were complete before this standard was started and do not comply with this standard but will in the future as they are revised. They are IEEE Std 1451.1™-1999, IEEE Std 1451.2™-1997, and IEEE Std 1451.3™-2003. IEEE Std 1451.1 is an application that, in the future, will fit between the user's network and this standard. IEEE Std 1451.2 and IEEE Std 1451.3 will also be modified to interface with this standard. When these changes are made, the functions of an IEEE 1451 transducer will be as defined in this standard as will be the commands and TEDS. IEEE 1451.5™-2007, which uses any of several different wireless communications media, and IEEE P1451.6™ have been written around the functions, commands, and TEDS as described in this standard. IEEE Std 1451.4 uses an analog signal interface and a TEDS that is not the same as that used by other members of the family. It may be used as the input to any of the other standards in the family but does not comply with the functions, commands, and TEDS defined in this standard. Items shown with a gray background are items that are not covered by any of the IEEE 1451 family of standards but that may be used.



The underlying purpose of this family of standards is to allow manufacturers to build elements of a system that are interoperable. To accomplish this goal, the IEEE 1451 family of standards divides the parts of a system into two general categories of devices. One is the network capable application processor (NCAP) that functions as a gateway between the users' network and the transducer interface modules (TIMs). The NCAP is a processor-based device that has two interfaces. The physical interface to the users' network is not specified in any of this family of standards. IEEE Std 1451.1 provides a logical object model for this

Copyright © 2007 IEEE. All rights reserved.

interface. Other applications may also be used at the manufacturer's discretion. The communications interface between the NCAP and the TIMs is defined in the remaining members of the family of standards. Different manufacturers may build the NCAPs and TIMs, and if both comply with this standard, they should be interoperable.

This standard provides a description of the functions that are to be performed by a transducer interface module or TIM. Provisions are made for a high level of addressing that is independent of the physical medium-level and low-level protocols that are used to implement the communications. It defines the common characteristics for all devices that implement the transducer modules. The timing of the acquiring or processing of the data samples is described. Methods of grouping the outputs from multiple transducers within one TIM are defined. Common status words are also defined.

A standard set of commands are defined to facilitate the setup and control of the transducer modules as well as to read and write the data used by the system. Commands are also provided for reading and writing the TEDS that supply the system with the operating characteristics that are needed to use the transducer modules. A method of adding manufacturer unique commands is included.

In addition, this standard provides formats for the TEDS. Several TEDS are defined in the standard. Four of these TEDS are required, and the remaining TEDS are optional. Some TEDS are provided to allow the user to define information and to store it in the TEDS.

This standard provides areas that are "open to manufacturers." It should be noted that any use of these areas may compromise the "plug-and-play" potential of controllers and TIMs.

Notice to users

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: http://standards.ieee.org/reading/ieee/updates/errata/index.html. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims or determining whether any licensing terms or conditions are reasonable or non-discriminatory. Further information may be obtained from the IEEE Standards Association.

Copyright © 2007 IEEE. All rights reserved.

Participants

At the time this standard was submitted to the IEEE-SA Standards Board for approval, the Common Functionality and TEDS Working Group had the following membership:

Kang Lee, *Chair*
Victoria K. Sweetser, *Vice Chair*
Jay J. Nemeth-Johannes, *Secretary*
Lee H. Eccles, *Editor*
Jefferson B. Burch, *API Architect*

Pat Blakely
Murat Bog
Charles H. Jones
Edward Koch
Dan Maxwell
David B. Perrussel
Andrew Segal
Mallikarjun Shankar
Robert Sinclair
Eugene Song
Rich Valde
Larry Whipple
James J. Wiczer
Darold Wobschall

Other individuals who have contributed to this standard are as follows:

Thurston Brooks
Wayne Catlin
Baozhi Chen
Stephen Chen
Ryan Coleman
Ken Cornett
Phil Ellerbrock
Peter Flitner
Fernando GenKuong
Robert N. Johnson
Alice Law
Myung Lee
Larry Malchodi
Loren Rittle
Jay Warrior
Stan Woods

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

William J. Ackerman
Chris B. Bagge
Jefferson B. Burch
Juan C. Carreon
Danila Chernetsov
Keith Chow
Tommy P. Cooper
Matthew T. Davis
David B. Droste
Lee H. Eccles
Michael D. Geipel
Fernando Genkuong
Sergiu R. Goma
Patrick S. Gonia
Randall C. Groves
Ryusuke Hasegawa
Werner Hoelzl
Dennis Horwitz
Jens Hult
Robert N. Johnson
Charles H. Jones
Innocent Kamwa
Piotr Karocki
James C. Kemerling
Edward Koch
Charles A. Lennon, Jr.
William Lumpkins
G. L. Luri
Joseph R. Marshall, Jr.
Gary L. Michel
Yinghua Min
Georges F. Montillet
Jerry R. Murphy
Jay J. Nemeth-Johannes
Michael S. Newman
Chris L. Osterloh
Howard W. Penrose
David B. Perrussel
Vikram Punj
Bogdan Seliger
Mallikarjun Shankar
Thomas M. Siep
Matthew L. Smith
Victoria K. Sweetser
Leroy O. Thielman
Stephen C. Webb
James J. Wiczer
Ernesto Jorge Wiedenbrug
Darold Wobschall
Derek T. Woo
Eric V. Woods
Oren Yuen
Janusz Zalewski

Copyright © 2007 IEEE. All rights reserved.

When the IEEE-SA Standards Board approved this standard on 22 March 2007, it had the following membership:

Steve M. Mills, *Chair*
Robert M. Grow, *Vice Chair*
Donald F. Wright, *Past Chair*
Judith Gorman, *Secretary*

Richard DeBlasio	Richard H. Hulett	Narayanan Ramachandran
Alex Gelman	Hermann Koch	Greg Ratta
William R. Goldbach	Joseph L. Koepfinger*	Robby Robson
Arnold M. Greenspan	John D. Kulick	Anne-Marie Sahazizian
Joanna N. Guenin	David J. Law	Virginia C. Sulzberger
Julian Forster*	Glenn Parsons	Malcolm V. Thaden
Kenneth S. Hanus	Ronald C. Petersen	Richard L. Townsend
William B. Hopf	Tom A. Prevost	Howard L. Wolfman

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish K. Aggarwal, *NRC Representative*
Alan H. Cookson, *NIST Representative*

Jennie M. Steinhagen
IEEE Standards Program Manager, Document Development

Norma B. Davis
IEEE Standards Program Manager, Technical Program Development

Copyright © 2007 IEEE. All rights reserved.

Contents

1 Overview 1

1.1 Scope 3
1.2 Purpose 3
1.3 Conformance 3

2 Normative references....5

3 Definitions, acronyms, and abbreviations 6

3.1 Definitions 6
3.2 Acronyms and abbreviations 9

4 Data types....10

4.1 Unsigned octet integer....10
4.2 Unsigned 16 bit integer 10
4.3 Signed 32 bit integer....10
4.4 Unsigned 32 bit integer 10
4.5 Single-precision real....11
4.6 Double-precision real 11
4.7 String 11
4.8 Boolean....11
4.9 IEEE1451Dot0::Args::TimeRepresentation....12
4.10 Data types for associated applications....13
4.11 Physical Units....13
4.12 Universal unique identification 15
4.13 Arbitrary octet array 15
4.14 String array....16
4.15 Boolean array 16
4.16 Array of 8 bit signed integers 16
4.17 Array of 16 bit signed integers 16
4.18 Array of 32 bit signed integers 16
4.19 Array of 8 bit unsigned integers 17
4.20 Array of 16 bit unsigned integers 18
4.21 Array of 32 bit unsigned integers 18
4.22 Array of single-precision real numbers 18
4.23 Array of double-precision real numbers....18
4.24 Array of TimeDuration data types....18
4.25 Array of TimeInstance data types....19

5 Smart transducer functional specification 19

5.1 IEEE 1451 family reference model 19
5.2 Plug-and-play capability....23
5.3 Addresses....23
5.4 Common characteristics 25
5.5 Transducer Electronic Data Sheets....27
5.6 TransducerChannel type descriptions....31
5.7 Embedded TransducerChannels 34
5.8 TransducerChannel groups....34

Copyright © 2007 IEEE. All rights reserved.

5.9 TransducerChannel proxy 35
5.10 Attributes and operating modes 36
5.11 Triggering 41
5.12 Synchronization 48
5.13 Status 49
5.14 Service request logic 55
5.15 Hot-swap capability 56

6 Message structures 56

6.1 Data transmission order and bit significance 56
6.2 Command message structure 57
6.3 Reply messages 58
6.4 TIM initiated message structure 58

7 Commands 59

7.1 Standard commands 60
7.2 Manufacturer-defined commands 81

8 TEDS specification 81

8.1 General format for TEDS 81
8.2 Order of octets in numeric fields 83
8.3 TEDS identification header 83
8.4 Meta-TEDS 84
8.5 TransducerChannel TEDS 94
8.6 Calibration TEDS 119
8.7 Frequency Response TEDS 136
8.8 Transfer Function TEDS 139
8.9 Text-based TEDS 149
8.10 End User Application Specific TEDS 154
8.11 User's Transducer Name TEDS 155
8.12 Manufacturer-defined TEDS 157
8.13 PHY TEDS 158

9 Introduction to the IEEE 1451.0 API 158

9.1 API goals 159
9.2 API design decisions 160
9.3 IEEE1451Dot0 162

10 Transducer services API 173

10.1 IEEE1451Dot0::TransducerServices::TimDiscovery 173
10.2 IEEE1451Dot0::TransducerServices::TransducerAccess 175
10.3 IEEE1451Dot0::TransducerServices::TransducerManager 181
10.4 IEEE1451Dot0::TransducerServices::TedsManager 187
10.5 IEEE1451Dot0::TransducerServices::CommManager 190
10.6 IEEE1451Dot0::TransducerServices::AppCallback 191

Copyright © 2007 IEEE. All rights reserved.

11 Module Communications API 193

11.1 IEEE1451Dot0::ModuleCommunication::Comm 193
11.2 IEEE1451Dot0::ModuleCommunication::P2PComm 197
11.3 IEEE1451Dot0::ModuleCommunication::NetComm 201
11.4 IEEE1451Dot0::ModuleCommunication::Registration 210
11.5 IEEE1451Dot0::ModuleCommunication::P2PRegistration 212
11.6 IEEE1451Dot0::ModuleCommunication::NetRegistration 214
11.7 IEEE1451Dot0::ModuleCommunication::Receive 217
11.8 IEEE1451Dot0::ModuleCommunication::P2PReceive 217
11.9 IEEE1451Dot0::ModuleCommunication::NetReceive 218

12 HTTP protocol 220

12.1 IEEE 1451.0 HTTP API 221
12.2 Discovery API 224
12.3 Transducer access API 226
12.4 TEDS Manager API 232
12.5 Transducer Manager API 238

Annex A (informative) Bibliography 244

Annex B (informative) Guidance to Transducer Services Interface 246

Annex C (informative) Guidance to Module Communication Interface 240

Annex D (informative) XML Schema for Text-based TEDS 261

Annex E (informative) Example Meta-Identification TEDS 268

Annex F (informative) Example TransducerChannel Identification TEDS 280

Annex G (informative) Example Calibration Identification TEDS 282

Annex H (informative) Example Commands TEDS 284

Annex I (informative) Example Location and Title TEDS 287

Annex J (infomative) Example Units Extension TEDS 289

Annex K (informative) Examples of Physical Units 290

Annex L (informative) TEDS read and write protocols 296

Annex M (informative) Trigger logic configurations 298

Annex N (informative) Notation summary for IDL 303

Annex O (informative) TEDS implementation of a simple sensor 307

Copyright © 2007 IEEE. All rights reserved.

IEEE Standard for a Smart Transducer Interface for Sensors and Actuators—Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats

1. Overview

This standard introduces the concept of a transducer interface module (TIM) and a network capable application processor (NCAP) connected by a media specified by another member of the IEEE 1451 family of standards. A TIM is a module that contains the interface, signal conditioning, analog-to-digital and/or digital-to-analog conversion and, in many cases, the transducer. A TIM may range in complexity from a single sensor or actuator to units containing many transducers (sensors and actuators). An NCAP is the hardware and software that provides the gateway function between the TIMs and the user network or host processor. Another member of the standards family provides the communications interface between an NCAP or host processor and one or more TIMs. Three types of transducers are recognized by this standard. They are sensors, event sensors, and actuators.

A transducer is denoted "smart" in this context because of three features:

— It is described by a machine-readable Transducer Electronic Data Sheet (TEDS).

— The control and data associated with the transducer are digital.

— Triggering, status, and control are provided to support the proper functioning of the transducer.

An NCAP or a host processor controls a TIM by means of a digital interface medium. The NCAP mediates between the TIM and a higher level digital network and may provide local intelligence.

This standard defines an application program interface (API) for applications that provide communications between the users' network and the IEEE 1451.0 layer. An API is also provided between the IEEE 1451.0 layer and the underlying physical communications layers usually referred to in this standard as IEEE 1451.X. These API definitions are provided for systems that have visible interfaces. For TIMs and

Copyright © 2007 IEEE. All rights reserved.

NCAPs with a single set of hardware and software without regard to distinguishing separate interfaces between IEEE 1451.0 functionality and IEEE 1451.X functionality, the API is optional as long as the messages at visible interfaces conform to the rest of the standard. The definition of these APIs is to facilitate modular design to the extent that multiple suppliers can provide different functionality and yet have the various parts integrate seamlessly.

This standard defines TIMs that can be plugged into a system and can be used without having to add special drivers, profiles, or make any other changes to the system. This process is referred to as a "plug-and-play" operation. The primary features that enable a plug-and-play operation are the TEDS and the command set. A TIM may be added to or removed from an active IEEE 1451 physical layer with no more than a momentary impact on the data being transferred over the bus. "Hot swap" is the term used to refer to this feature.

This standard is organized as follows:

Clause 1: "Overview" provides the scope of this standard.

Clause 2: "Normative references" lists references to other standards and documents that are useful in applying this standard.

Clause 3: "Definitions, acronyms, and abbreviations" provides definitions that are either not found in other standards or have been modified for use with this standard.

Clause 4: "Data types" defines the data types used in the standard.

Clause 5: "Smart transducer functional specification" specifies the functions required of a TIM and of each TransducerChannel it comprises.

Clause 6: "Message structures" specifies the message structures that are used to encapsulate the information being passed between an NCAP and TIMs.

Clause 7: "Commands" provides the command syntax and the expected replies.

Clause 8: "TEDS specification" specifies the transducer electronic data sheet structure and content.

Clause 9: "Introduction to the IEEE 1451.0 API" gives the common features of the two APIs.

Clause 10: "Transducer services API" gives the API that an application would use to utilize this standard.

Clause 11: "Module communications API" gives the API that this standard would use to communicate to the TIMs using the communications features of a physical interface defined by another member of the IEEE 1451 family of standards.

Clause 12: "HTTP protocol" is used to transfer or convey information on the World Wide Web. It is intended to provide a simpler protocol than is currently supplied by IEEE Std 1451.1-1999.

Annex: A: "Bibliography" provides references for additional information about topics referred to in this document.

Annex B: "Guidance to Transducer Services Interface" gives examples of the use of this interface for measurement and control applications that interact with the IEEE 1451.0 layer using the Transducer Services interface.

Annex C: "Guidance to Module Communication Interface" gives additional guidance for the logical communication between the NCAP and TIMs or between TIMs using the Module Communication API.

Annex D: "XML Schema for Text-based TEDS" gives the basic schemas for the Text-based TEDS defined in this standard.

Annex E: "Example Meta-Identification TEDS" gives an example of a possible Meta-Identification TEDS.

Copyright © 2007 IEEE. All rights reserved.

Annex F: "Example TransducerChannel Identification TEDS" gives an example of a possible TransducerChannel Identification TEDS.

Annex G: "Example Calibration Identification TEDS" gives an example of a possible Calibration Identification TEDS.

Annex H: "Example Commands TEDS" gives an example of a possible Commands TEDS.

Annex I: "Example Location and Title TEDS" gives an example of a possible Location and Title TEDS.

Annex J: "Example Units Extension TEDS" gives an example of a possible Units Extension TEDS.

Annex K: "Examples of Physical Units" gives a series of examples of implementations of Physical Units using the representation specified in this standard.

Annex L: "TEDS read and write protocols" describes processes that may be used to write or read the TEDS.

Annex M: "Trigger logic configurations" shows some possible configurations of the trigger logic allowed in this standard.

Annex N: "Notation summary for IDL" is intended to give guidance on the use of IDL notation in this standard.

Annex O: "TEDS implementation of a simple sensor" is an example of a simple sensor implemented using the structures defined in this standard.

1.1 Scope

This project develops a set of common functionality for the family of IEEE 1451 smart transducer interface standards. This functionality is independent of the physical communications media. It includes the basic functions required to control and manage smart transducers, common communications protocols, and media-independent TEDS formats. It defines a set of implementation-independent APIs. This project does not specify signal conditioning and conversion, physical media, or how the TEDS data are used in applications.

1.2 Purpose

There are currently three approved and three proposed smart transducer interface standards in the IEEE 1451 family of standards. They all share certain characteristics, but no common set of functions, communications protocols, and TEDS formats provides interoperability among these standards. This standard will provide that commonality and will simplify the creation of future standards for different physical layers that are interoperable within the family.

1.3 Conformance

The philosophy underlying the conformance requirements of this subclause is to provide the structure necessary to raise the level of interoperability of transducers and systems built to this standard, while leaving open opportunity for continued technical improvement and differentiation.

TIM implementation shall be deemed in conformance with this standard provided the following requirements are met:

a) The TIM supports the required functional specifications identified in Clause 5.

Copyright © 2007 IEEE. All rights reserved.

5.10.1.1 Trigger-initiated

In the trigger-initiated sampling mode, a sensor shall begin acquiring a data set immediately upon receipt of a trigger. An actuator shall start outputting a data set immediately upon receipt of a trigger. Sample processing continues until all samples in the data set are processed at a rate determined by the TIM and then stops.

5.10.1.2 Free-running without pre-trigger

In the free-running sampling mode, a sensor is measuring some physical parameter autonomously and continuously when in the transducer operating state. The data being acquired and converted are discarded until a trigger is received. Once a trigger has been received, the next sample converted is stored in the TransducerChannel or TIM as the first word in the data set. Consecutive samples are stored until the entire data set is completed. At this point, the TransducerChannel returns to discarding samples until the next trigger is received. The sampling operation of a TransducerChannel when in the transducer idle state is at the discretion of the manufacturer.

An actuator operating in the free-running sampling mode shall apply the data set received before the trigger in accordance with its End-of-data-set Operating mode.

5.10.1.3 Free-running with pre-trigger

In the free-running sampling mode, a sensor is measuring some physical parameter autonomously and continuously when in the transducer operating state. The data being acquired and converted is stored until a trigger is received or the pre-trigger count (see 7.1.2.2 and 8.5.2.32) is reached. After the number of samples that have been stored reaches the pre-trigger count, the next sample acquired shall cause the oldest sample to be discarded and the new sample shall be stored. Once a trigger has been received, the next sample converted is stored in the TIM as the next word in the data set. Consecutive samples are stored in the TIM until the entire data set is completed. The data set shall be complete when the number of samples equal to the data set size minus the pre-trigger count has been acquired after the trigger is received. The behavior after the data set is complete is described in 5.10.1.4 or 5.10.1.5. The response to additional triggers before the operation started by the first trigger is complete is described in 5.11.5.

An actuator may not be operated in the free running with pre-trigger mode.

5.10.1.4 Free-running with pre-trigger without buffers enabled

When the data set is complete the TransducerChannel returns to discarding samples until the next trigger is received or the data set is read. After the data set is read, the TransducerChannel shall start storing samples again while waiting for another trigger. In this mode, the data set may only be read once. Subsequent reads before the next trigger will return 0 octets. The sampling operation of a TransducerChannel when in the transducer idle state is at the discretion of the manufacturer.

5.10.1.5 Free-running with pre-trigger and buffers enabled

When the data set is complete, the TransducerChannel shall switch to the next empty buffer and start acquiring data samples for the next data set. If no empty buffers are available, it shall discard any samples acquired until a buffer becomes available. A buffer shall be considered available after it has been read. The sampling operation of a TransducerChannel when in the transducer idle state is at the discretion of the manufacturer.

Copyright © 2007 IEEE. All rights reserved.

5.10.1.6 Continuous sampling mode

In the continuous sampling mode, a sensor shall begin to acquire samples and store them in one of its buffers when it receives an initial trigger. The operation shall be similar to the free-running without pre-trigger mode described in 5.10.1.2 except that the TransducerChannel does not stop when a data set is acquired but switches to the next available buffer and continues to acquire data. Operating in this mode requires that the sensor have multiple buffers available for storing the data samples. Once all buffers are full, the data in the oldest buffer shall be discarded regardless of whether it has been transmitted to the NCAP and that buffer shall be used to store the data being acquired. If the streaming at a fixed interval transmission mode described in 5.10.2.3 is being used, the sensor shall switch to an empty buffer at the start of a new transmission interval, regardless of whether the current buffer is full. However, if the number of samples acquired within a transmission interval is greater than the Max Data Repetitions from the TransducerChannel TEDS (see 8.5.2.28), then the data set shall be truncated at the Max Data Repetitions and the TransducerChannel Missed data or event (see 5.13.5) bit shall be set.

After an initial trigger, an event sensor operated in the continuous sampling mode shall detect a change in state at its input, store that state change for transmission, and continue to look for additional changes in its input state. This requires that the event sensor have multiple buffers and use them like a sensor uses buffers. If the streaming at a fixed interval transmission mode is being used, the TransducerChannel shall switch to an empty buffer at the start of a new transmission interval, regardless of whether the current buffer is full.

In the continuous sampling mode, an actuator shall apply all data in its current buffer when the first trigger is received at a rate controlled by the TransducerChannel. Once all data in that buffer has been applied, it shall switch to the oldest filled buffer and continue applying the data. If another filled buffer is not available, the actuator shall take the action described in 5.10.4 and controlled by the setting of the End-of-data-set operations attribute as described in 8.5.2.48. If this action is to “recirculate,” the unit shall not look for a new filled buffer until it completes reapplying the current buffer. If that action is to “hold,” the unit shall switch to the new buffer as soon as it has been received and stored in memory. If an attempt is made to write data to this TransducerChannel and no empty buffers are available, the incoming data shall be ignored and the TransducerChannel Missed data or event bit shall be set.

5.10.1.7 Immediate operation

A sensor in this sampling mode will immediately acquire a data set and transmit it as a response to a Read TransducerChannel data-set segment command. The receipt of the Read TransducerChannel data-set segment command will function as a trigger.

An actuator in this sampling mode will immediately apply the data-set received from a write TransducerChannel data-set segment command. The receipt of the write TransducerChannel data-set segment command will function as a trigger.

5.10.2 Data transmission mode

Three data transmission modes are defined in this standard as shown in Table 7.

Copyright © 2007 IEEE. All rights reserved.

Table 7—Data transmission mode

Enumeration	Argument name	Data transmission mode	Description
0	XmitMode.reserved[0]	Reserved	
1	XmitMode.OnCommand	Only when commanded	A TIM shall transmit a data set only in response to a Read TransducerChannel data-set segment command (see 7.1.3.1).
2	XmitMode.BufferFull	Streaming when a buffer is full	Data are transmitted as soon as a buffer is full without waiting for the NCAP to issue a Read TransducerChannel data-set segment command (see 7.1.3.1).
3	XmitMode.Interval	Streaming at a fixed interval	Data buffers are transmitted at a fixed interval. The TransducerChannel shall stop using the current buffer regardless of how much data are in it, shall begin storing data in another buffer, and shall transmit the buffer it was using without waiting for a Read TransducerChannel data-set segment command (see 7.1.3.1).
4–127	XmitMode.reserved[N] 4≤N≤127	Reserved	
128–255	XmitMode.open[N] 128≤N≤255	Open to manufacturers	

5.10.2.1 Only when commanded mode

When in the Only when commanded mode, a TransducerChannel shall only transmit a data set in response to a Read TransducerChannel data-set segment command (see 7.1.3.1).

5.10.2.2 Streaming when a buffer is full mode

When in the Streaming when a buffer is full mode, a sensor or event sensor shall transmit a data set as soon as a complete data set has been acquired. An actuator does not stream data, so it may not be operated in this mode. The equivalent operating mode for an actuator is the Continuous sampling mode described in 5.10.1.6.

5.10.2.3 Streaming at a fixed interval mode

When in the Streaming at a fixed interval mode, a TransducerChannel shall transmit a data set or partial data set at regular, fixed intervals. When operated in this mode, the TransducerChannel data repetitions shall not be used to determine the number of samples in a data set. The number of samples in a data set shall be determined from the sampling rate and the periodic transmission interval. However, if the number of samples acquired within a transmission interval is greater than the Maximum data repetitions from the TransducerChannel TEDS (see 8.5.2.28), then the data set shall be truncated at the Maximum data repetitions and the TransducerChannel hardware error bit (see 5.13.7) shall be set.

The method of defining the sample interval is defined in the appropriate transmission media standard and is not covered in this standard.

5.10.3 Buffered operation mode

A sensor or actuator may be capable of being operated in buffered or non-buffered modes. In the buffered mode, one buffer is available to be read from a sensor or applied to an actuator output, whereas other buffers are available to be filled. A characteristic of a TransducerChannel operating in the buffered mode is

Copyright © 2007 IEEE. All rights reserved.

that the data available to be read or applied are always the data that were available in a buffer before the previous trigger. In the non-buffered mode, only a single buffer is available to store a data set. The buffering capabilities of a given TransducerChannel are described by the Buffered attribute in the TransducerChannel TEDS. See 8.5.2.47 for the details of this attribute.

A sensor that is in the transducer idle state and before it is placed into the transducer operating state for the first time after initialization shall return 0 data octets to any Read TransducerChannel data-set segment command (see 7.1.3.1). If a buffered sensor has buffers that are full or partially full when it is returned to the transducer idle state, it shall return the contents of the oldest buffer on receipt of a Read TransducerChannel data-set segment command. Upon the receipt of subsequent Read TransducerChannel data-set segment commands, the contents of the remaining buffers shall be returned. After the contents of each buffer that had unread data in it when the TransducerChannel idle command (see 7.1.4.2) was received have been read, the TransducerChannel or TransducerChannel proxy and shall return 0 data octets to subsequent Read TransducerChannel data-set segment commands.

All buffers shall be cleared when the TransducerChannel transitions from the idle to the transducer operating state.

5.10.4 End-of-data-set operation mode

This mode only applies to actuators. The End-of-data-set operation attribute in the TransducerChannel TEDS (8.5.2.48) defines the possible operations that an actuator may perform when it reaches the end of a data set. Two possible operating modes are described in 5.10.4.1 and 5.10.4.2.

NOTE—These operation modes describe ways to allow actuators to smoothly transition from one data set to another. Although this smooth transition is desirable in most cases, there are instances when a rapid transition to another condition is required, i.e., emergency shutdown. For these cases, a second actuator TransducerChannel in the same TIM may be used that executes the transition to the new condition. The mechanism for switching control of the output from one actuator to the other is internal to the TIM and not covered in the standard.

5.10.4.1 Hold

An actuator operating in the hold mode shall use all samples in a data set and then shall continue to use the last sample in the data set until a new trigger is received.

NOTE—The hold mode is suited to an actuator containing a valve or other mechanical positioner. This type seems to also fit bistable-output logic-type output devices but is also frequently associated with the trigger-initiated sampling mode.

5.10.4.2 Recirculate

An actuator operating in the recirculate mode shall apply all samples in a data set to the output, then shall return to the beginning of the data set, and shall repeat the application of that data set until another trigger is received or the TransducerChannel is disabled. When returning to the beginning of a data set or to a new data set, the appropriate sample interval shall be maintained. When another trigger is received, the TransducerChannel shall switch to the new data set when it reaches the end of the current data set.

5.10.5 Streaming operation

Streaming operation is achieved by combining the continuous sampling mode (see 5.10.1.6) with either the streaming when a buffer is full mode (see 5.10.2.2) or the streaming at a fixed interval mode (see 5.10.2.3). A sensor or event sensor in streaming operation shall acquire data and transmit it without further

Copyright © 2007 IEEE. All rights reserved.

commands from the NCAP. A streaming actuator shall apply the data received without a trigger command being received for each data set.

5.10.6 Edge-to-report operating mode

This operating mode applies to event sensors only.

An event sensor may be capable of operating in one of three modes. In the falling transitions mode, they report all falling transitions. In the rising transition mode, it reports all rising transitions. See 5.6.2.3 and 5.6.2.4 for the definition of falling and rising transitions. In the all-transitions mode, it reports both rising and falling transitions. The Edge-to-report attribute field in the TransducerChannel TEDS defines the capabilities of the event sensor with regard to this operating mode. When defaults are included, there are six possible values to this attribute. See 8.5.2.50 for details.

5.10.7 Actuator-halt mode

This attribute applies to actuators only. This mode defines what an actuator does when a TransducerChannel idle command is received. The Actuator-halt attribute in the TransducerChannel TEDS (see 8.5.2.51) defines the possible operations.

5.10.7.1 Halt immediate

The actuator shall hold the current state of its output until it is returned to the transducer operating state or put into the sleep state.

5.10.7.2 Halt at the end of the data set

The actuator shall finish applying the current data set and then hold the last output value of its output until it is returned to the transducer operating state or put into the sleep state.

5.10.7.3 Ramp to a predefined state

The actuator shall use a manufacturer-defined process to ramp the output to a predefined state.

5.11 Triggering

A trigger is a signal applied to a TransducerChannel or to a set of TransducerChannels to cause them to take a particular action. The state diagram in Figure 7 shows the triggering states for a sensor, and Figure 8 shows the triggering states of an actuator.

5.11.1 Sensor triggering

Movement from state to state is dependent on the commands received as defined in Clause 7, the state of the TransducerChannel (see Figure 3), the state of the TIM (see Figure 4), events occurring, and data set full/empty status.

Copyright © 2007 IEEE. All rights reserved.

If the TIM is taken out of the Active state, the sensor shall immediately return to the Sensor Trigger Initialize State. If the TransducerChannel is taken out of the Operating State, then the sensor shall immediately return to the Sensor Trigger Initialize State. If a Reset Command is received, then the sensor shall immediately return to the Sensor Trigger Initialize State.

5.11.2 Actuator triggering

Movement from state to state of the actuator triggering is dependent on the commands received as defined in Clause 7, the state of the TransducerChannel (see Figure 3), the state of the TIM (see Figure 4), events occurring, and the data set full/empty status.

If the TIM is taken out of the Active state and the actuator is in the Transverse Data-set state, then the actuator shall return to the Actuator Trigger Initialize State by way of the Actuator Halt State. If the Transducer is not in the Operating State and the actuator is in the Transverse Data-set state, then the actuator shall return to the Actuator Trigger Initialize State by way of the Actuator Halt State. If a Reset Command is received and the actuator is in the Transverse Data-set state, then the actuator shall return to the Actuator Trigger Initialize State by way of the Actuator Halt State. The Actuator Halt State uses the Actuator Halt Operating Mode (see 5.10.7) to process the data set.

If a Trigger is received while the actuator channel is in the Transverse Data-set state (see Figure 8) and the End of Data-set Operation mode (see 5.10.4) is set to recirculate and a new data set has been received, then the TransducerChannel shall switch to the new data set when it reaches the end of the current data set.

The Actuator Halt Mode state shall use the argument from the Actuator Halt Operating Mode command to exit the data set cleanly.

Several methods are recognized by this standard to initiate a trigger. They are explicit triggers commanded by the NCAP, an access of the TIM by the NCAP and events within a TIM that may be used as triggers, and a trigger command that may be initiated by a NCAP-enabled TIM when an event occurs within the TIM.

Copyright © 2007 IEEE. All rights reserved.

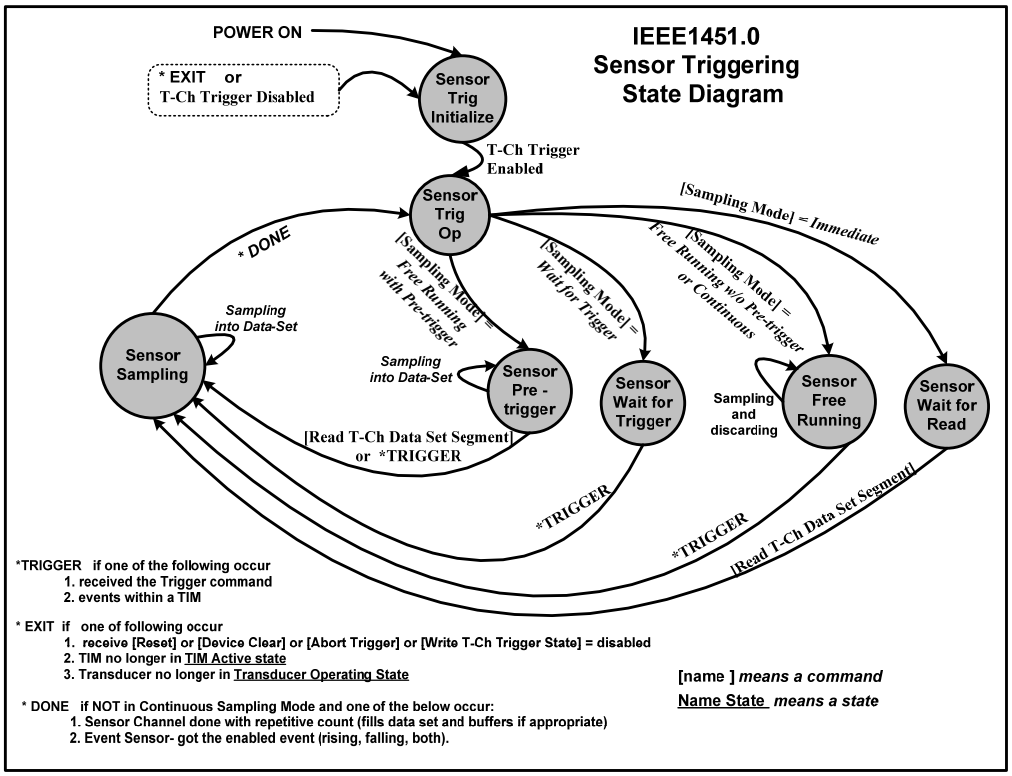


Figure 7—Sensor trigger states

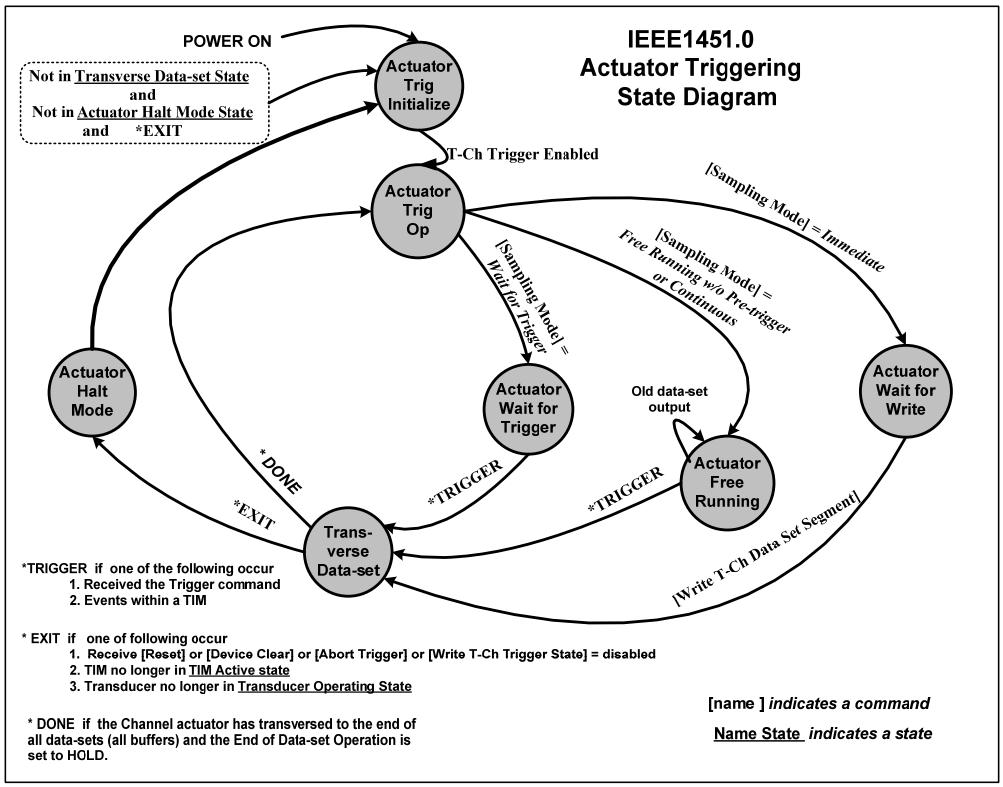


Figure 8—Actuator trigger states

Copyright © 2007 IEEE. All rights reserved.

5.11.2.1 Trigger commands

Trigger commands are sent from the NCAP to one or more TransducerChannels on a transducer common medium. A *trigger* command is defined in 7.1.3.3. A trigger command may be addressed to any of the following:

— TransducerChannel
— TransducerChannel Proxy
— TIM
— AddressGroup
— Global address

A trigger command addressed to a specific TransducerChannel applies to one TransducerChannel on one TIM.

A TransducerChannel proxy is an addressable resource within a single TIM (see 5.9) that is capable of "representing" one or more TransducerChannels within that TIM. A trigger command addressed to a TransducerChannel proxy triggers each TransducerChannel represented by that TransducerChannel proxy.

A trigger command addressed to a TIM triggers all trigger enabled TransducerChannels that are implemented on that TIM.

The system user may define AddressGroups (see 5.3). When the system is set up, each TransducerChannel to be included in an AddressGroup is programmed to respond to the AddressGroup identifier for that AddressGroup. A trigger command issued to that AddressGroup triggers all members of that AddressGroup. If the trigger is issued to multiple TIMs using the techniques described in 10.2.3 and 10.2.4, it will trigger all TransducerChannels on all TIMs to which the command is sent.

A global trigger applies to all trigger-enabled TransducerChannels in a TIM. The system issues a global trigger by issuing a trigger command to the global address. If the trigger is issued to multiple TIMs using the techniques described in 10.2.3 and 10.2.4, it will trigger all TransducerChannels on all TIMs to which the command is sent. Regardless of the address mode, a TransducerChannel shall only honor the trigger when triggering is enabled for that TransducerChannel.

5.11.2.2 Events used as triggers

Events may be used as triggers for other TransducerChannels within the same TIM directly. Events of this nature are not Trigger commands, but they serve the same function. An event used as a trigger may be formally implemented as an Event Sensor. An event may cause an event sensor to transmit a trigger message to other TIMs on the same communications media.

NOTE—For event sensors used as triggers in systems where it is important for the system to be able to determine the time of the event, it is necessary to augment the event sensor with a time interval sensor or TimeInstance sensor to be able to determine the time of the event accurately.

5.11.3 Nominal trigger logic

A functional block diagram of the logic within a TIM that is related to triggering is shown in Figure 9.

Copyright © 2007 IEEE. All rights reserved.

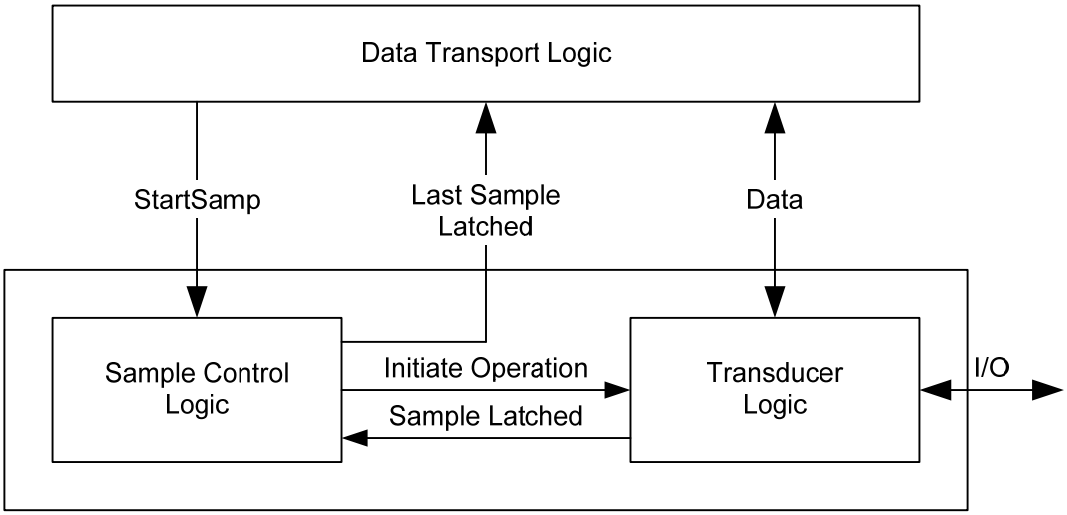


Figure 9—Simple TransducerChannel functional blocks

The Data Transport Logic responds to trigger commands. Upon receipt of a trigger command, provided that the TransducerChannel trigger is enabled, the Data Transport Logic issues a StartSamp signal.

The StartSamp signal shall cause a traditional sensor to acquire a new data set. For the simplest sensors, this may mean a digital latch is updated or an analog-to-digital converter begins a conversion. More complex sensors may take multiple samples per trigger, spaced apart incrementally in time or some other dimension.

The StartSamp signal shall cause an event sensor to begin monitoring the event of interest. The occurrence of the event shall cause the Sample Control logic to generate the Last Sample Latched signal.

The StartSamp signal shall cause an actuator TransducerChannel to apply a data set to its output. For the simplest actuators, this may mean a digital latch is updated or a digital-to-analog converter begins a conversion. More complex actuators may apply multiple samples per trigger, spaced apart incrementally in time or some other dimension.

The Sample Control logic is responsible for orchestrating the sequence of operations needed to acquire or apply a data set. The Transducer Logic block represents all signal conditioning, data conversion, and buffering logic. The Initiate Operation signal is used to initiate the process of data conversion on a single sample, and the Sample Latched signal indicates that a sample has been acquired or output.

For TransducerChannels implemented under the Nominal Triggering model, the determination of the time at which the sample was latched becomes the sole responsibility of the NCAP or host processor based on information provided by the TIM. There are two variations. The method used for determining T_N shall be specified in the Source for the time of sample field in the TransducerChannel TEDS (see 8.5.2.40).

This time calculation gives the time of the last sample in the data set. The following calculation describes how to calculate the time of any other sample in the data set.

$$T_i = T_N - (N - i)t_{si} \qquad \text{for } i = 1 \text{ to } N \tag{1}$$

where

N is the number of samples in the data set
T_i is the time of sample i

Copyright © 2007 IEEE. All rights reserved.

T_N is the time of the last sample in the data set
t_{si} is the sample interval

When the enumerated value of Source for the time of sample indicates that the delay between application of trigger and the subsequent latching of the sample is constant and the TIM lacks embedded hardware resources to support time tagging, the time of sample shall be computed from the time at which the trigger command was issued by adding an adjustment equal to the Incoming propagation delay through the data transport logic field of the TransducerChannel TEDS. This method neglects the effects of propagation delays over the transport media.

Calculation:

$$T_1 = T_{tm} + t_{pdi} \tag{2}$$

where

T_{tm} is the time of day at which the trigger command was sent by the NCAP
t_{pdi} is the value of the Incoming propagation delay through the data transport logic field found in the TransducerChannel TEDS
T_1 is the time of day when the first sample in the data set was latched

This time calculation gives the time of the first sample in the data set. The following calculation describes how to calculate the time of any other sample in the data set:

$$T_i = T_m + i * t_{si} \qquad \text{for } i = 1 \text{ to } N \tag{3}$$

where

N is the number of samples in the data set
T_i is the time of sample i
T_m is the time of the first sample in the data set
t_{si} is the sample interval

For other possible configurations of the trigger logic, see Annex L.

5.11.4 Trigger logic based on event recognition

The functional block diagram in Figure 10 depicts a TransducerChannel that is triggered by an event sensor. An event sensor may have additional embedded actuators to implement features such as a programmable analog threshold and hysteresis or a programmable digital pattern.

Copyright © 2007 IEEE. All rights reserved.

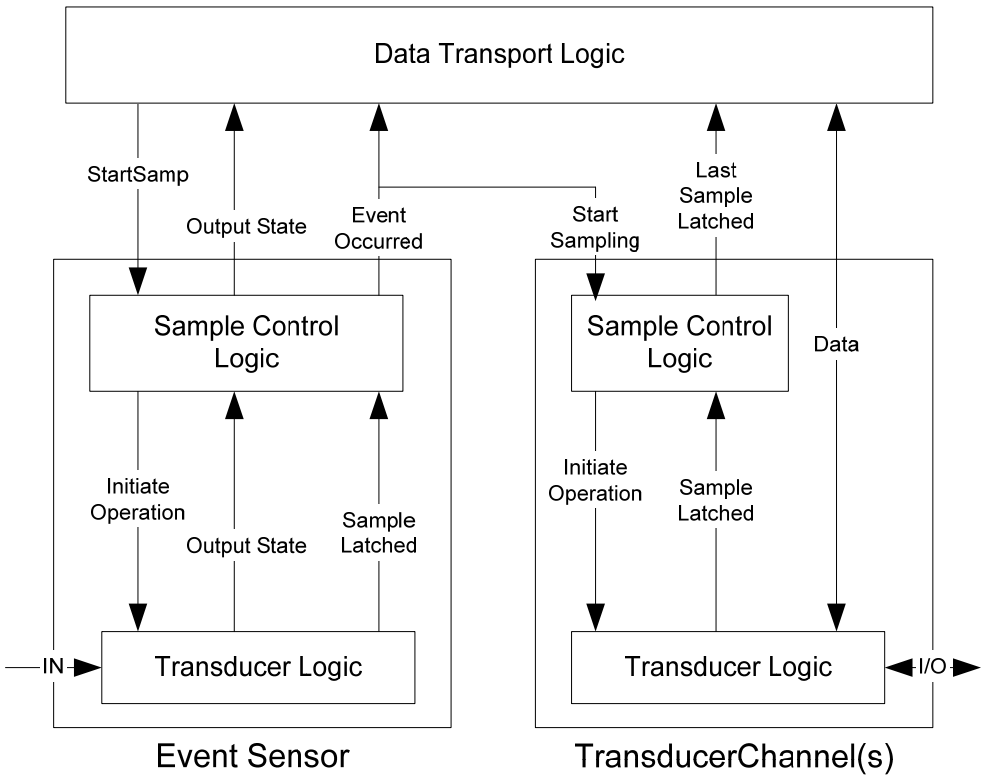


Figure 10—Event sensor output used as a trigger

The event sensor has a special hardware relationship to one or more TransducerChannels. This shall be identified as a ChannelGroup in the Meta-TEDS. A trigger command issued to the event sensor causes the event sensor to begin monitoring. The occurrence of the event may also be used to trigger the associated TransducerChannel(s).

The TransducerChannel may not honor a trigger command addressed to the TransducerChannels so configured.

5.11.5 Over-triggering a TransducerChannel

Irrespective of the time needed to read the TransducerChannel data, the NCAP is expected to wait for at least the duration of the TransducerChannel sampling period between successive triggers. The TransducerChannel sampling period is in the TransducerChannel TEDS (see 8.5.2.36).

The term "over-triggering" refers to issuing a second trigger before the TransducerChannel sampling period has expired. If this happens, the TransducerChannel may not have completely processed the previous trigger. Table 8 lists the response to over-triggering based on the operating mode of the TransducerChannel as described in 5.10.

Table 8—Response to over-triggering

Operating mode		**Response to the trigger**
Streaming	**Buffered**	
Yes	Don't care	Ignore all except the first trigger
No	Yes	Treat as a pre-trigger
No	No	Ignore triggers until previous operation complete

Copyright © 2007 IEEE. All rights reserved.

When in the non-streaming buffered mode, the unit treats a trigger received before all samples in a data set have been latched as a pre-trigger. In other words, it accepts the trigger as a command to start on another buffer when it completes the current buffer in such a way that all samples are acquired or applied while maintaining a uniform sample interval. If more than one extra trigger is received while processing a buffer, they are ignored. If this occurs, the time of a sample cannot be derived from the time that a trigger was sent. In all cases where a trigger command is ignored, the Command Rejected Bit (see 5.13.4) in the appropriate status register shall be set.

5.12 Synchronization

A fundamental requirement for the large-scale acquisition of data from distributed multipoint sensor arrays is the inclusion of an intrinsic capability for the system to provide time-synchronous triggering of the various data converters. The individual sampling of the spatially distributed sensors needs to be coordinated within rather precise time intervals to assure that the array-wide data points are acquired at essentially the same instant, and thus, they represent an accurate temporal "snapshot" of the measured parameters. Several features are built into this standard that are intended to aid in accomplishing this end. Subclauses 5.12.1 and 5.12.2 describe these features.

5.12.1 Methods of obtaining synchronicity

The basic tool that is used to obtain synchronization is the trigger. Since triggers may be addressed to groups of TransducerChannels, the members of the group are sampled synchronously within the limits of the propagation time and the variation of the delay between the receipt of the trigger by the TIM and its application in the TransducerChannel.

5.12.1.1 Using triggers

The AddressGroup and global triggers, as described in 5.3, are the simplest way of achieving synchronous sampling of the data. If the synchronous sampling is to be achieved across multiple TIMs, then a method of addressing multiple TIMs as described in 10.2.3 and 10.2.4 is required. The timing accuracy that can be achieved using this method is a function of the underlying communications media and protocol. This method works well with TransducerChannels that acquire or process a single sample for each trigger but is less effective with TransducerChannels that have data sets that contain more than one sample. It also has limitations when used over long cables or with extremely tight synchronization requirements.

5.12.1.2 Using a delayed trigger

The problem of different delays associated with different lengths of cable between the NCAP and the TIMs and different delays within the TIM may be addressed by adding the delayed trigger capability within the TIMs. This process is described in more detail in Annex L. To make this feature of the TIMs useful, this requires that the NCAP have a method of determining the delay associated with each TIM. For this method to be useful, the standard for the physical transmission media should provide a method for determining the transmission delay.

5.12.1.3 Using a synchronization signal

The synchronization signal provides a synchronous clock that is the same frequency for all TIMs on the common medium. The use of the synchronization signal makes the use of AddressGroup or global triggers with TransducerChannels that have a data set size greater than one more useful. However, it does not directly address the different delays associated with long cables.

Copyright © 2007 IEEE. All rights reserved.

5.12.2 Synchronization signal

In some standards in the family, the NCAP may generate a synchronization clock and transmit it on the transducer interface media. The TIMs may receive this clock, buffer it, and use it to generate the appropriate clocks within the TIM. The details of how this clock is transmitted by the NCAP to the TIMs are given in the appropriate standard in the IEEE 1451 family of standards.

5.13 Status

As shown in Figure 11 there are two types of status registers. Both are 32 bits wide. One register is called the Condition register. It may be read using the Read Status-Condition Register command (see 7.1.1.9). It contains the current state of the attributes being reported. The second register is the event register, and it is true if the condition register has been true since the last time the status-event register was cleared. It may be read using the Read Status-Event Register command (see 7.1.1.8). Some bits in the status-event register represent actual events, such as command errors. In this case, the condition register should always be zero for that bit, as the model would show that the condition occurred and was immediately cleared.

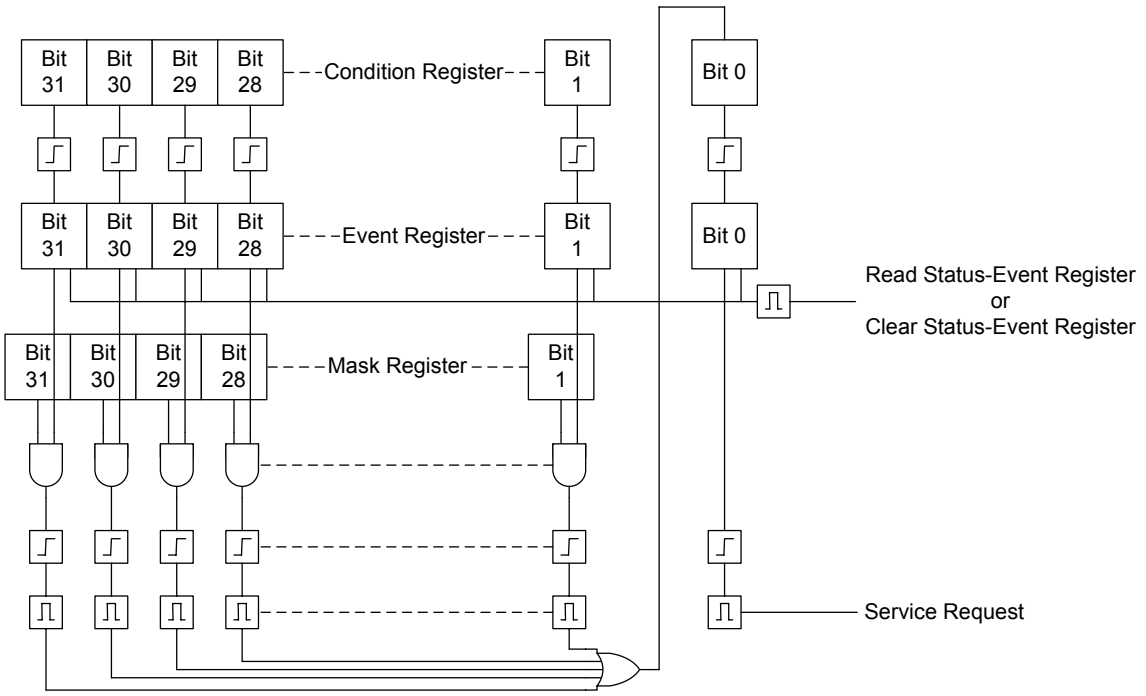


Figure 11—Status message generation logic

Both Status registers shall be implemented for the TIM and for each implemented TransducerChannel in a TIM. The returned status-event for the TIM represents the state of the TIM as a whole. In many cases, a bit in the TIM status-event represents the logical *OR* of corresponding bits in all implemented TransducerChannels.

Each bit in the status-event register represents the presence or absence of a particular event occurrence. A one in the appropriate bit position shall represent the presence of a condition or that an event has occurred since status was last read or cleared.

Copyright © 2007 IEEE. All rights reserved.

Each status-event register is used in conjunction with an associated service request mask (see 5.14.1) to control which status bits will be used to generate a service request.

Status bits defined by this standard are tabulated in Table 9. Some status bits are *reserved* for future versions of this standard. Some bits are optional, and the TIM manufacturer may choose not to implement them. Bits designated as "open to manufacturers" may be used to report conditions not represented by bits that are already defined. New bit definitions in the TIM status-event registers shall reflect conditions within the TIM as a whole. Status bits labeled "reserved," and unimplemented "optional" and "open to manufacturers" status bits, shall be reported as zero when read.

Table 9—Status bits

Bit	TIM status bits	TransducerChannel status bits	Required/ optional
0	Service request	Service request	Required
1	TEDS changed	TEDS changed	Optional
2	Invalid command	Reserved	Required
3	Command rejected	Command rejected	Required
4	Missed data or event	Missed data or event	See 5.13.4
5	Data/event	Data/event	See 5.13.5
6	Hardware error	Hardware error	See 5.13.6
7	Not operational	Not operational	Required
8	Protocol error	Reserved	Required
9	Data available/data processed	Data available/data processed	Optional
10	Busy	Busy	Optional
11	Failed calibration	Failed calibration	Optional
12	Failed self-test	Failed self-test	Optional
13	Data over range or under range	Data over range or under range	Optional
14	Corrections disabled	Corrections disabled	Optional
15	Consumables exhausted	Consumables exhausted	Optional
16	Reserved	Not-the-first-read-of-this-data-set	Required
17–23	Reserved	Reserved	—
24–31	Open to manufacturers	Open to manufacturers	—

Bits in any status-event register may be cleared in one of several ways as shown in the following list:

Summary bits shall be cleared immediately when the underlying status-event register has been read. Any bit in the TIM status-event register that is defined as the *OR* of the corresponding bit in the TransducerChannel status-event registers shall be cleared in this manner.

Other bits shall be cleared when the condition they report goes away *and* a status-event register read is performed or receipt of the clear status command. They shall not be cleared on receipt of a device clear protocol.

Status bits may also be changed by a change in operating states. The operating states are described in 5.4.1.

During power-on or reset initialization, data transport may be held off longer than the hold-off times specified in the TEDS. The TIM shall assure that reads of status registers return an accurate representation of the TIM's state. The TIM shall hold off a read of any status register for which this is not the case.

The status registers shall be accessed by the read status-event register command and the read status-condition register command defined in 7.1.1.8 and 7.1.1.9. The returned status shall be 4 octets wide. In Table 9, the column "TIM Status-Event Bits" defines the information that is returned if the TIM is addressed. If a TransducerChannel is addressed, then the bits are as defined under the column titled "TransducerChannel Status-Event Bits."

Copyright © 2007 IEEE. All rights reserved.

The status-event register may also be sent via a TIM initiated message (see 6.4) if the mask has been set for the status-event register bit that has been asserted and the Status-event protocol has been enabled using the write status-event protocol state command (see 7.1.1.11). When this protocol is enabled, a stream shall be initiated that will send the 32 bit status register any time the service request bit is asserted. If a channel is requesting service, the channel register shall be sent, and if the TIM itself is requesting service, the TIM status register shall be sent. The status information sent via the Status-event Protocol shall be identical to the information returned from a read status-event register command.

5.13.1 Service request bit

The TransducerChannel service request bit of any TransducerChannel shall be set when that TransducerChannel is requesting service and shall be cleared when read, when a status protocol message is sent, or when a clear status-event register command (see 7.1.1.10) is sent to that channel. Service Request masks are used to define conditions for which a TransducerChannel requests service.

See 5.14.1 for a description of the service request mask.

The TIM service request bit shall be set whenever the TIM is requesting service, as defined by the TIM masks. It shall be cleared when read, when a status protocol message is sent, or when a Clear Transducer Status-Event Register command is sent. It shall not be cleared when a device clear protocol is received.

A TransducerChannel service request bit shall be implemented for each TransducerChannel in a TIM. A TIM service request bit shall be implemented for each TIM.

The service request bit will be cleared by a change of operating state if the status bit that caused it to be set is cleared.

The service request bit shall be evaluated at power on and shall be asserted if any enabled status bit is asserted. It shall specifically be asserted if the power on status bit is enabled.

5.13.2 TEDS changed bit

The TIM Teds changed bit shall be set whenever the TIM changes the contents of an adaptive TEDS. The TransducerChannel Teds changed bit shall be set whenever the TransducerChannel changes the contents of an adaptive TEDS. It shall be cleared when read.

This bit is not affected by a change of transducer operating state.

5.13.3 Invalid command bit

The TIM invalid command bit shall be set whenever the TIM detects an unimplemented command or a read or write to an unimplemented function. It shall be cleared when read.

This bit is not affected by a change of transducer operating state.

5.13.4 Command rejected bit

The Command rejected bit shall be set whenever the TIM detects a valid command that cannot be executed because of the current mode of operation of the TIM or TransducerChannel. It shall also be set if an argument to the command is not acceptable for any reason. It shall be cleared when read.

Copyright © 2007 IEEE. All rights reserved.

The Command rejected status bit is mandatory and shall be implemented for the TIM and all TransducerChannels.

This bit is not affected by a change of operating state.

5.13.5 Missed data or event bit

The TransducerChannel missed data or event bit shall be set coincident with a data sample time of an event sensor or sensor with the sampling mode set to free-running if the TransducerChannel is in the transducer operating state but is not currently triggered. The exception to the above is that this bit shall not be set before the first trigger on any such TransducerChannel after the bit has been cleared. This bit shall be cleared when read.

The TIM missed data or event bit shall be set when any TransducerChannel missed data or event bit is set.

This status bit shall be implemented for any event sensor or any sensor capable of operating with the sampling mode set to free-running. It shall also be implemented in any TIM that contains such a sensor or event sensor.

The TransducerChannel missed data or event bit shall be cleared when the TransducerChannel transitions from the halted to the transducer operating state. The operating states are described in 5.4.1.

NOTE—While the TransducerChannel is not acquiring or consuming data in the transducer idle state, the bit shall remain set until it is read or otherwise cleared if it was set when the TransducerChannel transitioned from the transducer operating state to the transducer idle state.

5.13.6 Data/event bit

The TransducerChannel data/event bit shall be set at the data sample time for a sensor in the transducer operating state when the sampling mode is set to free-running or when an event sensor detects an event. It shall be cleared when read. The TIM data/event bit shall be set when any of the TransducerChannel data/event bits are set.

This status bit shall be implemented for any event sensor or any sensor capable of operating with the sampling mode set to the free-running state. It shall also be implemented in any TIM that contains such a sensor or event sensor.

The TransducerChannel data or event bit shall be cleared when the TransducerChannel transitions from the transducer idle to the transducer operating state. The operating states are described in 5.4.1.

NOTE—While the TransducerChannel is not acquiring or consuming data in the transducer idle state, the bit shall remain set until it is read or otherwise cleared if it was set when the TransducerChannel transitioned from the transducer operating state to the transducer idle state.

5.13.7 Hardware error bit

The TransducerChannel hardware error bit shall be set when the condition it reports becomes valid. It shall be cleared when read provided the condition it reports is no longer valid. An event sensor shall set this bit if consistency checks are made and fail. The TIM hardware error bit shall be set when any of the TransducerChannel hardware error bits are set. The TIM manufacturer may determine any additional criteria for hardware errors.

Copyright © 2007 IEEE. All rights reserved.

The TIM hardware error bit in the returned TIM status and the TransducerChannel hardware error bit shall be implemented for any TIM that contains an event sensor if consistency checks are made (see 5.6.1).

This bit is not affected by a change of operating state.

5.13.8 Not operational bit

The TransducerChannel not operational bit shall be set when the TransducerChannel fails to comply with the manufacturers specifications. This bit shall be set during warm-up or any other condition when the TransducerChannel does not comply with specifications or is not operational.

The TIM not operational bit shall be the OR of all TransducerChannel not operational bits. It shall also be set if the TIM is not operational due to any other condition. The TIM not operational bit shall be cleared when the condition causing it to be set is removed.

This bit is not affected by a change of operating state.

5.13.9 Protocol error bit

A message has been received from the transducer interface media with protocol errors.

This bit is not affected by a change of operating state.

5.13.10 Data available/data processed bit

The TransducerChannel data available/data processed bit shall be set whenever the TransducerChannel has data available to be read for a sensor or has completed processing the data in an actuator. The TIM data available/data processed bit shall be set when any TransducerChannel in the TIM has data available to be read for a sensor or has completed processing the data in an actuator. They shall remain set as long as the condition persists. For a sensor, this means that it shall remain set until the data have been read. For an actuator, it shall remain set until new data have been written to the device or the status is read.

The TransducerChannel data availabel/data processed bit shall be cleared when the TransducerChannel transitions from the halted to the transducer operating state. The operating states are described in 5.4.1.

NOTE—While the TransducerChannel is not acquiring data in the transducer idle state, the bit shall remain set until it is read or otherwise cleared if it was set when the TransducerChannel transitioned from the transducer operating state to the transducer idle state.

5.13.11 Busy bit

The TransducerChannel busy bit shall be set whenever a TransducerChannel cannot support read or write access of TransducerChannel data over the data transport. Commands that may cause the TransducerChannel to become busy include the Run self-test command (see 7.1.1.5), Calibrate TransducerChannel command (see 7.1.2.10), Reset command (see 7.1.7.1), Zero TransducerChannel command (see 7.1.2.11), or any other command that interferes with normal operations. This bit shall be set only while the condition that it reports persists. The TIM busy bit shall be the OR of all TransducerChannel busy bits in a TIM.

This bit is not affected by a change of operating state.

Copyright © 2007 IEEE. All rights reserved.

5.13.12 Failed calibration bit

The TransducerChannel failed calibration bit shall be set when a "calibration" check (see 7.1.2.10) fails to produce the expected results. It shall be cleared when it is read. The TIM failed calibration bit shall be the OR of all TransducerChannel failed calibration bits in a TIM.

This bit is not affected by a change of operating state.

5.13.13 Failed self-test bit

The TransducerChannel failed self-test bit and the TIM failed self-test bit shall be set when a run self-test commands fails to provide the expected results (see 7.1.1.5). It shall be cleared when read. The TIM failed self-test bit shall be the OR of all TransducerChannel self-test bits in a TIM.

This bit is not affected by a change of operating state.

5.13.14 Data over-range or under-range bit

The TransducerChannel data over-range or under-range bit shall be set when the TransducerChannel detects an over-range or under-range condition. The TIM data over-range or under-range bit shall be the OR of all TransducerChannel data over-range or under-range bits in a TIM. This status bit shall be cleared when read.

An overrange condition exists when the output of the TransducerChannel represents a value greater than or more positive than the value given in the Design operational upper range limit field of the TransducerChannel TEDS (see 8.5.2.19). An underrange condition exists when the output of the TransducerChannel represents a value less than or more negative than the value given in the Design operational lower range limit field of the TransducerChannel TEDS (see 8.5.2.7).

The TransducerChannel data over-range or under-range bit shall be cleared when the TransducerChannel transitions from the halted to the transducer operating state. The operating states are described in 5.4.1.

NOTE—While the TransducerChannel is not acquiring or consuming data in the transducer idle state, the bit shall remain set until it is read or otherwise cleared if it was set when the TransducerChannel transitioned from the transducer operating state to the transducer idle state.

5.13.15 Corrections disabled bit

The Corrections disabled bit shall be set in any TIM or TransducerChannel that is capable of correcting the data when that capability is disabled. It shall be implemented for any TransducerChannel or TIM with this capability. This bit shall be set to zero for any TransducerChannel or TIM not implementing this capability.

This bit is not affected by a change of operating state.

5.13.16 Consumables exhausted bit

The TransducerChannel consumables exhausted bit shall be set when the TransducerChannel can no longer obtain any consumables that it requires to continue operation. The TIM consumables exhausted bit shall be the OR of all TransducerChannel consumables exhausted bits in a TIM. This status bit shall only be set while the condition persists.

Copyright © 2007 IEEE. All rights reserved.

This bit is not affected by a change of operating state.

5.13.17 Not-the-first-read-of-this-data-set bit

The Not-the-first-read-of-this-data-set bit of any TransducerChannel shall be set when a data set is read more than once. This bit shall be cleared when read.

This bit is not affected by a change of operating state.

5.14 Service request logic

Service requests are sent from a TIM to the NCAP when some condition exists within the TIM that requires attention. They are roughly analogous to interrupts in computers. However, unlike computer interrupts, service requests have no independent mechanism for communicating with the NCAP that the condition exists unless the Status-event protocol is enabled (see 7.1.1.11). The TIM service request bit may be read using the Read Status-Event Register command (see 7.1.1.8). This means that the NCAP is not able to respond within some minimum time to a service request. It may also be sent back to the NCAP when the service request bit is asserted via a status protocol message.

5.14.1 Service request masks

The TIM shall contain a service request mask register for the TIM and for each implemented TransducerChannel in a TIM. These registers are 4 octets wide. Writing a one to any bit position in the service request mask register allows the service request bit to be set when the corresponding bit in the status register is set. See Figure 11 for details.

The service request mask register bit positions correspond one-to-one with the bit positions in the status-event register, as defined in Table 9. The value placed into the bit 0 position of the service request mask register is not used as this corresponds to the service request bit. Since the service request bit directly generates the service request, it cannot be masked. The default power up value for the service request mask registers is all zeros (i.e., no status bits can generate a service request). The value of this register shall not be affected by receipt of either a Device Clear protocol or a Reset command.

5.14.2 Service requests

The service request signal is used in conjunction with the status registers and the service request mask registers to indicate exceptional conditions in the TIM. The NCAP typically reads the status of all TransducerChannels to determine which TransducerChannels are requesting service, and for what reason. The NCAP is not required to respond to a service request immediately.

If automatic status reporting has been configured, the TIM or any channel requesting service shall cause the status-event register to be returned via a status-event protocol message.

Figure 12 shows the logic used to generate the TIM service request.

Copyright © 2007 IEEE. All rights reserved.

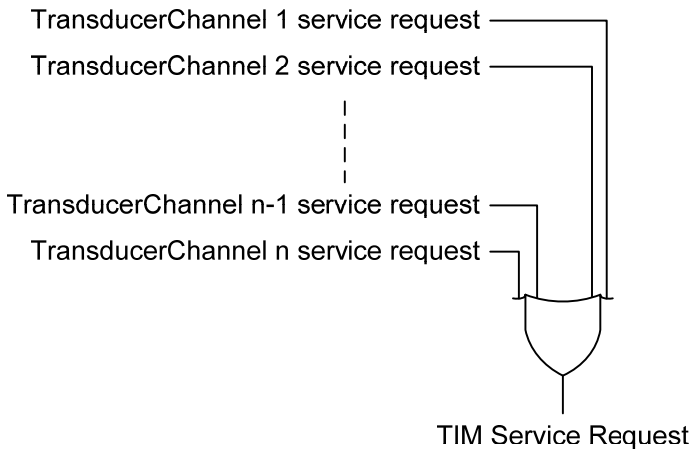


Figure 12—TIM service request generation

5.15 Hot-swap capability

It shall be possible to disconnect a TIM from the interface media or to connect a TIM to the interface media without powering down the system and without damage to either the TIM being inserted or anything else connected to the media. Transient impact upon transmissions in progress when a TIM is added or removed from the media is allowed, but no permanent damage is allowed.

The ability of the NCAP to detect an added or removed TIM is supported by this standard but is dependent on the capabilities of the underlying physical layers ability to detect such changes and to call the appropriate methods in the module communications API (see 11.5 and 11.6).

6. Message structures

This clause defines the structure of the messages sent across the Module Communications Interface.

6.1 Data transmission order and bit significance

The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a group of octets, the order of transmission of those octets is the normal order in which they are read in English. For example, in Table 10, the octets are transmitted in the order they are numbered.

NOTE—This transmission order applies to the Module Communications Interface and is conceptual only. The data transmission order and bit significance may be different at the physical layer.

Table 10—Transmission order of octets

1-Octet								1-Octet								1-Octet								1-Octet							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
First octet transmitted								2								3								4							
5								6								7								8							
9								—																							

Copyright © 2007 IEEE. All rights reserved.

Whenever an octet represents a numeric quantity, the left-most bit shown in Table 11 is the high-order or most significant bit. That is, the bit labeled 7 is the most significant bit. For example, the numeric value in Table 11 represents the value 170 (decimal) or 0xAA (Hexadecimal).

Table 11—Example of bit significance

	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value = 170 (decimal)	1	0	1	0	1	0	1	0

6.2 Command message structure

The message format of a command message is provided in Table 12.

Table 12—Command message structure

1-Octet							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Destination TransducerChannel Number (most significant octet)							
Destination TransducerChannel Number (least significant octet)							
Command class							
Command function							
Length (most significant octet)							
Length (least significant octet)							
Command-dependent octets . . .							

6.2.1 Destination TransducerChannel number

This field gives the 16 bit TransducerChannel number for the destination of the message.

6.2.2 Command class

The command class is defined in 7.1. Table 15 is the master index of command classes.

6.2.3 Command function

The command function is defined in Clause 7. Command function shall be interpreted in the context of command class, as described throughout Clause 7.

6.2.4 Length

Length is the number of command-dependent octets in this message. If the length of a received message does not match the length field of the received message, the message shall be discarded and the protocol error bit in the status register (see 5.13.9) shall be set.

Copyright © 2007 IEEE. All rights reserved.

6.2.5 Command-dependent octets

This field contains the information that is to be defined for the command. See the command definitions in Clause 7 for the details of what should be in this field.

6.3 Reply messages

Reply messages are used to reply to a received command. The message format of a reply message is provided in Table 13.

Table 13—Reply message structure

1-Octet							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Success/Fail Flag							
Length (most significant octet)							
Length (least significant octet)							
Reply-dependent octets . . .							

6.3.1 Success/Fail flag

If this octet is nonzero, it indicates that the command was successfully completed. If it is zero, the command failed and the system should check the status to determine why.

6.3.2 Length

Length is the number of reply-dependent octets in this message. If the length of a received message does not match the length field of the received message, the message shall be discarded and the protocol error bit in the status register (see 5.13.9) shall be set.

6.3.3 Reply-dependent octets

This field contains the information that is to be defined for the command. See the command definitions in Clause 7 for the details of what should be in this field.

6.4 TIM initiated message structure

The message format of a message that is initiated by a TIM is provided in Table 14. Examples of these messages are streaming data and status messages.

Copyright © 2007 IEEE. All rights reserved.

Table 14—TIM initiated message structure

1-Octet							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Source TransducerChannel Number (most significant octet)							
Source TransducerChannel Number (least significant octet)							
Command class							
Command function							
Length (most significant octet)							
Length (least significant octet)							
Command-dependent octets . . .							

6.4.1 Source TransducerChannel number

This field gives the 16 bit TransducerChannel number for the source of the message.

6.4.2 Command class

This field is the same as 6.2.2.

6.4.3 Command function

This field is the same as 6.2.3.

6.4.4 Length

This field is the same as 6.2.4.

6.4.5 Command-dependent octets

This field is the same as 6.2.5.

7. Commands

Commands are divided into two categories, standard and manufacturer-defined. Regardless of the category, the command is divided into 2 octets. The most significant octet shall be used to define the class of the command. The least significant octet, called the function, shall identify the specific command within the class. For example, if the most significant octet defines the Transducer idle state commands class, the least significant octet then specifies the command within that class as listed in Table 25.

Table 15 lists the various command classes.

Copyright © 2007 IEEE. All rights reserved.

Table 15—Standard command classes

cmdClassId	Attribute name	Category
0	Reserved	Reserved
1	CommonCmd	Commands common to the TIM and TransducerChannel
2	XdcrIdle	Transducer idle state
3	XdcrOperate	Transducer operating state
4	XdcrEither	Transducer either idle or operating state
5	TIMsleep	Sleep state
6	TIMActive	Tim active state commands
7	AnyState	Any state
8—127	ReservedClass	Reserved
128—255	ClassN	Open for manufacturers – N = class number

A TIM may generate a reply to a command under either of two circumstances. The first circumstance is when the command itself requires a reply. An example of this situation is a Query TEDS command. The second circumstance is when the Status-event protocol is enabled. In this case, the TransducerChannel or TIM will transmit whenever there is a non-masked change in the status register.

The information described as being passed is passed using the Command message structure, as described in 6.2. The following paragraphs describe what should be in the command-dependent octets in Table 12. If a reply to a command is being transmitted or received, the reply message structure (see 6.3) is used.

In the tables in this clause, some enumerations are “reserved” for future versions of this standard. Some enumerations are optional, and the TIM manufacturer may choose not to implement them. Enumerations designated as “open to manufacturers” may be used to designate conditions not described by the standard.

7.1 Standard commands

The control function allows commands to be sent to the TIM as a whole, or to each TransducerChannel thereof, that affect their state or operation. The list of standard command classes is given in Table 15.

The TIM shall respond to all unimplemented commands by setting the TIM invalid command bit in the status register. See 5.13.2 for a complete description of this bit.

7.1.1 Commands common to the TIM and TransducerChannel

This class of commands may be addressed to either the TIM or the TransducerChannel. The address, as described in 5.3, is used to designate whether it is to be executed by the TIM or the TransducerChannel logic. Table 16 lists the commands in this class. Commands in this class shall not be addressed to an AddressGroup, a TransducerChannel proxy, or globally. If one of these commands is addressed to an AddressGroup, a TransducerChannel proxy or globally the command shall be ignored and the command rejected bit in the TIM status-condition register shall be set.

Copyright © 2007 IEEE. All rights reserved.

Table 16—Common commands

cmdFunctionId	Command	State		Reply expected	Required/ optional
		TransducerChannel	TIM		
0	Reserved	—	—	—	—
1	Query TEDS	Any	Active	Yes	Required
2	Read TEDS segment	Any	Active	Yes	Required
3	Write TEDS segment	Any	Active	No	Required
4	Update TEDS	Any	Active	Yes	Required
5	Run self-test	Idle	Active	No	Required
6	Write service request mask	Any	Active	No	Required
7	Read service request mask	Any	Active	Yes	Required
8	Read status-event register	Any	Active	Yes	Required
9	Read status-condition register	Any	Active	Yes	Required
10	Clear status-event register	Any	Active	No	Required
11	Write status-event protocol state	Idle	Active	No	Required
12	Read status-event protocol state	Any	Active	Yes	Required
13–127	Reserved	—	—	—	—
128–255	Open for manufacturers	—	—	—	—

7.1.1.1 Query TEDS command

Argument attribute name: TEDSAccessCode data type UInt8

This command is used by the NCAP to solicit information required to read or write the TEDS. There is a single argument to this command and that is the TEDS Access Code that identifies the TEDS to be accessed. Table 17 lists the TEDS Access Codes for the TEDS defined by this standard.

The reply to a Query TEDS Command shall contain the information listed in Table 18. The TIM is required to provide a reply to all Query TEDS commands, regardless of whether the TEDS access code selects an implemented TEDS. The attributes field in the reply (see Table 19) shall indicate whether the TEDS is supported, whether it may be changed, whether the current contents are valid, and whether the TEDS is embedded in the TIM or located remotely (virtual). The TEDS status field is described in Table 20.

When the Unsupported TEDS attribute is set, the TIM shall return a zero for the “TEDSSize” and the “MaxTEDSSize” attributes.

When the Virtual TEDS attribute is set, the Read-only attribute shall also be set and the TIM shall return a zero for the “TEDSSize,” “TEDSCkSum,” and “MaxTEDSSize” attributes. It becomes the responsibility of the NCAP or the host processor to determine the sizes and attributes that are returned to the calling application. If the file cannot be located, the Unsupported attribute shall be set and the TIM shall return a zero for the “TEDSSize,” “TEDSCkSum,” and “MaxTEDSSize” attributes. If the file is found, the Invalid and Unsupported attributes shall be cleared and the Read-Only attribute, the “TEDSSize” attribute, the “TEDSCkSum” attribute, and the “MaxTEDSSize” attribute shall be determined from the file attributes of the remote file in which the TEDS resides.

Copyright © 2007 IEEE. All rights reserved.

Table 17—TEDS access codes

TEDS access code	TEDS name attribute	TEDS	Required/ optional
0	—	Reserved	—
1	MetaTEDS	Meta-TEDS1	Required
2	MetaIdTEDS	Meta-identification TEDS2	Optional
3	ChanTEDS	TransducerChannel TEDS1	Required
4	ChanIdTEDS	TransducerChannel Identification TEDS2	Optional
5	CalTEDS	Calibration TEDS1	Optional
6	CalIdTEDS	Calibration identification TEDS2	Optional
7	EUASTEDS	End users' application-specific TEDS3	Required
8	FreqRespTEDS	Frequency response TEDS1	Optional
9	TransferTEDS	Transfer function TEDS1	Optional
10	CommandTEDS	Commands TEDS2	Optional
11	TitleTEDS	Location and title TEDS2	Optional
12	XdcrName	User's transducer name TEDS3	Required
13	PHYTEDS	PHY TEDS1	Required
14	GeoLocTEDS	Geographic location TEDS2	Optional
15	UnitsExtention	Units extention TEDS2	Optional
16–127	—	Reserved	—
128–255	—	Manufacturer-defined TEDS	Optional

NOTES
1—A binary TEDS.
2—A text-based TEDS.
3—User-defined information content.

Table 18—Query TEDS response in the data field

Field	Data type	Field attribute name	Function
1	UInt8	TEDSAttrib	TEDS attributes (see Table 19)
2	UInt8	TEDSStatus	TEDS status
3	UInt32	TEDSSize	Current size of the TEDS
4	UInt16	TEDSCkSum	TEDS checksum
5	UInt32	MaxTEDSSize	Maximum TEDS size

Table 19—TEDS attributes

Bit	Data type	Field attribute name	Definition
0 (lsb)	Boolean	TEDSAttrib.ReadOnly	Read-only—Set to true if TEDS may be read but not written.
1	Boolean	TEDSAttrib.NotAvail	Unsupported—Set to true if TEDS is not supported by this TransducerChannel.
2	Boolean	TEDSAttrib.Invalid	Invalid—Set to true if the current TEDS image is invalid.
3	Boolean	TEDSAttrib.Virtual	Virtual TEDS—This bit is set to true if this is a virtual TEDS. (A virtual TEDS is any TEDS that is not stored in the TIM. The responsibility for accessing a virtual TEDS is vested in the NCAP or host processor.)
4	Boolean	TEDSAttrib.TextTEDS	Text TEDS—Set to true if the TEDS is text based.
5	Boolean	TEDSAttrib.Adaptive	Adaptive—Set to true if the contents of the TEDS can be changed by the TIM or TransducerChannel without the NCAP issuing a WriteTEDS segment command.
6	Boolean	TEDSAttrib.MfgrDefine	MfgrDefine—Set to True if the contents of this TEDS are defined by the manufacturer and will only conform to the structures defined in the standard if the TextTEDS attribute is also set.
7 (msb)	Boolean	TEDSAttrib.Reserved[7]	Reserved

Copyright © 2007 IEEE. All rights reserved.

Table 20—TEDS status

Bit	Data type	Field attribute name	Definition
0 (lsb)	Boolean	TEDSStatus.TooLarge	Too Large—The last TEDS image received was too large to fit in the memory allocated to this TEDS.
1	Boolean	TEDSStatus Reserved[1]	Reserved
2	Boolean	TEDSStatus Reserved[2]	Reserved
3	Boolean	TEDSStatus Reserved[3]	Reserved
4	Boolean	TEDSStatus Open[4]	Open to manufacturers
5	Boolean	TEDSStatus Open[5]	Open to manufacturers
6	Boolean	TEDSStatus Open[6]	Open to manufacturers
7 (msb)	Boolean	TEDSStatus Open[7]	Open to manufacturers

7.1.1.2 Read TEDS segment command

This command is used to read a TEDS into the NCAP. The arguments for this command are as shown in Table 21. Since the maximum size for an octet array is less than the maximum size for a TEDS, the TEDS segment offset is used to identify where in the TEDS the read access should start.

NOTE—Most TEDS will be small enough to fit within a single segment. In these cases, the contents of the segment offset field should be zero. However, TEDS are allowed to be larger than the maximum size of an octet array. The transmission of these large TEDS requires the segmentation of the TEDS for transmission.

Table 21—Data field for a read TEDS segment command

Field	Data type	Field attribute name	Function
1	Uint8	TEDSAccessCode	TEDS access code, as defined in Table 17.
2	UInt32	TEDSOffset	TEDS Segment offset (0 to [current size – 1])—This is the address relative to the beginning of the TEDS at which the block of data shall be read.

The reply to a Read TEDS Segment uses the Reply message (see 6.3). The reply-dependent octets returned within the message shall be as shown in Table 22. The first field contains the offset into the TEDS at which the block of data was taken and will in most cases match the TEDS Segment Offset in the Read TEDS Segment command. The remaining octets contain the data read from the TEDS. The reply shall contain all ones in the TEDS segment offset and 0 data octets if the TEDS is "virtual," is not supported, or is invalid. The number of octets returned is a function of the design and is determined from the message header (see 6.3). If the TEDSOffset is greater than the length of the TEDS, the TEDSOffset in the reply shall be equal to the TEDS length and the reply will contain 0 octets.

Table 22—Data field for a TEDS segment command reply

Field	Data type	Field attribute name	Function
1	UInt32	TEDSOffset	TEDS Segment offset (0 to [current size – 1])
2	OctetArray	RawTEDSBlock	TEDS data octets

7.1.1.3 Write TEDS segment command

This command is used to write a part of the TEDS. The arguments for a Write TEDS segment command are described in Table 23. Since the maximum size for an octet array is less than the maximum size for a TEDS, the TEDS segment offset is used to identify where in the TEDS the data in field 3 should be written. If the TEDSOffset is greater than the maximum length of the TEDS, the data shall be discarded and the command rejected bit in the status word (see 5.13.4) shall be set.

NOTE—Most TEDS will be small enough to fit within a single segment. In these cases, the contents of the segment offset field should be zero. However, TEDS are allowed to be larger than the maximum size of an octet array. The transmission of these large TEDS requires the segmentation of the TEDS for transmission.

Copyright © 2007 IEEE. All rights reserved.

Table 23—Data field for a write TEDS command

Field	Data type	Field attribute name	Function
1	UInt8	TEDSAccessCode	TEDS access code, as defined in Table 17.
2	UInt32	TEDSOffset	TEDS Segment offset (0 to [current size – 1])
3	OctetArray	RawTEDSBlock	TEDS Contents

The Write TEDS Segment command uses the Command message structure (see 6.2). If the maximum TEDS size is exceeded, the additional data shall not be written into memory and the current size of the TEDS shall be set to zero.

A Write TEDS Segment command shall create a new TEDS if one does not already exist with that access code. If the TIM is not designed to allow the TEDS to be created, the Write TEDS Segment command shall not write any data into TEDS memory because the TEDS is unsupported.

When the TIM begins to overwrite an existing TEDS, the TEDS being overwritten shall be marked as Invalid. It shall remain marked as Invalid until the Update TEDS command is received, as described in 7.1.1.4.

The Write TEDS Segment command does not generate a reply.

7.1.1.4 Update TEDS command

Argument attribute name: TEDSAccessCode data type UInt8

This command is used to cause a TEDS that was previously written into a TransducerChannel to be verified and copied into non-volatile memory (if this was not done as the TEDS was received). After the TEDS is verified, the TEDS may then be marked as Valid. If the verification fails, the TEDS shall remain marked as invalid.

There is one argument to this command, and it contains the TEDS access code, as defined in Table 17.

The reply to an Update TEDS command shall contain the information listed in the reply to a Query TEDS command as described in 7.1.1.1.

7.1.1.5 Run self-test command

Argument attribute name: Test2Run data type UInt8

This command is used to cause the addressed device to run a self-test. There is one argument to this command and that is an 8 bit enumeration specifying the diagnostic to run as shown in Table 24. The manufacturer may identify additional tests that are available for a particular device. If an enumeration is requested that is not implemented, the invalid command status bit (see 5.13.2) shall be set in the appropriate status register.

Table 24—Enumerations for the run diagnostic command

Enumeration	Argument attribute name	Diagnostic
0	Test2Run.ConfidenceTest	Short confidence test
1	Test2Run.TestAll	Test all
2–255	Test2Run.RunTest[N] $2 \leq N \leq 255$	Manufacturer-defined

If a manufacturer implements additional tests, they shall be listed in the Commands TEDS for the TransducerChannel or TIM.

Copyright © 2007 IEEE. All rights reserved.

This command shall only be executed when the TransducerChannel is in the idle state. If this command is received when the TransducerChannel is in any other state, the command rejected bit (see 5.13.4) shall be set.

If the manufacturer wishes to write a single diagnostic program that tests the TIM and the TransducerChannels, then the command should be addressed to the TIM and all TransducerChannels shall be in the idle state, else the command shall be rejected and the command rejected status bit set.

7.1.1.6 Write service request mask

Argument attribute name: SRMask data type UInt32

This command is used to write the service request mask for the addressed TransducerChannel or TIM. The mask is a 32 bit word that is used as described in 5.14.1.

7.1.1.7 Read service request mask

Reply argument attribute name: SRMask data type UInt32

This command is used to read the service request mask from the addressed TransducerChannel or TIM. The command has no arguments. However, it does generate a reply that has a single argument. This argument gives the currently active service request mask as defined in 5.14.1 for the addressed TransducerChannel.

7.1.1.8 Read status-event register

Reply argument attribute name: SERegister data type UInt32

This command is used to read the status from the addressed TransducerChannel or TIM. The command has no arguments. However, it does generate a reply that has a single argument. This argument gives the current contents of the Status-Event register as defined in 5.13 for the addressed TransducerChannel.

7.1.1.9 Read status-condition register

Reply argument attribute name: SCRegister data type UInt32

This command is used to read the status-condition register from the addressed TransducerChannel or TIM.

The command has no arguments. However, it does generate a reply that has a single argument. This argument gives the current contents of the Status-Condition register as defined in 5.13 for the addressed TransducerChannel.

7.1.1.10 Clear status-event register

This command is used to clear the status-event register for the addressed TransducerChannel or TIM. If the entire TIM is specified, the command shall clear all status-event registers for the TIM itself, including all TransducerChannel event registers. It does not clear the corresponding mask or condition registers. This command has no arguments.

Copyright © 2007 IEEE. All rights reserved.

7.1.1.11 Write status-event protocol state

Argument attribute name: SEProtocol data type Boolean

This command is used to enable or disable the status-event protocol defined in 5.13. When this state is enabled, a stream shall be initiated that will send the 32 bit status register any time the service request bit is asserted. Note that if a channel is requesting service, the channel register shall be sent and if the TIM itself is requesting service, the TIM status register shall be sent.

This command has a single argument. If the value is true, the status-event protocol is enabled. Otherwise the status-event protocol is disabled.

7.1.1.12 Read status-event protocol state

Reply argument attribute name: SEProtocol data type Boolean

This command is used to read the state of the status-event protocol defined in 5.13. The command has no arguments. However, it does generate a reply that has a single argument. This argument gives the current state of the status-event protocol. If the value is true, the status-event protocol is enabled. Otherwise the status-event protocol is disabled.

7.1.2 Transducer idle state commands

The idle state class of commands, as listed in Table 25, shall only be executed when the TransducerChannel is in the idle state. If one of these commands is received when the TransducerChannel is in any other state, the command rejected bit in the TransducerChannel Status-Condition Register (see 5.13) shall be set and the command shall be ignored.

All commands in this class require a destination TransducerChannel number greater than zero. If the destination TransducerChannel number in the octet array (see 5.3) is zero, the command rejected bit in the TransducerChannel Status-Condition Register (see 5.13) shall be set and the command shall be ignored.

If a Transducer Idle State command is sent to a TransducerChannel that does not support that feature or does not support changing that feature, the command shall be ignored and the TransducerChannel invalid command bit in the status word (see 5.13.2) shall be set.

The TIM shall be in the active state for any of these commands to be received by the transducer channel. If one of these commands is received when the TIM is not in the active state, the command rejected bit in the TIM Status-Condition Register (see 5.13) shall be set and the command shall be ignored.

Commands listed as “Optional” in Table 25 become required if the feature that they support is programmable.

Some commands in this class that may be issued to a TransducerChannel proxy (i.e., there is a “Yes” under the column labeled “Proxy” in Table 25). When one of these commands is issued to a proxy, it shall have the same effect as issuing that same command to each member of the TransducerChannel proxy individually.

Copyright © 2007 IEEE. All rights reserved.

Table 25—Transducer Idle state commands

cmdFunctionId	Command	Address class			Reply expected	Required/ optional
		TransducerChannel	Proxy	Group		
0	Reserved	—	—	—	—	—
1	Set TransducerChannel data repetition count	Yes	No	No	No	Optional
2	Set TransducerChannel pre-trigger count	Yes	No	No	No	Optional
3	AddressGroup definition	Yes	Yes	No	No	Required
4	Sampling mode	Yes	Yes	No	No	Optional
5	Data Transmission mode	Yes	Yes	Yes	No	Optional
6	Buffered state	Yes	Yes	No	No	Optional
7	End-of-data-set operation	Yes	Yes	No	No	Optional
8	Actuator-halt mode	Yes	Yes	No	No	Optional
9	Edge-to-report	Yes	Yes	No	No	Optional
10	Calibrate TransducerChannel	Yes	Yes	Yes	Yes	Optional
11	Zero TransducerChannel	Yes	Yes	Yes	Yes	Optional
12	Write corrections state	Yes	Yes	No	No	See 7.1.2.12
13	Read corrections state	Yes	Yes	No	No	See 7.1.2.13
14	Write TransducerChannel initiate trigger state	Yes	No	No	No	Optional
15	Write TransducerChannel initiate trigger configuration	Yes	No	No	No	Optional
16–127	Reserved	—	—	—	—	—
128–255	Open for manufacturers	—	—	—	—	—
NOTE—Optional commands become required if the features they support are programmable.						

7.1.2.1 Set TransducerChannel data repetition count

Argument attribute name: RepCount data type UInt16

This command is used to change the number of data samples in a data set to a number between zero and the number found in the Maximum data repetitions field of the TransducerChannel TEDS (see 8.5.2.28).

The only argument to this command is a 16 bit integer giving the number of samples in a data set. If the value of the argument exceeds the Maximum data repetitions, the data repetition count shall not be changed and the TransducerChannel command rejected bit shall be set in the status register (see 5.13.4).

7.1.2.2 Set TransducerChannel pre-trigger count

Argument attribute name: PreTrigCount data type UInt16

This command is used by TransducerChannels that are capable of being operated in the free-running with pre-trigger mode (see 5.10.1.3), to change the number of data samples in a data set that can be acquired and stored before the receipt of a trigger. This number may be any number between zero and the number found in the Maximum pre-trigger samples field of the TransducerChannel TEDS (see 8.5.2.32).

Copyright © 2007 IEEE. All rights reserved.

The only argument to this command is a 16 bit integer giving the number of samples in a data set that are to be acquired before the trigger. If the value of the argument exceeds the Maximum pre-trigger samples (see 8.5.2.28), the pre-trigger count shall not be changed and the TransducerChannel command rejected bit shall be set in the status register (see 5.13.4).

7.1.2.3 AddressGroup definition

Argument attribute name: GroupAdd data type UInt16

This command assigns a TransducerChannel to an AddressGroup. It has a single argument in the data field and that is the group address. See Table 5 for more details of addresses. The addressed TransducerChannel, whether normal or proxy, is assigned to the group. If the argument to this command is zero, the addressed TransducerChannel is removed from all AddressGroups to which it has been assigned.

7.1.2.4 Sampling mode

Argument attribute name: SampleMode data type UInt8

The NCAP uses this command when changing the sampling mode of a TransducerChannel. Not all allowable modes of operation defined by this standard are accepted by a given TransducerChannel. The allowable sampling modes for a given TransducerChannel are based on the sampling attributes for that TransducerChannel found in the TransducerChannel TEDS (see 8.5.2.44).

This command has a single argument. The list of allowable values for this argument is shown in Table 26. If a Sampling mode command is sent to a TransducerChannel that does not support that mode or does not support changing that mode, the TransducerChannel invalid command bit in the status word (see 5.13.2) shall be set.

Table 26—Allowable values for the sampling mode argument

Enumeration	Argument attribute name	Operating mode
0	SampleMode.Reserved[0]	Reserved
1	SampleMode.TriggerInit	Trigger-initiated
2	SampleMode.FreeNoPre	Free-running without pre-trigger
3	SampleMode.FreePreTrig	Free-running with pre-trigger
4	SampleMode.Continuous	Continuous sampling
5	SampleMode.Immediate	Immediate operation
6–127	SampleMode.Reserved[N] $6 \leq N \leq 127$	Reserved
128–255	SampleMode.Open[N] $128 \leq N \leq 255$	Open to manufacturers

7.1.2.5 Data transmission mode

Argument attribute name: XmitMode data type UInt8

The command is used to control the data transmission mode of operation as described in 5.10.2. Whether a given TransducerChannel supports the allowable data transmission modes of operation is described in the TransducerChannel TEDS in the data transmission attribute (see 8.5.2.49).

This command has a single argument. The list of allowable values for this argument is shown in Table 7. If a Data transmission mode command is sent to a TransducerChannel that does not support that mode or does not support changing that mode, the TransducerChannel invalid command bit in the status word (see 5.13.2) shall be set.

Copyright © 2007 IEEE. All rights reserved.

7.1.2.6 Buffered state

Argument attribute name: BufferOnOff data type Boolean

The command is used to turn on or off buffered operation as described in 5.10.3. Whether a given TransducerChannel supports buffered operation is described in the TransducerChannel TEDS in the buffered attribute (see 8.5.2.47).

This command has a single argument. If the value is true, buffering is enabled. Otherwise buffering is disabled.

7.1.2.7 End-of-Data-Set operation mode

Argument attribute name: EndDataSetMode data type UInt8

The command is used for actuators to describe what action the actuator should take when it reaches the end of a data set as described in 5.10.4. Whether a given actuator supports changing the End-of-Data-Set operation and the allowable modes is described in the TransducerChannel TEDS in the End-of-Data-Set operation attribute (see 8.5.2.48).

This command has a single argument. The list of allowable values for this argument is shown in Table 27. If an End-of-Data-Set operation mode command is sent to a TransducerChannel that does not support that mode or does not support changing that mode, the TransducerChannel invalid command bit in the status word (see 5.13.2) shall be set.

Table 27—Allowable values for the End-of-Data-Set operation mode argument

Enumeration	Argument attribute name	Operating mode
0	EndDataSetMode.Reserved[0]	Reserved
1	EndDataSetMode.Hold	Hold
2	EndDataSetMode.Recirc	Recirculate
3–127	EndDataSetMode.Reserved[N] 3≤N≤127	Reserved
128–256	EndDataSetMode.Open[N] 128≤N≤255	Open to manufacturers

7.1.2.8 Actuator halt operating mode

Argument attribute name: ActuatorHalt data type UInt8

The command is used to control what the output of an actuator will do when it receives a TransducerChannel Idle command while it is processing a data set. The possibilities for a given actuator are given in the TransducerChannel TEDS in the actuator halt attribute (see 8.5.2.51).

This command has a single argument. The list of allowable values for this argument is shown in Table 28.

Table 28—Allowable values for the actuator halt operating mode argument

Enumeration	Argument attribute name	Operating mode
0	ActuatorHalt.Reserved[0]	Reserved
1	ActuatorHalt.Hold	Hold
2	ActuatorHalt.Finish	Finish data set
3	ActuatorHalt.Ramp	Ramp
4–127	ActuatorHalt.Reserved[N] 4≤N≤127	Reserved
128–256	ActuatorHalt.Open[N] 128≤N≤255	Open to manufacturers

Copyright © 2007 IEEE. All rights reserved.

7.1.2.9 Edge-to-Report

Argument attribute name: EdgeReported data type UInt8

The command is used to describe which edge or edges an event sensor will report. The possibilities for a given event sensor are given in the TransducerChannel TEDS in the Edge-to-Report attribute (see 8.5.2.50).

This command has a single argument. The list of allowable values for this argument is shown in Table 29.

Table 29—Allowable values for the Edge-to-Report operation mode argument

Enumeration	Argument attribute name	Operating mode
0	EdgeReported.Reserved[0]	Reserved
1	EdgeReported.Rising	Report Rising Edges
2	EdgeReported.Falling	Report Falling Edges
3	EdgeReported.BothEdges	Report both Edges
4–127	EdgeReported.Reserved[N] $4 \leq N \leq 127$	Reserved
128–256	EdgeReported.Open[N] $128 \leq N \leq 255$	Open to manufacturers

7.1.2.10 Calibrate TransducerChannel

This command is used to cause the TransducerChannel to run a procedure to verify that its output is what should be expected. The exact definition of what happens during a Calibrate TransducerChannel command is a function of the TransducerChannel design and is not a subject of this standard.

A reply to this command shall always be generated upon completion of the operation. The reply shall be the same as the reply to a Read Status-Event Register command (see 7.1.1.8).

NOTE—A “shunt Cal” where a resistor of known value is applied across one leg of a Wheatstone bridge is a common form of what is expected when this command is executed.

7.1.2.11 Zero TransducerChannel

This command is used to cause the TransducerChannel to run a procedure to zero its input or output. The exact definition of what happens as a result of a Zero TransducerChannel command is a function of the TransducerChannel design and is not a subject of this standard.

A reply to this command shall always be generated upon completion of the operation. The reply shall be the same as the reply to a Read Status-Event Register command (see 7.1.1.8).

NOTE—A common result of executing this command is to have a sensor output go to zero and to use this value as a zero reference.

7.1.2.12 Write corrections state

Argument attribute name: CorrectState data type UInt8

This command is used with TransducerChannels that are capable of applying the corrections as described in 8.6.1 within the TIM. This command causes the TransducerChannel to enable the correction process so that the unit outputs data in the Physical Units specified in the Calibration TEDS. This command is mandatory for TransducerChannels with a built-in correction capability.

Copyright © 2007 IEEE. All rights reserved.

This command has a single argument. If the value is true, the corrections capability is enabled. Otherwise the corrections capability is disabled.

The Corrections disabled status bit (see 5.13.15) shall be cleared upon receipt of this command.

7.1.2.13 Read corrections state

Reply argument attribute name: CorrectState data type UInt8

This command is used with TransducerChannels that are capable of correcting the data within the TIM. This command is mandatory for TransducerChannels with a built-in correction capability.

A reply to this command has a single argument. If the value is true, the corrections capability is enabled. Otherwise the corrections capability is disabled.

7.1.2.14 Write TransducerChannel initiate trigger state

Argument attribute name: EventTrig data type Boolean

This command allows an event sensor TransducerChannel to initiate a trigger command when an event occurs. If a sensor or actuator receives this command it shall set the command rejected bit in the status word (see 5.13.4).

This command has a single argument. If the value is true, the event sensor is enabled to initiate a trigger when an event occurs. Otherwise the trigger capability is disabled.

7.1.2.15 Write TransducerChannel initiate trigger configuration

Argument attribute name: InitTrig data type Boolean

This command is used to provide the information needed by an event sensor to initiate a trigger command. The arguments for this command are as shown in Table 30.

Table 30—Write TransducerChannel initiate trigger configuration

Field	Data type	Argument attribute name	Function
1	UInt16	InitTrig.destId	The “destId” specifies the desired destination.
2	UInt16	InitTrig.channelId	The “channelId” specifies the desired TransducerChannel.
3	Struct	InitTrig.qosParams	The “qosParams” is the desired quality of service parameters. See 9.3.1.3 for details.

7.1.3 Transducer operating state commands

The Transducer operating state class of commands shall only be executed when the TransducerChannel is in the operational state. If one of these commands is received when the TransducerChannel is in any other state, the command rejected bit (see 5.13) in the TransducerChannel Status-Condition register shall be set and the command shall be ignored.

The commands that are allowed in the transducer operating state are listed in Table 31.

Copyright © 2007 IEEE. All rights reserved.

The TIM shall be in the active state for any of these commands to be received by the TransducerChannel. If one of these commands is received when the TIM is not in the active state, the command rejected bit in the TIM Status-Condition Register (see 5.13) shall be set and the command shall be ignored.

All commands in this class require a destination TransducerChannel number greater than zero. If destination TransducerChannel number in the message is zero, the command rejected bit (see 5.13) in the TIM Status-Condition register shall be set and the command shall be ignored.

Table 31—Transducer operating state commands

cmdFunctionId	Command	Address class			Reply expected	Required/ optional
		TransducerChannel	Proxy	Group/ global		
0	Reserved	—	—	—	—	—
1	Read TransducerChannel data-set segment	Yes	Yes	No	Yes	See NOTE
2	Write TransducerChannel data set segment	Yes	Yes	No	No	See NOTE
3	Trigger command	Yes	Yes	Yes	No	Required
4	Abort Trigger	Yes	Yes	Yes	No	Optional
5–127	Reserved	—	—	—	—	—
128 –255	Open for manufacturers	—	—	—	—	—
NOTE—A Read TransducerChannel data set segment command is required for sensors. A Write TransducerChannel data set segment command is required for an actuator.						

7.1.3.1 Read TransducerChannel data-set segment

Argument attribute name: DataSetOffset data type UInt32

This command is used to read a segment of a data set. There is a single argument to this command. That argument specifies the offset into the data set at which reading should start. Since the maximum size for an octet array that may be handled by a given physical transport layer is less than the maximum size for a data set, the data set offset is used to identify where in the data set the read access should start.

NOTE—Most data sets will be small enough to fit within a single segment. However, the standard allows the maximum TransducerChannel data repetitions to be 65 356 and the data model, which defines the number of octets in a data sample, to be 255, which gives the maximum data set size of 16 777 216 octets. In cases where the data set will fit within a single message the contents of the segment offset field should be zero. However, larger data sets that will not fit within one message require segmentation of the data set for transmission and the segment offset should be appropriate for each segment.

The reply to a Read TransducerChannel data set segment command uses the Reply message structure (see 6.3). The reply-dependent octets returned within the datagram reply message shall be as shown in Table 13. As shown in Table 32, the first field contains the offset into the data set at which the block of data was taken and will in most cases match the Data-set Segment Offset in the Read TransducerChannel data set segment command. The remaining octets contain the data read from the data set. The reply shall contain all ones in the data set segment offset and 0 data octets if the data set is empty. The number of octets returned is a function of the design of the TransducerChannel, and the NCAP will determine the number from the arguments to the API method used to return the message. (see 11.2 or 11.3). If the ReadSensor.Offset is greater than the number of octets in the data set, the ReadSensor.Offset in the reply shall be equal to the maximum number of octets in the data set and the reply will contain 0 octets.

Copyright © 2007 IEEE. All rights reserved.

Table 32—Read TransducerChannel data set segment reply arguments

Field	Data type	Argument attribute name	Function
1	UInt32	ReadSensorOffset	Data set offset (0 to [current size – 1])—This is the address relative to the beginning of the data set at which reading of the data shall begin.
2	N*UInt8	ReadSensorData	The block of data read from the sensor.

For a TransducerChannel being operated in one of the streaming data transmission modes (see 5.10.2), the command rejected bit in the status register shall be set if this command is received and the command shall be ignored.

If the destination TransducerChannel number in the octet array is zero, the command rejected bit in the TIM Status-Condition Register (see 5.13.4) shall be set and the command shall be ignored.

7.1.3.2 Write TransducerChannel data-set segment

Argument attribute name: WriteActuator

This command is used to write a data set into a TransducerChannel overwriting any previous content. The arguments to a write TransducerChannel data set segment command shall be as described in Table 33.

Since the maximum size for an octet array that may be handled by a given physical transport layer is less than the maximum size for a data set, the data set offset is used to identify where in the data set the write access should start. If the WriteActuator.Offset is greater than the maximum length of the data set, the data shall be discarded and the command rejected bit in the status word (see 5.13.4) shall be set.

NOTE—Most data sets will be small enough to fit within a single segment. However, the standard allows the maximum TransducerChannel data repetitions to be 65 356 and the data model, which defines the number of octets in a data sample, to be 255, which gives the maximum data set size of 16 777 216. In cases where the data set will fit within a single message, the contents of the segment offset field should be zero. However, data sets larger than will fit within one message require the segmentation of the data set for transmission.

Table 33—Write TransducerChannel data set segment command data field

Field	Data type	Argument attribute name	Function
1	UInt32	WriteActuator.Offset	Data set offset (0 to [current size – 1])—This is the address relative to the beginning of the data set at which writing of the data shall begin.
2	N*UInt8	WriteActuator.DataBlock	Data block

For a TransducerChannel being operated in the streaming mode, the command rejected bit (see 5.13) in the status register shall be set if this command is received and the command shall be ignored.

If the destination TransducerChannel number in the octet array is zero, the command rejected bit (see 5.13) in the TIM Status-Condition Register shall be set and the command shall be ignored.

7.1.3.3 Trigger command

This command causes an action as described in trigger commands in 5.11.2.1.

There are no arguments to this command.

It may be addressed to a TransducerChannel, a TransducerChannel proxy, an AddressGroup, or globally. If it is addressed globally, it shall trigger each trigger-enabled TransducerChannel within the TIM. If the TransducerChannel is a sensor, the TransducerChannel shall behave according to the Sensor Triggering

Copyright © 2007 IEEE. All rights reserved.

State Diagram (see Figure 7). If the TransducerChannel is an actuator, then the TransducerChannel shall behave according to the Actuator Trigger State Diagram (see Figure 8).

7.1.3.4 Abort trigger

This command is used to abort a trigger operation. If the TransducerChannel is a sensor, the TransducerChannel shall behave according to the Sensor Triggering State Diagram (see Figure 7). If the TransducerChannel is an actuator, then the TransducerChannel shall behave according to the Actuator Trigger State Diagram (see Figure 8).

There are no arguments to this command.

It may be addressed to a TransducerChannel, a TransducerChannel proxy, an AddressGroup, or globally. If it is addressed globally, it shall abort the trigger on each triggered TransducerChannel within the TIM.

7.1.4 Transducer either idle or operating state commands

The commands in this class may be issued to a TransducerChannel at any time after it leaves the Transducer Initialization state. The commands that are allowed in this class are listed in Table 34.

Table 34—TransducerChannel idle or operational state commands

cmdFunctionId	Command	Address class			Reply expected	Required/ optional
		TransducerChannel	Proxy	Group/ global		
0	Reserved	—	—	—	—	—
1	TransducerChannel Operate	Yes	Yes	Yes	No	Required
2	TransducerChannel Idle	Yes	Yes	Yes	No	Required
3	Write TransducerChannel trigger state	Yes	Yes	Yes	No	Optional
4	Read TransducerChannel trigger state	Yes	Yes	Yes	No	Required
5	Read TransducerChannel data repetition count	Yes	No	No	Yes	See NOTE 1
6	Read TransducerChannel pre-trigger count	Yes	No	No	Yes	See NOTE 2
7	Read AddressGroup assignment	Yes	Yes	No	Yes	Required
8	Read Sampling mode	Yes	Yes	No	Yes	Optional
9	Read data transmission mode	Yes	Yes	No	Yes	
10	Read buffered state	Yes	Yes	No	Yes	Optional
11	Read end-of-data-set operation	Yes	Yes	No	Yes	Optional
12	Read actuator halt mode	Yes	Yes	No	Yes	Optional
13	Read edge-to-report mode	Yes	Yes	No	Yes	Optional
14	Read TransducerChannel initiate trigger state	Yes	No	No	Yes	Optional
15	Read TransducerChannel initiate trigger configuration	Yes	No	No	Yes	Optional
16	Device clear	Yes	Yes	Yes	No	Optional
17–127	Reserved	—	—	—	—	—
128–255	Open for manufacturers	—	—	—	—	—
NOTE 1—This function is required if the Set TransducerChannel data repetition count is implemented. NOTE 2—This function is required if the Set TransducerChannel pre-trigger count is implemented.						

Copyright © 2007 IEEE. All rights reserved.

If any optional command in this class is sent to a TransducerChannel that does not support that command, the command rejected bit (see 5.13.4) in the TransducerChannel Status-Condition register shall be set and the command shall be ignored.

All commands in this class require a destination TransducerChannel number greater than zero. If destination TransducerChannel number in the message is zero, the command rejected bit in the TransducerChannel Status-Condition Register (see 5.13.4) shall be set and the command shall be ignored.

The TIM shall be in the active state for any of these commands to be received by the transducer channel.

7.1.4.1 TransducerChannel operate

This command causes a TransducerChannel in the Transducer Idle state to transition into the Transducer Operating state. If the TransducerChannel is already in the Transducer Operating state, the command is ignored. See 5.4.1 for a description of the operating states of a TransducerChannel.

This command has no arguments.

7.1.4.2 TransducerChannel idle

This command forces the addressed TransducerChannel to transition into the Transducer Idle state. If the TransducerChannel is already in the Transducer Idle state, the command is ignored. See 5.4.1 for a description of the operating states of a TransducerChannel.

This command has no arguments.

7.1.4.3 Write TransducerChannel trigger state

Argument attribute name: TrigState data type Boolean

This command is used to enable or disable TransducerChannel triggering as described in 5.4.2 and 5.11.

This command has a single argument. If the value is true, triggering is enabled for the addressed TransducerChannel. Otherwise triggering is disabled for the addressed TransducerChannel.

If the TransducerChannel does not support triggering and this command is received, the command rejected bit in the status-event register shall be set (see 5.13.4) and the command shall be ignored.

7.1.4.4 Read TransducerChannel trigger state

Reply argument attribute name: TrigState data type Boolean

This command is used to read the triggering state of the TransducerChannel as described in 5.4.2.

The reply to this command has a single argument. If the value is true, triggering is enabled for the addressed TransducerChannel. Otherwise triggering is disabled for the addressed TransducerChannel. If the TransducerChannel does not support triggering, the reply to this command shall be False.

Copyright © 2007 IEEE. All rights reserved.

7.1.4.5 Read TransducerChannel data repetition count

Reply argument attribute name: RepCount data type UInt16

This command is used to read the actual number of TransducerChannel data repetition counts (see 7.1.2.1) that are assigned for the addressed TransducerChannel.

The command has no arguments. However, it does generate a reply that has a single UInt16 argument. This argument gives the current assigned value for the TransducerChannel data repetition count.

7.1.4.6 Read TransducerChannel pre-trigger count

Reply argument attribute name: PreTrigCount data type UInt16

This command is used to read the actual number of TransducerChannel pre-trigger counts that are assigned for the addressed TransducerChannel (see 7.1.2.2).

The command has no arguments. However, it does generate a reply that has a single UInt16 argument. This argument gives the current assigned value for the TransducerChannel pre-trigger count.

7.1.4.7 Read AddressGroup assignment

Reply argument attribute name: GrpAssignment data type UInt16

This command is used to read the AddressGroup(s) to which the addressed TransducerChannel is assigned (see 11.3.14).

The command has no arguments. However, it does generate a reply that has a single UInt16 argument that gives the AddressGroup addresses to which the TransducerChannel is assigned. A value of zero means that this TransducerChannel is not assigned to any AddressGroup.

7.1.4.8 Read sampling mode

Reply argument attribute name: SampleMode data type UInt8

The Read Sampling Mode command allows the NCAP to determine the actual sampling mode in which the TransducerChannel is operating (see 5.10.1 and 8.5.2.44).

The command has no arguments. However, it does generate a reply that has a single UInt8 argument that gives the current sampling mode for the TransducerChannel. The list of allowable values for this argument is shown in Table 26.

7.1.4.9 Read data transmission mode

Reply argument attribute name: XmitMode data type UInt8

The Read data transmission mode command allows the NCAP to determine the data transmission mode of the TransducerChannel (see 5.10.2 and 8.5.2.49).

Copyright © 2007 IEEE. All rights reserved.

The command has no arguments. However, it does generate a reply that has a single UInt8 argument that gives the current data transmission mode for the TransducerChannel. Table 7 lists the allowable arguments for the reply to this command.

7.1.4.10 Read buffered state

Reply argument attribute name: BufferOnOff data type Boolean

The command has no arguments. However, it does generate a reply that has a single argument of type Boolean. If that argument is true, buffering is enabled. Otherwise buffering is disabled.

7.1.4.11 Read end-of-data-set operation mode

Reply argument attribute name: EndDataSetMode data type UInt8

The Read end-of-data-set operation mode command allows the NCAP to determine the action that an actuator TransducerChannel will take upon reaching the end of the current data set if another data set has not been written while it was applying the current data set.

The command has no arguments. However, it does generate a reply that has a single argument. Table 27 gives the allowable values for the argument.

7.1.4.12 Read actuator halt mode

Reply argument attribute name: ActuatorHalt data type UInt8

The command has no arguments. However, it does generate a reply that has a single argument. Table 28 gives the allowable values for the argument.

7.1.4.13 Read Edge-to-Report mode

Reply argument attribute name: EdgeReported data type UInt8

The command has no arguments. However, it does generate a reply that has a single argument. Table 29 gives the allowable values for the argument.

7.1.4.14 Read TransducerChannel initiate trigger state

Reply argument attribute name: EventTrig data type Boolean

This command allows an NCAP to determine whether an event sensor TransducerChannel is set up to initiate a trigger command when an event occurs. If a sensor or actuator receives this command, it shall set the Command rejected bit in the status word (see 5.13.4).

This command has no arguments. However, the reply to this command has a single argument. If the value is true, the event sensor is enabled to initiate a trigger when an event occurs. Otherwise the trigger capability is disabled.

Copyright © 2007 IEEE. All rights reserved.

7.1.4.15 Read TransducerChannel initiate trigger configuration

Reply argument attribute name: InitTrig data type Boolean

This command is used to read the information needed to allow a TransducerChannelto initiate a trigger command.

This command has no arguments. However, the reply to this command has the arguments listed in Table 30. See 11.3.2 for the meaning of these arguments.

7.1.4.16 Device clear

This command is used to clear all input buffers and output buffers of the addressed TransducerChannels. For more information on device clear, see Figure 7 for sensor trigger states and Figure 8 for actuator trigger state diagrams.

This command has no arguments and does not generate a reply.

7.1.5 TIM sleep state commands

The following commands may only be executed when the TIM is in the sleep state. See Table 35 for the list of allowable commands for this class.

All commands in this class require a destination TransducerChannel number of zero. If destination TransducerChannel number in the message is not zero, the command rejected bit in the TransducerChannel Status-Condition Register (See 5.13.4) shall be set and the command shall be ignored.

Table 35—Sleep state commands

cmdFunctionId	Command	Address class		Reply expected	Required/ optional
		TIM	Global		
0	Reserved	—	—	—	—
1	Wake-up	Yes	No	Yes	Optional
2–127	Reserved	—	—	—	—
128–255	Open for manufacturers	—	—	—	—

7.1.5.1 Wake-up

This command forces the addressed TIM to transition into the active state.

This command has no arguments.

A reply to this command shall always be generated upon completion of the operation. The reply shall be the same as the reply to a Read Status-Event Register command (see 7.1.1.8) addressed to the TIM.

NOTE—If this command is sent to multiple TIMs, the underlying IEEE 1451.X must be able to handle replies from multiple TIMs without disrupting the operation of the system.

Copyright © 2007 IEEE. All rights reserved.

7.1.6 TIM active state commands

The following commands may only be executed when the TIM is in the active state (see Table 36). If one of these commands is received when the TIM is not in the active state, the command rejected bit in the TIM Status-Condition Register (see 5.13) shall be set and the command shall be ignored.

All commands in this class require a destination TransducerChannel number of zero. If destination TransducerChannel number in the message is not zero, the command rejected bit in the TIM Status-Condition Register (see 5.13.4) shall be set and the command shall be ignored.

Table 36—TIM active state commands

cmdFunctionId	Command	Address class		Reply expected	Required /optional
		TIM	Global		
0	Reserved	—	—	—	—
1	Read TIM version	Yes	No	Yes	Required
2	TIM Sleep	Yes	No	No	Optional
3	Store operational setup	Yes	No	No	Required
4	Recall operational setup	Yes	No	No	Required
5	Read IEEE 1451.0 Version	Yes	No	Yes	Required
6–127	Reserved	—	—	—	—
128–255	Open for manufacturers	—	—	—	—

7.1.6.1 Read TIM version

Reply argument attribute name: TIMVersion data type UInt16

This command is used to read the version number from a TIM.

The command has no arguments. However, it does generate a reply that has a single argument. The value of this argument is the manufacturer-defined TIM version number. The contents of this number are left to the manufacturer, but they shall be sufficient to allow the user to determine that something within the device has been changed even though the model number has not changed.

7.1.6.2 TIM sleep

This command puts the addressed TIM into a low-power state where it only responds to the wake-up command.

This command has no arguments and does not generate a reply.

7.1.6.3 Store state

Argument attribute name: StorState[N] $0 \leq N \leq 255$ data type UInt8

This command is used to cause a TIM and all TransducerChannels in the TIM to store their state information and enough additional information that it can be restored to the same setup it had when this command was received. Two conditions will cause the state to be restored. They are as follows:

— Receipt of the recall state command

Copyright © 2007 IEEE. All rights reserved.

— Return from a power failure

NOTE—This should include the signal conditioning parameters, the operation modes, and so on.

There is a single argument to this command of type UInt8. This argument shall be used to identify the state if multiple states are to be stored. The power-up state shall be state zero. There is no reply to this command.

7.1.6.4 Recall state

Argument attribute name: StorState[N] 0≤N≤255 data type UInt8

This command is used to return the setup of a TIM and all TransducerChannels in the TIM to the state they were in when the Store State command that defined the state specified by the argument to this command was received.

There is a single argument to this command of type UInt8. This argument shall be used to identify the state to be recalled. There is no reply to this command.

7.1.6.5 Read the IEEE 1451.0 version

Reply argument attribute name: StandardVersion data type UInt8

This command is used to read the version of this standard supported by this TIM.

This command has no arguments. The reply to this command shall have a single octet of the IEEE 1451.0 version number as shown in Table 37 in the data field.

Table 37—Enumeration of IEEE 1451.0 version numbers

Standard version	IEEE 1451.0 standard version
0	Reserved for prototype or other nonstandard IEEE 1451.0 versions
1	This will correspond to the original release of this standard
2–127	Reserved for future releases of the standard
128–255	Open to manufacturers

7.1.7 TIM any state commands

The following commands may be executed when the TIM is in any state. Table 38 gives a list of the commands in this class.

All commands in this class require a destination TransducerChannel number of zero. If destination TransducerChannel number in the message is not zero, the command rejected bit in the TransducerChannel Status-Condition Register (see 5.13.4) shall be set and the command shall be ignored.

Table 38—TIM any state commands

cmdFunctionId	Command	Address class		Reply expected	Required/ optional
		TIM	Global		
0	Reserved	—	—	—	—
1	Reset	Yes	No	No	Optional
2–127	Reserved	—	—	—	—
128–255	Open for manufacturers	—	—	—	—

Copyright © 2007 IEEE. All rights reserved.

7.1.7.1 Reset

This command is used to reset the TIM and all TransducerChannels associated with that TIM.

After a Reset command is issued to a TIM, the TIM and all TransducerChannels associated with that TIM shall go into the initialization/boot-up state immediately. If a state or states have been stored using the Store State (see 7.1.6.3) command, then state 0 will be the state restored during initialization.

This command has no arguments and does not produce a reply.

7.2 Manufacturer-defined commands

A manufacturer may include nonstandard commands that need to be exposed to the user by implementing a Commands TEDS as described in 5.5.2.5. The user is responsible for the software necessary to use these commands.

8. TEDS specification

This clause specifies the contents of all TEDS defined in this standard.

8.1 General format for TEDS

All TEDS have the general format shown in Table 39. The first field in any TEDS is the TEDS length. It is a 4 octet unsigned integer. The next block is the information content for the TEDS. Depending on the TEDS, the information may be binary information or it may be text-based. The last field in any TEDS is a checksum that shall be used to verify the integrity of the TEDS.

Table 39—Generic format for any TEDS

Field	Description	Type	# octets
—	TEDS length	UInt32	4
1 to N	Data block	Variable	Variable
—	Checksum	UInt16	2

8.1.1 TEDS length

Data type: unsigned integer used for field length (UInt32, 4 octets)

The TEDS length is the total number of octets in the TEDS data block plus the 2 octets in the checksum.

8.1.2 Data block

Data type: a structure defined by the specific TEDS

This structure contains the information that is stored in a specific TEDS. The fields that comprise this structure are different for each TEDS type. All TEDS prepared by a transducer manufacturer use a Type/Length/Value (TLV) data structure. In the case of text-based TEDS, this Type/Length/Value (TLV)

Copyright © 2007 IEEE. All rights reserved.

data structure is used to provide a directory to give access into different sections of the text portion of the TEDS that uses XML for the information content.

Using the TLV construct as shown in Table 40, each entry is stored as a TLV tuple. The "Type" field is a 1 octet tag that identifies the TLV, similar in function to HTML or XML tags. The "Length" field specifies the number of octets of the value field, and the "Value" field is the actual data. Each entry may be composed of one or more TLVs. The structure, or data type, of the value field is defined in the specification for the TEDS in this standard. See Table 43 for an example.

Table 40—Definition of the Type/Length/Value structure

Field	Description
Type	This code identifies the field in the TEDS that is contained within the value field. Except for types 2 and 3, the same number in the type field will have a different meaning in each different TEDS.
Length	The number in this field gives the number of octets in the value field. The number of octets in the length field is controlled by an entry in the TEDS Identification TLV.
Value	This field contains the TEDS information.

8.1.3 Unused type codes

Within the definition of each TEDS, some type codes are not used in that TEDS. These type codes are listed within the definition of each TEDS as either "reserved" or "open to manufacturers."

8.1.3.1 Reserved types

Type codes listed as "reserved" are reserved by the Common Functionality and TEDS Working Group for future revisions to the standard. They shall not be used by manufacturers or other groups.

8.1.3.2 Open to manufacturers

Type codes listed as "open to manufacturers" may be used by manufacturers to implement features that are not defined in the standard. If a manufacturer chooses to implement type codes in a transducer module that are not described in the standard, and that device is operated in a system that does not recognize the manufacturer specific type fields, then all of the features described in the standard shall function normally, but the additional manufacturer's features will not be supported.

8.1.4 IEEE Std 1451.2-1997 compatibility

The TEDS for IEEE Std 1451.2-1997 do not use TLV tuples. However, the first octet following the length field in the Meta-TEDS always contains the number two. Since the first octet following the length field in any IEEE 1451.0 TEDS is always a type code, the type code 2 is reserved and shall not be used. Since the Meta-TEDS is the only TEDS in IEEE Std 1451.2-1997 that contains TEDS version information, it is necessary when using that standard to read the Meta-TEDS before attempting to read any other TEDS.

8.1.5 IEEE Std 1451.3-2003 compatibility

The TEDS for IEEE Std 1451.3-2003 do not use TLV tuples. However, the first octet following the length field in the Meta-TEDS always contains the number one. Since the first octet following the length field in any IEEE 1451.0 TEDS is always a type code, the type code 1 is reserved and shall not be used. Since the Meta-TEDS is the only TEDS in IEEE Std 1451.3-2003 that contains TEDS version information, it is necessary when using that standard to read the Meta-TEDS before attempting to read any other TEDS.

Copyright © 2007 IEEE. All rights reserved.

8.1.6 Checksum

Data type: unsigned 16-bit integer (UInt16, 2 octets)

The checksum shall be the one's complement of the sum (modulo 2^{16}) of all preceding octets, including the initial TEDS length field and the entire TEDS data block. The checksum calculation excludes the checksum field.

$$\text{checksum} = 0\text{xFFFF} - \sum_{i=1}^{\text{TotalOctets}-2} \text{TEDSOctet}(i) \tag{4}$$

The one's complement of the sum N is $(2^{16} - 1) - N$. The checksum may be calculated by taking the hex value 0xFFFF minus the sum from the first octet to the octet before the checksum as shown in Equation (4). Another way to calculate the ones complement value of a number is by inverting the number's digits.

NOTE—Computing the checksum starts with adding the individual octets while keeping the sum in a 16 bit number. If the 16 bit number overflows, ignore the overflow and keep only the lower 16 bits. Take the logical (one's) complement of the resulting 16 bit number.

8.2 Order of octets in numeric fields

For numeric values requiring more than 1 octet, the first octet following the tuple length field shall be the most significant octet. The last field shall contain the least significant octet.

8.3 TEDS identification header

Field Type: 3

Field Name: TEDSID

Default value: Not applicable. This field is required in all TEDS.

The TEDS Identifier consists of the four fields shown in Table 41 and is standard for all TEDS. This field is always the first one in the TEDS. The Tuple Length for this field is assumed to be one.

The contents are as follows:

IEEE 1451 standards family number (0 for this standard)

TEDS class

Version number

Tuple length

Copyright © 2007 IEEE. All rights reserved.

Table 41—TEDS identifier structure

Field	Contents	Function
Type	03	Type field for the TEDS Identifier.
Length	04	This field is always set to 04, indicating that the value field contains 4 octets.
Family	00	This field identifies the member of the IEEE 1451 family of standards that defines this TEDS.
Class	See Table 17	This field identifies the TEDS being accessed. The value is the TEDS access code found in Table 17.
Version	See Table 42	This field identifies the TEDS version. The value is the version number identified in the standard. A value of zero in this field indicates that the TEDS do not conform to any released standard. Table 42 lists the allowable values for this field.
Tuple Length	Number of octets	This field gives the number of octets in the length field of all tuples in the TEDS except this tuple. NOTE—For most TEDS, the number of octets in the length field of the tuples is one, meaning that there are 255 or less octets in the value field. However, some cases may require more than 8 bits for the number of octets in the value field, so this field specifies the number of octets in the length field of a tuple. All tuples within a TEDS, except the TEDS Identifier, shall have the same number of octets in the length field.

Table 42—Enumeration of TEDS version numbers

TEDS version	IEEE 1451.0 standard version
0	Reserved for prototype or other nonstandard TEDS versions.
1	This will correspond to the original release of this standard.
2–255	Reserved for future releases of the standard.

8.4 Meta-TEDS

The Meta-TEDS is a required TEDS. The function of the Meta-TEDS shall be to make available at the interface all information needed to gain access to any TransducerChannel, plus information common to all TransducerChannels.

8.4.1 Access

The Meta-TEDS is accessed using a Query TEDS command, a Read TEDS segment command, a write TEDS segment command, or an Update TEDS command. The argument of the command shall specify the TEDS access code of the Meta-TEDS, as defined in Table 17.

This TEDS should be implemented as a read-only TEDS to prevent it from being changed in the field because changes could cause unpredictable behavior. If it is implemented as a read-only TEDS, the write TEDS segment command and Update TEDS command shall not apply.

8.4.2 Data block

Table 43 summarizes the content of the data block. Subordinate subclauses explain each data field in this data block. The length and checksum fields are not technically part of the TEDS data block but are shown in Table 43 to more completely describe the TEDS (see 8.1).

Copyright © 2007 IEEE. All rights reserved.

Table 43—Structure of the Meta-TEDS data block

Field type	Field name	Description	Data type	# octets
—		Length	UInt32	4
0–2	—	Reserved	—	—
3	TEDSID	TEDS Identification Header	UInt8	4
4	UUID	Globally Unique Identifier	UUID	10
5–9	—	Reserved	—	—
Timing-related information				
10	OholdOff	Operational time-out	Float32	4
11	SHoldOff	Slow-access time-out	Float32	4
12	TestTime	Self-Test Time	Float32	4
Number of implemented TransducerChannels				
13	MaxChan	Number of implemented TransducerChannels	UInt16	2
14	CGroup	ControlGroup information sub-block	—	—
Types 20, and 21 define one ControlGroup.				
20	GrpType	ControlGroup type	UInt8	1
21	MemList	ControlGroup member list	array of UInt16	NTc
15	VGroup	VectorGroup information sub-block	—	—
Types 20 and 21 define one VectorGroup.				
20	GrpType	VectorGroup type	UInt8	1
21	MemList	VectorGroup member list	array of UInt16	NTv
16	GeoLoc	Specialized VectorGroup for geographic location	—	—
Types 24, 20, and 21 define one set of geographic location information.				
24	LocEnum	An enumeration defining how location information is provided	UInt8	1
20	GrpType	VectorGroup type	UInt8	1
21	MemList	VectorGroup member list	array of UInt16	NTv
17	Proxies	TransducerChannel proxy definition sub-block	—	—
Types 22, 23, and 21 define one TransducerChannel proxy.				
22	ChanNum	TransducerChannel number of the TransducerChannel proxy	UInt16	1
23	Organiz	TransducerChannel proxy data-set organization	UInt8	1
21	MemList	TransducerChannel proxy member list	array of UInt16	NTp
18–19	—	Reserved	—	—
25–127	—	Reserved	—	—
128–255	—	Open to manufacturers	—	—
—		Checksum	UInt16	2

8.4.2.1 TEDS identification header

The TEDS Identification Header shall be as defined in 8.3.

This field is a required field. If this field is omitted or contains illegal values, the NCAP shall report a fatal TEDS error.

8.4.2.2 Globally unique identifier

Field Type: 4

Field Name: UUID

Copyright © 2007 IEEE. All rights reserved.

Data type: UUID, 10 octets

This field shall be present in the Meta-TEDS of each TIM and shall be unique (as shown in Figure 13). If this field is omitted, the NCAP shall report a fatal TEDS error.

See 4.12 for a description of the contents of this field.

8.4.2.3 Worst-case time-out values

Two time-out values are provided in this TEDS. Both values are intended for detecting nonresponsive TIMs. Two time-out values are specified to allow for two classes of response times. The operational time-out is used for most operations and is the shorter of the two periods. The second time-out value, slow access time-out, is used for commands that are expected to take longer to execute such as commands that write non-volatile memory.

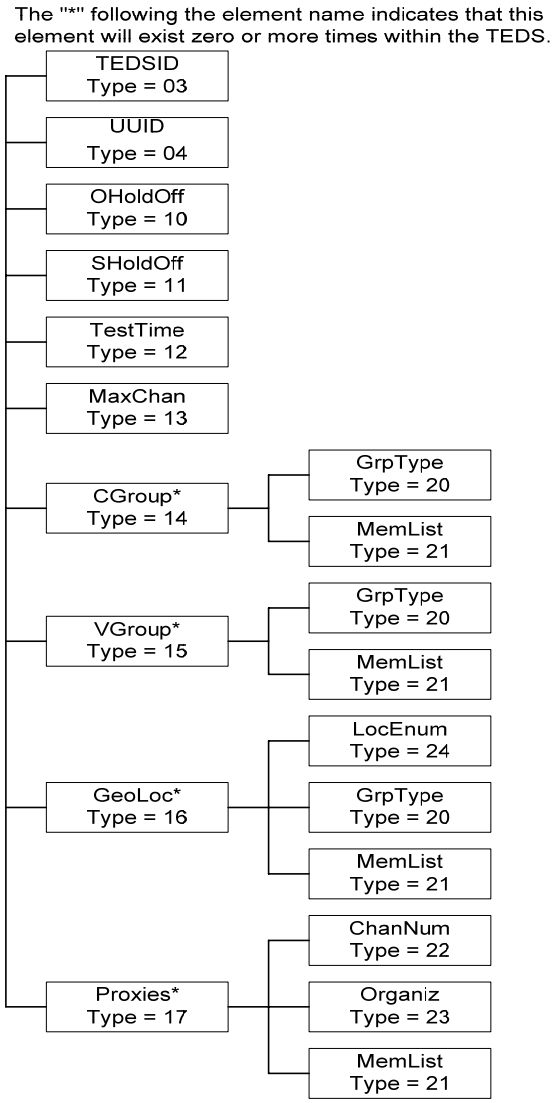


Figure 13—Structure of the Meta-TEDS data block

Copyright © 2007 IEEE. All rights reserved.

8.4.2.4 Operational time-out

Field Type: 10

Field Name: OHoldOff

Data type: single-precision real (Float32, 4 octets)

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

The Operational time-out field contains the time interval, in seconds, after an action for which the lack of reply following the receipt of a command may be interpreted as a failed operation. This may be the longer of the time required to begin a response to the slowest-to-execute command supported by the device (for example, worst-case command response) OR the time by which the device should be ready to accept the next command when no reply is required by the current command.

This field only gives the portion of the time that is used within the transducer module. Additional delays will exist within the NCAP or host processor and in the transmission that are related to the implementation, which shall be added to this value to produce a time-out value.

8.4.2.5 Slow access time-out

Field Type: 11

Field Name: SHoldOff

Data type: single-precision real (Float32, 4 octets)

The slow access time-out field contains the time interval, in seconds, after which an action for which the lack of a reply following the receipt of a command may be interpreted as a failed operation. This may be the longer of the time required to begin a response to the slowest-to-execute of these commands supported by the device (for example, worst-case command response) OR the time by which the device should be ready to accept the next command when no reply is required by the current one. The commands that are expected to require the longer time-out are identified in the description of the individual commands (see Clause 7).

This field is optional. If a single time-out is sufficient to allow the NCAP to operate without being excessively slow, then this field may be omitted.

This field only gives the portion of the time that is used within the transducer module. Additional delays will exist within the NCAP or host processor and in the transmission that are related to the implementation, which shall be added to this value to produce a time-out value.

8.4.2.6 Transducer module self-test time requirement

Field Type: 12

Field Name: TestTime

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error

Data type: single-precision real (Float32, 4 octets)

Copyright © 2007 IEEE. All rights reserved.

This field contains the maximum time, in seconds, required to execute the self-test. If no self-test is implemented, this field is zero.

8.4.2.7 Number of implemented TransducerChannels

Field Type: 13

Field Name: MaxChan

Data type: unsigned 16 bit integer (UInt16, 2 octet)

This field is required for all TIMs. If this field is omitted, the NCAP shall report a fatal TEDS error.

This field contains the number of TransducerChannels implemented in this TIM. TransducerChannels shall be numbered starting at one and be contiguous up to this number. No TransducerChannel numbers may be skipped.

8.4.2.8 ControlGroups

Field Type: 14

Field Name: CGroup

Data type: unsigned octet integer (UInt8, 1 octet)

This field is required for TIMs that implement ControlGroups. This field is omitted if no ControlGroups exist in this transducer module.

ControlGroups identify transducers that are used to control the operation of other transducers as shown in Table 44. The ControlGroup definition is a hierarchical structure with the following subfields (TLV tuples):

Group type

Member list

In Table 44, the Enumeration column contains the value that will be placed in the group type field to identify the particular ControlGroup that is being defined. The function column defines the function of each TransducerChannel in the group. The member list is a list of the TransducerChannel numbers for the Transducers that perform each function. The numbers in the member list order column define the order in which the TransducerChannel numbers shall be listed. The member list is an ordered array, and no elements may be omitted. If a TransducerChannel is not required, then the TransducerChannel number for that function shall be zero.

Enumeration 2 may be used to identify the embedded actuator TransducerChannels used to set the high-pass filter, low-pass filter, and scale factor associated with a sensor of any type.

Enumeration 3 is used to identify an embedded actuator TransducerChannel that may be used to set the TransducerChannel sampling period.

Enumeration 8 is used when the requirements for simultaneous sampling require that a programmable delay be introduced to make the delays equal for all TransducerChannels within the group.

Copyright © 2007 IEEE. All rights reserved.

Enumerations 1 and 4 may be used to identify the embedded actuator TransducerChannels used to set up an event sensor. They also identify a sensor TransducerChannel that may be used to read the level of the signal in an analog event sensor or the current pattern input to a digital event sensor.

Table 44—Enumeration of ControlGroup types

Enumeration	Member list order	Function
0		Reserved
1	1	Analog event sensor TransducerChannel
	2	Analog input sensor TransducerChannel that measures the input value for the same input as the member 1 event sensor provides the state
	3	Upper threshold embedded actuator TransducerChannel
	4	Hysteresis embedded actuator TransducerChannel
2	1	Sensor TransducerChannel (any type)
	2	High-pass filter embedded actuator TransducerChannel
	3	Low-pass filter embedded actuator TransducerChannel
	4	Scale factor embedded actuator TransducerChannel
3	1	TransducerChannel (any type)
	2	Sample interval embedded actuator TransducerChannel
4	1	Digital event sensor TransducerChannel
	2	Digital input sensor TransducerChannel that measures the input value for the same input as the member 1 event sensor provides the state
	3	Event pattern embedded actuator TransducerChannel
5	1	Time interval sensor TransducerChannel
	2	TransducerChannel number of the transducer that causes the output of the time interval sensor to be latched
6	1	TimeInstance sensor TransducerChannel
	2	TransducerChannel number of the transducer that causes the output of the TimeInstance sensor to be latched
7	1	TransducerChannel number of an event sensor used to trigger other transducers
	2 - N	The remaining N entries give a list of TransducerChannels triggered by the event
8	1	TransducerChannel (any type)
	2	Embedded time delay actuator TransducerChannel
9	1	Location sensor TransducerChannel
	2	TransducerChannel number of the transducer that causes the output of the location sensor to be latched
10–127		Reserved for future expansion
128–255		Open to manufacturers

8.4.2.9 Group type

Field Type: 20

Field Name: GrpType

Data type: unsigned octet integer (UInt8, 1 octet)

This field is required for TIMs that implement ControlGroups. This field is omitted if no ControlGroups exist in this transducer module.

The group type is defined in the enumeration column in Table 44 for ControlGroups or in Table 45 for VectorGroups.

Copyright © 2007 IEEE. All rights reserved.

8.4.2.10 Member list

Field Type: 21

Field Name: MemList

Data type: a one-dimensional array of unsigned 16 bit integers (UInt16, 2 to 510 octets)

This field is required for TIMs that implement ControlGroups and VectorGroups. This field is omitted if no ControlGroups or VectorGroups exist in this transducer module.

The group type is defined in the value column in Table 44 for ControlGroups or in Table 45 for VectorGroups.

This field is a list of transducer channel numbers that go to make up the ControlGroup. They are in the order that is specified in Table 44 or Table 45.

8.4.2.11 VectorGroups

Field Type: 15

Field Name: VGroup

Data type: unsigned octet integer (UInt8, 1 octet)

This field is required for TIMs that implement VectorGroups. This field is omitted if no VectorGroups exist in this transducer module.

VectorGroups identify the relationships between the data sets within a transducer module as shown in Table 45. The VectorGroup definition is a hierarchical structure with the following subfields:

Group type

Member list

In Table 45, the Enumeration column contains the value that will be placed in the group type field to identify the particular VectorGroup that is being defined. The function column defines the function of each TransducerChannel in the group. The member list is a list of the TransducerChannel numbers for the Transducers that perform each function. The numbers in the member list order column define the order in which the TransducerChannel numbers shall be listed. The member list is an ordered array, and no elements may be omitted. If a TransducerChannel is not required, then the TransducerChannel number for that function shall be zero.

8.4.2.12 Group type

This field is the same as defined in 8.4.2.9.

This field is required for TIMs that implement VectorGroups. This field is omitted if no VectorGroups exist in this transducer module.

Copyright © 2007 IEEE. All rights reserved.

8.4.2.13 Member list

The structure of this field is the same as defined in 8.4.2.10.

This field is required for TIMs that implement VectorGroups. This field is omitted if no VectorGroups exist in this transducer module.

This field is a list of transducer channel numbers that go to make up the VectorGroup. They are in the order that is specified in Table 45.

Table 45—Enumeration of VectorGroup types

Enumeration	Member list order	Function
0	—	An arbitrary relation
1	1 2 3	x component of a right-hand rectangular spatial vector y component of a right-hand rectangular spatial vector z component of a right-hand rectangular spatial vector
2	1 2 3	ρ component of a right-hand cylindrical spatial vector ϕ component of a right-hand cylindrical spatial vector z component of a right-hand cylindrical spatial vector
3	1 2 3	r component of a right-hand spherical spatial vector θ component of a right-hand spherical spatial vector ϕ component of a right-hand spherical spatial vector
4	1 2 3	Latitude component of a planetary coordinate system Longitude component of a planetary coordinate system Altitude component of a planetary coordinate system
5	1 2	In-phase component of a two-dimensional vector Quadrature component of a two-dimensional vector
6	1 2 3	Red component in a color vector Green component in a color vector Blue component in a color vector
7	1 2	Real component of a complex number Imaginary component of a complex number
8–127		Reserved for future expansion
128–255		Open to manufacturers

8.4.2.14 Geographic location group

Field Type: 16

Field Name: GeoLoc

This field is required for TIMs that implement dynamic location information as specified in Table 46. If this field is omitted, the NCAP shall assume that this TIM does not provide location information.

The geographic location group is a hierarchical structure with the following subfields (TLV tuples):

LocEnum

Group type

Member list

This field implements a specialized VectorGroup type that is used to provide dynamic geographic location information.

Copyright © 2007 IEEE. All rights reserved.

8.4.2.15 Location enumeration

Field Type: 24

Field Name: LocEnum

Data type: unsigned octet integer (UInt8, 1 octet)

This field is an optional field. If this field is omitted, the NCAP shall assume that this TIM does not provide any geographic location information.

The values for TransducerChannel type key are defined in Table 46.

Table 46—Enumeration of geographic location types

Value	Meaning
0	No geographic location information is provided by this TIM
1	Static geographic location information is provided via the geographic location TEDS
2	Dynamic geographic location information is provided
3	Dynamic geographic location information is provided that is relative to the location specified in the geographic location TEDS
4–255	Reserved for future expansion

8.4.2.16 Group type

This is the same as defined in 8.4.2.12 except that the group type shall be limited to types 1–4.

This field is required if the location enumeration is 2 or 3. This field is omitted if no geographic location is provided by this TIM.

8.4.2.17 Member list

The structure of this field is the same as defined in 8.4.2.10.

This field is required if the location enumeration is 2 or 3. This field is omitted if no geographic location is provided by this TIM.

This list of transducer channel numbers makes up the VectorGroup. They are in the order that is specified in Table 45 for enumerations 1 through 4.

8.4.2.18 TransducerChannel proxies

Field Type: 17

Field Name: Proxies

This field is required for TIMs that implement TransducerChannel proxies. This field is omitted if no TransducerChannel proxies exist in this transducer module.

NOTE—If this field is omitted and one or more TransducerChannel proxies exist within the TIM, TransducerChannels will appear to exist without TransducerChannel TEDS, which is a fatal error.

Copyright © 2007 IEEE. All rights reserved.

A TransducerChannel proxy is an artificial construct used to combine the outputs of multiple sensors or the input to multiple actuators into a single structure. A TransducerChannel proxy has a TransducerChannel number and may be triggered, read, or written, but it does not have the other characteristics of a TransducerChannel. TransducerChannel Proxies do not have TransducerChannel TEDS, calibration TEDS, Frequency Response TEDS, or Transfer Function TEDS. However, there may be text-based TEDS, End-user application-specific TEDS, or users transducer name TEDS associated with a proxy.

The TransducerChannel Proxy TLV is made up of three subfields as follows:

TransducerChannel number of the TransducerChannel proxy

TransducerChannel proxy data set organization

TransducerChannel proxy member list

8.4.2.19 TransducerChannel number of the TransducerChannel proxy

Field Type: 22

Field Name: ChanNum

Data type: unsigned 16 bit integer (UInt16, 2 octets)

This field is required for TIMs that implement TransducerChannel proxies. This field is omitted if no TransducerChannel proxies exist in this transducer module.

This field contains the transducer channel that will be used when addressing this proxy. It may be used to read data from a sensor proxy, to write data to an actuator proxy, or to trigger all transducers represented by the proxy.

8.4.2.20 TransducerChannel proxy data-set organization

Field Type: 23

Field Name: Organiz

Data type: unsigned octet integer (UInt8, 1 octet)

This field is required for TIMs that implement TransducerChannel proxies. This field is omitted if no TransducerChannel proxies exist in this transducer module.

As shown in Table 47, there are two methods of combining the data sets to or from multiple transducers. The block method of combining shall be used if the data sets from the different transducers contain different numbers of words. The Block method of combining data sets is shown graphically on the left side of Figure 6. The interleave method of combining data sets is shown on the right side of Figure 6. The first entry in the combined data set, “Data sample from transducer X”, is only present in Interleave method 2 and is expected to be some type of a time-tag to allow the NCAP to figure out when the first sample in the data set was taken for some sensor types. The time of the remaining samples can usually be determined from the characteristics of the sensors if the time of the first sample is known. If interleave method 1 is used, the “Data sample from Transducer X” is omitted. The need for this time tagging capability exists in the block method also, but no special characteristics are required to implement it with the block method.

Copyright © 2007 IEEE. All rights reserved.

Table 47—Enumerations for combined data sets

Enumeration	Meaning
0	Block method.
1	Interleave method 1.
2	Interleave method 2: This is the same as Interleave method 1 except that the list of interleaved TransducerChannels shall be preceded by a one-sample data set from another TransducerChannel or proxy. This is typically used for a time stamp identifying the time of the first sample in the combined data set.
3–255	Reserved.

8.4.2.21 TransducerChannel proxy member list

The structure of this field is the same as defined in 8.4.2.10.

This list of transducer channel numbers is used to make a proxy. They are in the order that their data will appear in the proxy. If the block method is being used, the channel number for the first block of data will be listed first, the channel number for the second block of data is next, and so forth up to the data for block *N*. If one of the interleave methods is used, the first entry in the list will be the channel number for transducer channel *X*, if it is required, followed by the channel number for what in Figure 6 is transducer 1, and so forth up to the channel number for channel *N*.

8.5 TransducerChannel TEDS

This is a required TEDS. The function of the TransducerChannel TEDS shall be to make available at the interface all of the information concerning the TransducerChannel being addressed to enable the proper operation of the TransducerChannel.

8.5.1 Access

The TransducerChannel TEDS is accessed using a Query TEDS command, a Read TEDS segment command, a write TEDS segment command, or an Update TEDS command. The argument of the command shall specify the TEDS access code of the TransducerChannel TEDS, as defined in Table 17.

This TEDS may be implemented as a read-only TEDS to prevent changes from being made in the field because changes could cause unpredictable behavior. If it is implemented as a read-only TEDS, the TransducerChannel write TEDS segment and TransducerChannel Update TEDS commands shall not apply.

8.5.2 Data block

Table 48 and Figure 14 show the information required to be in this TEDS. Subsequent subclauses explain each field in the structure.

8.5.2.1 TEDS identifier

The TEDS identifier shall comply with the structures defined in 8.3.

This field is a required field. If this field is omitted or contains illegal values, the NCAP shall report a fatal TEDS error.

Copyright © 2007 IEEE. All rights reserved.

8.5.2.2 Calibration key

Field Type: 10

Field Name: CalKey

Data type: unsigned octet integer (UInt8, 1 octet)

This field is a required field. If this field is omitted, the NCAP shall report a fatal TEDS error.

This field contains an enumeration that denotes the calibration capabilities of this TransducerChannel. Table 49 provides the list of enumerated values and their meanings. The column labeled "Name" defines manifest constants to symbolize the enumerated values, and these names are used throughout the remainder of this standard.

8.5.2.3 System corrections

Calibration key enumerations CAL_SUPPLIED or CAL_CUSTOM are to be used when the correction is performed in the NCAP, a host processor, or elsewhere in the system.

8.5.2.4 Transducer module corrections

Calibration key enumerations TIM_CAL_SUPPLIED and TIM_CAL_SELF are to be used when the correction is performed in the TIM using one of the correction methods specified in 8.6.1.1 and information stored in the Calibration TEDS (8.6.3). TIM_CAL_CUSTOM is used when the correction method is *not* one of the methods specified in 8.6.1.1.

8.5.2.5 TransducerChannel type key

Field Type: 11

Field Name: ChanType

Data type: unsigned octet integer (UInt8, 1 octet)

This field is a required field. If this field is omitted, the NCAP shall report a fatal TEDS error.

Copyright © 2007 IEEE. All rights reserved.

Table 48—Structure of the TransducerChannel TEDS data block

Field	Field name	Description	Type	# octets
—		TEDS length	UInt32	4
0–2	—	Reserved	—	—
3	TEDSID	TEDS Identification	UInt8	4
4-9	—	Reserved	—	—
TransducerChannel related information				
10	CalKey	Calibration key	UInt8	1
11	ChanType	TransducerChannel type key	UInt8	1
12	PhyUnits	Physical Units	UNITS	11
50	UnitType	Physical Units interpretation enumeration	UInt8	1
51	Radians	The exponent for Radians	UInt8	1
52	SterRad	The exponent for Steradians	UInt8	1
53	Meters	The exponent for Meters	UInt8	1
54	Kilogram	The exponent for Kilograms	UInt8	1
55	Seconds	The exponent for Seconds	UInt8	1
56	Amperes	The exponent for Amperes	UInt8	1
57	Kelvins	The exponent for Kelvins	UInt8	1
58	Moles	The exponent for Moles	UInt8	1
59	Candelas	The exponent for Candelas	UInt8	1
60	UnitsExt	TEDS access code for units extension	UInt8	1
13	LowLimit	Design operational lower range limit	Float32	4
14	HiLimit	Design operational upper range limit	Float32	4
15	OError	Worst-case uncertainty	Float32	4
16	SelfTest	Self-test key	UInt8	1
17	MRange	Multi-range capability	UInt8	1
—		Data converter-related information	—	—
18	Sample		—	—
40	DatModel	Data model	UInt8	1
41	ModLenth	Data model length	UInt8	1
42	SigBits	Model significant bits	UInt16	2
19	DataSet			
43	Repeats	Maximum data repetitions	UInt16	2
44	SOrigin	Series origin	Float32	4
45	StepSize	Series increment	Float32	4
46	SUnits	Series units	UNITS	11
47	PreTrigg	Maximum pre-trigger samples	UInt16	2
Timing-related information				
20	UpdateT	TransducerChannel update time (t_u)	Float32	4
21	WSetupT	TransducerChannel write setup time (t_{ws})	Float32	4
22	RSetupT	TransducerChannel read setup time (t_{rs})	Float32	4
23	SPeriod	TransducerChannel sampling period (t_{sp})	Float32	4
24	WarmUpT	TransducerChannel warm-up time	Float32	4
25	RDelayT	TransducerChannel read delay time (t_{ch})	Float32	4
26	TestTime	TransducerChannel self-test time requirement	Float32	4
Time of the sample information				
27	TimeSrc	Source for the time of sample	UInt8	1
28	InPropDl	Incoming propagation delay through the data transport logic	Float32	4
29	OutPropD	Outgoing propagation delay through the data transport logic	Float32	4
30	TSError	Trigger-to-sample delay uncertainty	Float32	4
Attributes				
31	Sampling	Sampling attribute	—	—
48	SampMode	Sampling mode capability	UInt8	1
49	SDefault	Default sampling mode	UInt8	1
32	DataXmit	Data transmission attribute	UInt8	1
33	Buffered	Buffered attribute	UInt8	1
34	EndOfSet	End-of-data-set operation attribute	UInt8	1
35	EdgeRpt	Edge-to-report attribute	UInt8	1
36	ActHalt	Actuator-halt attribute	UInt8	1

Copyright © 2007 IEEE. All rights reserved.

Table 48—Structure of the TransducerChannel TEDS data block (*continued*)

Field	Field name	Description	Type	# octets
Sensitivity				
37	Directon	Sensitivity direction	Float32	4
38	DAngles	Direction angles	Two Float32	8
Options				
39	ESOption	Event sensor options	UInt8	1
61–127	—	Reserved	—	—
128–255	—	Open to manufacturers	—	—
—		Checksum	UInt16	2

Table 49—Enumeration of calibration keys

Value	Calibration capability	Name
0	No calibration information needed or provided by the transducer module. This implies that there is no Calibration TEDS associated with this TransducerChannel. If an attempt is made to access the Calibration TEDS, the Calibration TEDSLength shall be zero.	CAL_NONE
1	Calibration information is provided as a Calibration TEDS, as specified in 8.5.2.55. Correction is performed outside of the TIM.	CAL_SUPPLIED
2	Reserved.	
3	Calibration information is provided but in a form that is not described in this standard. Correction is performed outside of the TIM.	CAL_CUSTOM
4	Calibration information is provided as a Calibration TEDS, as specified in 8.5.2.55. Correction is applied in the TIM.	TIM_CAL_ SUPPLIED
5	Calibration information is provided as a Calibration TEDS, as specified in 8.5.2.55, and is applied in the TIM. The calibration information is adjusted by a self-calibration capability.	TIM_CAL_SELF
6	Calibration information is provided but in a form that is not described in this standard. It may be a manufacturer-defined TEDS. Correction is performed in the TIM. If an attempt is made to access the Calibration TEDS, the Calibration TEDSLength shall be zero.	TIM_CAL_CUSTOM
7–255	Reserved.	—

The values for TransducerChannel type key are defined in Table 50.

Table 50—Enumeration of TransducerChannel types

Value	Meaning
0	Sensor
1	Actuator
2	Event sensor
3–255	Reserved for future expansion

Copyright © 2007 IEEE. All rights reserved.

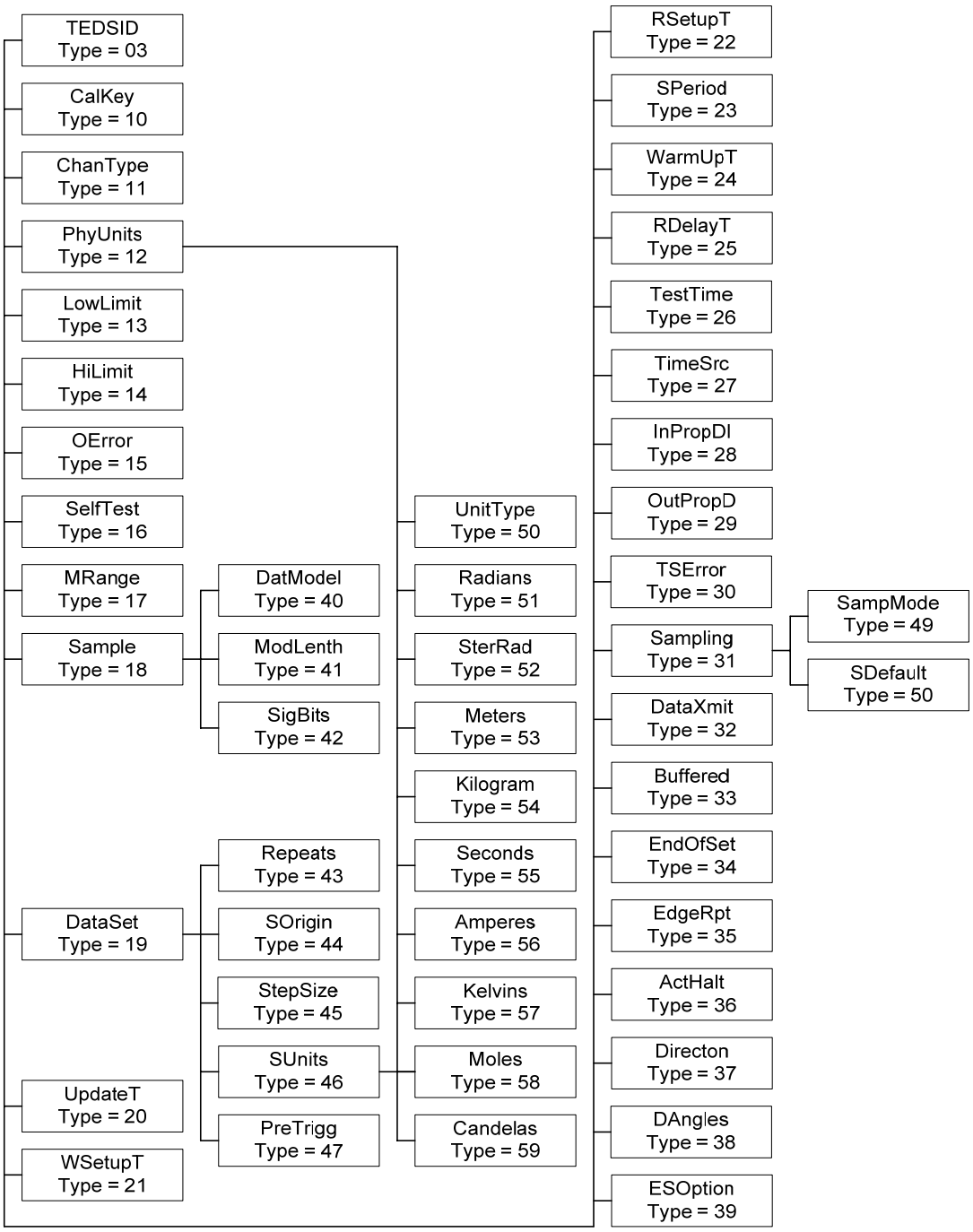


Figure 14—TransducerChannel TEDS

8.5.2.6 Physical Units

Field Type: 12

Field Name: PhyUnits

Copyright © 2007 IEEE. All rights reserved.

This field is required for all TransducerChannel TEDS. If this field is omitted, the NCAP shall report a fatal TEDS error.

The Physical Units field is used to define the SI units for the physical quantity that is being measured or controlled.

If the exponent of any of the fields described in 8.5.2.8 through 8.5.2.16 is zero, the field may be omitted. In these cases the default value shall be 128.

See 4.11or Annex J for details on how to construct the value placed in the Physical Units fields.

8.5.2.7 Physical Units interpretation enumeration

Field Type: 50

Field Name: UnitType

Data type: Unsigned 8 bit integer (UInt8, 1 octet).

This field is required for all TransducerChannel TEDS. If this field is omitted, the NCAP shall report a fatal TEDS error.

This field is described in 4.11.

8.5.2.8 The exponent for Radians

Field Type: 51

Field Name: Radians

Data type: Unsigned 8 bit integer (UInt8, 1 octet).

This field is described in 4.11.

8.5.2.9 The exponent for Steradians

Field Type: 52

Field Name: Steradians

Data type: Unsigned 8 bit integer (UInt8, 1 octet).

This field is described in 4.11.

8.5.2.10 The exponent for Meters

Field Type: 53

Field Name: Meters

Data type: Unsigned 8 bit integer (UInt8, 1 octet).

Copyright © 2007 IEEE. All rights reserved.

This field is described in 4.11.

8.5.2.11 The exponent for Kilograms

Field Type: 54

Field Name: Kilograms

Data type: Unsigned 8 bit integer (UInt8, 1 octet).

This field is described in 4.11.

8.5.2.12 The exponent for Seconds

Field Type: 55

Field Name: Seconds

Data type: Unsigned 8 bit integer (UInt8, 1 octet).

This field is described in 4.11.

8.5.2.13 The exponent for Amperes

Field Type: 56

Field Name: Amperes

Data type: Unsigned 8 bit integer (UInt8, 1 octet).

This field is described in 4.11.

8.5.2.14 The exponent for Kelvins

Field Type: 57

Field Name: Kelvins

Data type: Unsigned 8 bit integer (UInt8, 1 octet).

This field is described in 4.11.

8.5.2.15 The exponent for Moles

Field Type: 58

Field Name: Moles

Data type: Unsigned 8 bit integer (UInt8, 1 octet).

Copyright © 2007 IEEE. All rights reserved.

This field is described in 4.11.

8.5.2.16 The exponent for Candelas

Field Type: 59

Field Name: Candelas

Data type: Unsigned 8 bit integer (UInt8, 1 octet).

This field is described in 4.11.

8.5.2.17 TEDS access code for units extension

Field Type: 60

Field Name: UnitsExt

Data type: unsigned octet integer (UInt8, 1 octet)

This field is optional for TransducerChannel TEDS. If it is not implemented, the NCAP will assume that no units extension TEDS exists within this TransducerChannel.

This field is provided to allow a text-based extension to the Physical Units. It shall not be used as a subsutitute for the Physical Units. Since there can be more than one set of Physical Units in a TransducerChannel TEDS, the contents of this field shall provide the TEDS access code for the text-based TEDS that provides this extension. If a single units extension TEDS is required, it should use the TEDS access code identified in Table 17.

8.5.2.18 Design operational lower range limit

Field Type: 13

Field Name: LowLimit

Data type: single-precision real (Float32, 4 octets)

This field is required for all TransducerChannel TEDS. If this field is omitted, the NCAP shall report a fatal TEDS error.

For sensors, this shall be the lowest valid value for TransducerChannel data that the TransducerChannel is designed to provide after correction is applied. It shall be interpreted in the units specified by the Physical Units field of the TransducerChannel TEDS. If the corrected TransducerChannel data lie below this limit, it may not comply with TransducerChannel specifications set by the manufacturer.

NOTE—The design operational lower range limit is always expressed in SI units. If the Calibration TEDS is used, the output of the correction process may be converted into SI units by applying the SI units conversion constants supplied in the Calibration TEDS (see 8.6.1.6).

For actuators, this shall be the lowest valid value for TransducerChannel data that the TransducerChannel is designed to accept before correction is applied. It shall be interpreted in the units specified by the

Copyright © 2007 IEEE. All rights reserved.

Physical Units field of the TransducerChannel TEDS. Writing corrected TransducerChannel data below this limit may result in behavior outside the transducer module specifications set by the manufacturer.

NOTE—For TransducerChannels that use multiple inputs to produce a single output, this limit, which is expressed in terms of the output, will normally be a nominal value rather than a precise value.

In cases where no correction is applied and the TransducerChannel data model is not single-precision real, conversion of the TransducerChannel data to single-precision real (or conversion of the limit value to the data model of the TransducerChannel) is necessary before making the comparison.

NOTE—For example, this conversion is required when the Calibration key is CAL_NONE and the data model is N-octet integer. Note that this conversion may limit the practical range or precision of the converted TransducerChannel data.

When this parameter is not applicable, it shall be NaN (see 4.5.1).

NOTE—As an example of an application in which range limits do not apply, consider a bank of switches modeled as N-octet data. In this case, both range limit fields shall be set to NaN. This is not to say that range limits do not apply to N-octet data. For example, a 12 bit integer with no expressed units, such as raw ADC output, would also be modeled as N-octet data. In this case, range limits are applicable.

If the Maximum data repetitions field (see 8.5.2.28) of this TransducerChannel is non-zero, then the value of this field shall be interpreted to apply to all of the repetition instances.

8.5.2.19 Design operational upper range limit

Field Type: 14

Field Name: HiLimit

Data type: single-precision real (Float32, 4 octets)

This field is required for all TransducerChannel TEDS. If this field is omitted, the NCAP shall report a fatal TEDS error.

For sensors, this shall be the highest valid value for TransducerChannel data that the TransducerChannel is designed to provide after correction is applied. It shall be interpreted in the units specified by the Physical Units field of the TransducerChannel TEDS. If the corrected TransducerChannel data lies above this limit, it may not comply with transducer module specifications set by the manufacturer.

NOTE—The design operational upper range limit is always expressed in SI units. If the Calibration TEDS is used, the output of the correction process may be converted into SI units by applying the SI units conversion constants supplied in the Calibration TEDS (see 8.6.1.6).

For actuators, this shall be the highest valid value for TransducerChannel data that the TransducerChannel is designed to accept before correction is applied. It shall be interpreted in the units specified by the Physical Units field of the TransducerChannel TEDS. Writing corrected TransducerChannel data above this limit may result in behavior outside the TransducerChannel specifications set by the manufacturer.

NOTE—For TransducerChannels that use multiple inputs to produce a single output, this limit, which is expressed in terms of the output, will normally be a nominal value rather than a precise value.

In cases where no correction is applied and the TransducerChannel data model is not single-precision real, conversion of the TransducerChannel data to single-precision real (or conversion of the limit value to the data model of the TransducerChannel) is necessary before making the comparison.

Copyright © 2007 IEEE. All rights reserved.

When this parameter is not applicable, it shall be NaN (see 4.5.1).

If the Maximum data repetitions field (see 8.5.2.28) of this TransducerChannel is nonzero, then the value of this field shall be interpreted to apply to all of the repetition instances.

8.5.2.20 Uncertainty under worst-case conditions

Field Type: 15

Field Name: OError

Data type: single-precision real (Float32, 4 octets)

This field required for all TransducerChannel TEDS. If this field is omitted, the NCAP shall report a nonfatal TEDS error.

The "Combined Standard Uncertainty" defined in [B10], Appendix C, Section 2.2. The value of this field shall be expressed in the units specified in the Physical Units field of the TransducerChannel TEDS.

This field is used to describe the uncertainty that will exist in this TransducerChannel's output over the worst-case combination of variations in the environment and any other factors, such as power supply voltage, that could change the TransducerChannel output.

8.5.2.21 Self-test key

Field Type:16

Field Name: SelfTest

Data type: unsigned octet integer (UInt8, 1 octet)

This field is required for all TransducerChannel TEDS. If this field is omitted, the NCAP shall report a non-fatal TEDS error.

This field defines the self-test capabilities of the TransducerChannel as shown in Table 51.

Table 51—Enumeration of self-test keys

Value	Meaning
0	No self-test function needed or provided
1	Self-test function provided
2–255	Reserved for future expansion

8.5.2.22 Multi-range capability

Field Type: 17

Field Name: MRange

Data type: unsigned octet integer (UInt8, 1 octet)

This field is optional for TransducerChannel TEDS. If it is not implemented, the NCAP will assume that no multirange capability exists within this TransducerChannel.

Copyright © 2007 IEEE. All rights reserved.

The content of this field identifies whether any transducers in the TIM have the capability of being operated over different ranges. If the value is true, this TransducerChannel is capable of being operated in more than one range. Otherwise no multirange capability exists. If a multirange capability exists, a Commands TEDS is required. The Commands TEDS is required to define the commands needed to select the operating range.

8.5.2.23 Sample definition

Field Type: 18

Field Name: Sample

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

The sample definition field consists of three fields and is required for all TransducerChannel TEDS.

The subfields that make up this type are as follows:

Data model

Data model length

Model significant bits

8.5.2.24 Data model

Field Type: 40

Field Name: DatModel

Data type: unsigned octet integer (UInt8, 1 octet)

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

This field describes the data model used when issuing Read TransducerChannel data-set segment (see 7.1.3.1) or Write TransducerChannel data-set segment (see 7.1.3.2) commands to this TransducerChannel. Values are enumerated in Table 52.

There are two differences between the integer forms (enumeration 0 and 5) and fractional forms (enumeration 3 and 6):

The radix point (that divides integer from fractional bits) is to the right of the least significant bit for an N-octet integer or long integer. It is immediately to the right of the most significant bit for the N-octet fraction or long fraction. Justification of the significant bits differs, as explained in the Model significant bits field (8.5.2.26).

Copyright © 2007 IEEE. All rights reserved.

Table 52—Enumeration of data models

Value	Base data type	Model	Constraint on data model length
0	UInt8	N-octet integer (unsigned)	0≤N≤8
1	Float32	Single-precision real	N=4
2	double	Double-precision real	N=8
3	UInt8	N-octet fraction (unsigned)	0≤N≤8
4	UInt8	Bit sequence	0≤N≤8
5	UInt8	Long integer (unsigned)	9≤N≤255
6	UInt8	Long fraction (unsigned)	9≤N≤255
7	TimeInstance	Time of day	N=8
8–255	—	Reserved for future expansion	—

The use of enumerations 5 and 6 is expected to be rare. An NCAP is not expected to process data with these enumerations but may pass them on unprocessed to the entity requesting the data.

The N-octet fraction type may be used to keep the multinomial coefficients (see 8.6.3.22) within representable bounds and to avoid overflows when converting the data to Physical Units.

The Bit sequence data model is for collections of bits that have no numeric value. An example of this type is the position of each switch in a bank of switches. Bit significance is user defined.

Boolean data (such as {0,1} or {false, true}) shall be represented as a bit sequence type with a length of 1 octet. The representation of true and false is defined in 4.8.

The Time of day data model supports TimeInstance data types defined in 4.9.2.

8.5.2.25 Data model length

Field Type: 41

Field Name: ModLenth

Data type: unsigned octet integer (UInt8, 1 octet)

This field is required. This field may be omitted in the case of single-precision real, double-precision real, and TimeInstance data types since those types have fixed lengths. For all other cases, if this field is omitted, the NCAP shall report a fatal TEDS error.

This field contains the number of octets in the representation specified in the Data model field. Constraints on the model length are summarized in Table 52. A value of zero in the data model length field indicates the existence of a TransducerChannel that produces or uses no data.

8.5.2.26 Model significant bits

Field Type: 42

Field Name: SigBits

Data type: unsigned 16 bit integer (UInt16, 2 octets)

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

Copyright © 2007 IEEE. All rights reserved.

When the Data model is an N-octet integer (enumeration 0 or 5) or N-octet fraction (enumeration 3 or 6), the value of this field is the number of bits that are significant.

For example, if data from a TransducerChannel come from a 12-bit ADC, then:

Data model = 0 (N-octet integer)

Data model length = 2 (the number of octets needed to hold 12 bits)

Model significant bits = 12

When the Data model is an N-octet integer, N-octet fraction, long integer, or long fraction, the field Model significant bits shall not exceed eight times the Data model length.

When the Data model is an N-octet integer or long integer, the data bits shall be right justified within the octet stream. The field Model significant bits specifies the number of bits that are significant. A value of zero is illegal.

When the Data model is an N-octet fraction or long fraction, the data bits shall be left justified within the octet stream. The field Model significant bits specifies the number of bits that are significant.

When the Data model is single-precision or double-precision real (enumerations 1 or 2), the value of this field is the number of bits in the TransducerChannel's signal converter.

When the Data model is single-precision real, double-precision real, bit sequence, or time of day, the field Model significant bits is fixed and it may be ignored.

8.5.2.27 Data set definition

Field Type: 19

Field Name: DataSet

The data set definition field is required for all TransducerChannel TEDS.

This field is optional. If this field is omitted, the Maximum data repetitions and the Maximum pre-trigger samples shall be assumed to be zero..

The subfields that make up this type are as follows:

Maximum data repetitions

Series origin

Series increment

Series units

Maximum pre-trigger samples

8.5.2.28 Maximum data repetitions

Field Type: 43

Field Name: Repeats

Data type: unsigned octet integer (UInt16, 2 octets)

Copyright © 2007 IEEE. All rights reserved.

This field is optional. If this field is omitted, the maximum data repetitions shall be zero.

This field contains the number (L) of repetitions of the TransducerChannel value that may be produced or required by a single trigger above the initial sample. Each repetition represents an additional measurement or actuation value produced or consumed by the TransducerChannel at each trigger event, which shall be spaced apart from the initial value associated with the trigger along some axis (for example, time) by an amount defined by the TransducerChannel TEDS fields Series increment (8.5.2.30) and Series units (8.5.2.31), respectively. When L is zero, the values of Series origin, Series increment, and Series units may be ignored. The purpose of this structure is to allow TransducerChannels to produce a time series or a mass spectrum or to generate a waveform with the application of a single trigger.

In operation, the number of TransducerChannel values that may be produced or required by a single trigger may be set to any value lower than the contents of this field by issuing the Set TransducerChannel data repetition count command as defined in 7.1.2.1 if that command is implemented.

When reading or writing data with data repetition count greater than zero, the order of transmittal shall be with the zeroth (oldest) data sample transmitted first, the first repetition (next oldest) transmitted second, and so on.

8.5.2.29 Series origin

Field Type: 44

Field Name: SOrigin

Data type: single-precision real (Float32, 4 octets)

This field may be omitted if the Maximum data repetitions field (See 8.5.2.28) is zero. If the Maximum data repetitions field is nonzero and this field is omitted, the NCAP shall assume a value of zero for the Series origin.

For the case where the data repetition count is greater than zero, the Series origin represents the value of the independent variable associated with the first datum in a data set. The Series originis expressed in units defined by the Series units field in the TransducerChannel TEDS (see 8.5.2.31). If the Series units are seconds, the content of this field is the number of seconds to delay before beginning the acquisition or application of the data set.

For example: If the Series units represent time, this value is the delay between the receipt of the trigger and the acquisition or application of the first sample.

8.5.2.30 Series increment

Field Type: 45

Field Name: StepSize

Data type: single-precision real (Float32, 4 octets)

This field may be omitted if the Maximum data repetitions field (See 8.5.2.28) is zero. If the Maximum data repetitions field is nonzero and this field is omitted, the NCAP shall report a fatal TEDS error.

For the case where the data repetition count is greater than zero, the Series increment represents the minimum spacing between values of the independent variable associated with successive members of the

Copyright © 2007 IEEE. All rights reserved.

data-set. The Series increment is expressed in units defined by the Series units field in the TransducerChannel TEDS (8.5.2.31).

For TransducerChannels that implement the capability to program the *Series increment*, this field shall give the minimum interval that the TransducerChannel supports.

NOTE—For TransducerChannels that implement the ability to program the spacing between samples, it is recommended that an embedded actuator be used for this function. This allows the spacing to be specified in Physical Units and a Calibration TEDS to specify how to convert this into the bit pattern required to program the hardware.

8.5.2.31 Series units

Field Type: 46

Field Name: SUnits

Data type: physical units (UNITS, 10 octets)

This field may be omitted if the Maximum data repetitions field (see 8.5.2.28) is zero. If the Maximum data repetitions field is nonzero and this field is omitted, the NCAP shall report a fatal TEDS error.

This field contains the Physical Units associated with the Series origin (8.5.2.29) and Series increment (8.5.2.30) fields in the TransducerChannel TEDS.

8.5.2.32 Maximum pre-trigger samples

Field Type: 47

Field Name: PreTrigg

Data type: unsigned octet integer (UInt16, 2 octets)

This field may be omitted if the Maximum data repetitions field (see 8.5.2.28) is zero or if the TransducerChannel does not support the free-running with pre-trigger sampling modes (see 5.10.1.3). If this field is omitted and the Sampling mode attribute indicates that this TransducerChannel may be operated in the free-running with pre-trigger sampling modes, then a nonfatal TEDS error shall be reported and zero maximum pre-trigger samples shall be assumed.

This field contains the number (L) of repetitions of the TransducerChannel value that may be sampled and stored prior to a trigger when operating in the free-running with pre-trigger modes. Each repetition represents an additional measurement value produced by the TransducerChannel, which shall be spaced apart from the initial value along some axis (for example, time) by an amount defined by the TransducerChannel TEDS fields Series increment (8.5.2.30) and Series units (8.5.2.31), respectively. When L is zero, the sensor may not be operated in the free-running pre-trigger mode.

In operation, the number of TransducerChannel values that may be acquired and stored prior to a trigger may be set to any value lower than the contents of this field by issuing the Set TransducerChannel pre-trigger count command as defined in 7.1.2.2 if that command is implemented.

When reading or writing data with data repetition count greater than zero, the order of transmittal shall be with the zeroth (oldest) data sample transmitted first, the first repetition (next oldest) transmitted second, and so on.

Copyright © 2007 IEEE. All rights reserved.

8.5.2.33 TransducerChannel update time

Field Type: 20

Field Name: UpdateT

Data type: single-precision real (Float32, 4 octets)

This field is required for all TransducerChannel TEDS. For TransducerChannels being operated in the free-running without pre-trigger sampling mode (see 5.10.1.2) or the free-running with pre-trigger sampling mode (see 5.10.1.3), this number should be equal to the sample interval. If the sample interval is programmable and the TransducerChannel is being operated in either of the free-running sampling modes, this value is not relevant and should be set to zero. If this field is omitted, a fatal TEDS error shall be reported.

This field contains the maximum time (t_u), expressed in seconds, between the trigger event and the latching of the first sample in a data set for this TransducerChannel. This time interval shall be calculated assuming that the data repetition count is set to the value specified in the Maximum data repetitions field (see 8.5.2.28). This parameter allows NCAPs to determine time-out values if appropriate.

For sensors in the free-running sampling mode, this parameter only applies when they are in the transducer operating state.

8.5.2.34 TransducerChannel write setup time

Field Type: 21

Field Name: WSetupT

Data type: single-precision real (Float32, 4 octets)

This field is required for all actuator TransducerChannels. If this field is omitted for an actuator, a fatal TEDS error shall be reported. It may be omitted for sensors or event sensors.

This field contains the minimum time (t_{ws}), expressed in seconds, between the end of a write data frame and the application of a trigger. (For most TransducerChannels, this is a setup time characteristic of the electronics. For more complex TransducerChannels, particularly those with a microprocessor, there may be additional time needed that is specified by this constant.)

8.5.2.35 TransducerChannel read setup time

Field Type: 22

Field Name: RSetupT

Data type: single-precision real (Float32, 4 octets)

This field is required for all sensor TransducerChannels. If this field is omitted for a sensor, a fatal TEDS error shall be reported. It may be omitted for actuators or event sensors.

This field contains the maximum time (t_{rs}), expressed in seconds, between the receipt of the trigger by the transducer module and the time that the data is available to be read. (For most devices, this is a setup time

Copyright © 2007 IEEE. All rights reserved.

characteristic of the TransducerChannel electronics. For more complex TransducerChannels, particularly those with a microprocessor, there may be additional time needed that is specified by this constant.)

If the value of TransducerChannel read setup time is zero, the TransducerChannel may be read at any time after it is triggered.

8.5.2.36 TransducerChannel sampling period

Field Type: 23

Field Name: SPeriod

Data type: single-precision real (Float32, 4 octets)

This field is required for all TransducerChannels. If this field is omitted, a fatal TEDS error shall be reported. For many embedded TransducerChannels, the sampling period is not relevant and may be set to zero to indicate that it is not relevant.

The TransducerChannel sampling period (t_{sp}) shall be the minimum sampling period expressed in seconds of the TransducerChannel unencumbered by read or write considerations (since there is no requirement to read or write with each trigger).

For sensor and actuator TransducerChannels, this time will typically be limited by A/D or D/A conversion times, transducer module processor speed, and so on, but in more complex TransducerChannels, it may reflect TransducerChannel or sample handling times as well, for example, a pH sensor that on each trigger extracts a new sample using a pump. If reads or writes are involved, the actual sampling rates are further limited by setup and data transfer times depending on the TransducerChannel type.

In the case of TransducerChannels with the free-running attribute set, this parameter shall represent the sampling time determined by the transducer module implementation.

NOTE—For TransducerChannels that implement the ability to program the spacing between samples, it is recommended that an embedded actuator be used for this function. This allows the spacing to be specified in Physical Units and a Calibration TEDS to specify how to convert this into the bit pattern required to program the hardware.

In the case of event sensors, this parameter shall represent the minimum event resolution time.

The TransducerChannel sampling period shall be expressed in seconds.

8.5.2.37 TransducerChannel warm-up time

Field Type: 24

Field Name: WarmUpT

Data type: single-precision real (Float32, 4 octets)

This field is required. If this field is omitted, the NCAP shall report a nonfatal TEDS error.

This field contains the period of time, expressed in seconds, in which the TransducerChannel stabilizes its performance to predefined tolerances, as specified in Uncertainty under worst-case conditions (see 8.5.2.7), after the application of power to the TransducerChannel.

Copyright © 2007 IEEE. All rights reserved.

8.5.2.38 TransducerChannel read delay time

Field Type: 25

Field Name: RDelayT

Data type: single-precision real (Float32, 4 octets)

This field is required for all sensor TransducerChannels. If this field is omitted for a sensor, a fatal TEDS error shall be reported. It may be omitted for actuators or event sensors.

This field contains the maximum delay time (t_{ch}), expressed in seconds, between the receipt of a Read TransducerChannel data-set segment command (see 7.1.3.1) and the beginning of the transmission of the data frame.

8.5.2.39 TransducerChannel self-test time requirement

Field Type: 26

Field Name: TestTime

Data type: single-precision real (Float32, 4 octets)

This field is required for all TransducerChannels that implement a self-test capability as indicated by the Self-test key (see 8.5.2.21). If the self-test key indicates that no self-test is implemented, this field may be omitted. If the self-test key indicates that a self-test capability exists and this field is omitted, the NCAP shall report a nonfatal TEDS error. The NCAP may assume a self-test time requirement or not access the self-test capability.

This field contains the maximum time, expressed in seconds, required to execute the self-test.

8.5.2.40 Source for the time of sample

Field Type: 27

Field Name: TimeSrc

Data type: unsigned octet integer (UInt8, 1 octet)

This field is optional and may be omitted. If it is omitted, the NoHelp option shall be assumed.

The possible values for this field are shown in Table 53.

If “incoming” is indicated, then the TransducerChannel TEDS field incoming propagation delay through the data transport logic (see 8.5.2.41) is required. If “outgoing” is indicated, then the TransducerChannel TEDS field outgoing propagation delay through the data transport logic (see 8.5.2.42) is required.

The operation with MInterval and SInterval require an embedded sensor to provide the same information as the TEDS field does for Incoming. For ToDSense, the transducer module provides the time of day that the sample was taken. Operation with MInterval, OInterval, or ToDSense requires a ControlGroup to be defined in the Meta-TEDS to identify the time interval sensor or TimeInstance sensor to be used with this TransducerChannel.

Copyright © 2007 IEEE. All rights reserved.

Table 53—Enumeration of sample time sources

Value	Name	Function
0	NoHelp	No facilities in the transducer module are available to support determination of the time that a sample was latched.
1	Incoming	The time interval between the receipt of the trigger and the latching of the sample is not measured. The default delay between the trigger and the latching of the sample is found in the incoming propagation delay through the data transport logic field of the TransducerChannel TEDS (see 8.5.2.41).
2	Outgoing	The time interval between the latching of the sample and when the data are transmitted is not measured. The default delay between the latching of the sample and the transmission of the sample is found in the outgoing propagation delay through the data transport logic field of the TransducerChannel TEDS (see 8.5.2.42). This is only useful when operation occurs in the immediate operation sampling mode (see 5.10.1.7).
3	MInterval	The time interval between the receipt of a trigger and the latching of the sample is measured using a time interval sensor. The Calibration TEDS for the time interval sensor provides the method of converting the output of the time interval sensor to time.
4	SInterval	The time interval between the receipt of the trigger and the latching of the sample is measured using a time interval sensor with a clock derived from a synchronization signal. The Calibration TEDS for the time interval sensor provides the method of converting the output of the embedded sensor to time. NOTE—This capability does not exist in all members of the IEEE 1451 family.
5	ToDSense	A TimeInstance Sensor supplies the time of day that the sample was processed. The characteristics of this sensor are defined in the TEDS for this TimeInstance sensor.
6–127		Reserved.
128–255		Available for use by manufacturers. NOTE—The use of these enumerations may compromise the interoperability of the TransducerChannel. Special software is required in the system to use it.

8.5.2.41 Incoming propagation delay through the data transport logic

Field Type: 28

Field Name: InPropDl

Data type: single-precision real (Float32, 4 octets)

This field is required if the Source for the time of sample field (see 8.5.2.40) contains the value “Incoming”; otherwise it may be omitted. If the Source for the time of sample field contains the value Incoming and this field is omitted, then a nonfatal TEDS error shall be reported and the Source for the time of sample field shall be assumed to be NoHelp.

This field contains a single-precision real number defining the time delay, expressed in seconds, between the receipt of a trigger by the physical layer of the communications protocol stack and the acquisition or application of a sample. This number may be used to determine the time a sample was processed by the TIM based on the time that the trigger was issued.

8.5.2.42 Outgoing propagation delay through the data transport logic

Field Type: 29

Field Name: OutPropD

Data type: single-precision real (Float32, 4 octets)

Copyright © 2007 IEEE. All rights reserved.

This field is required if the Source for the time of sample field (see 8.5.2.40) contains the value "outgoing." It may be omitted otherwise. If the Source for the time of sample field contains the value Outgoing and this field is omitted, then a nonfatal TEDS error shall be reported and the Source for the time of sample field shall be assumed to be NoHelp.

This field contains a single-precision real number defining the time delay, expressed in seconds, between the last sample latched signal (see 5.9.2) and a signal from the data transport logic that the data message has been transmitted. This number may be used to determine the time a sample was processed by the transducer module based on the time that the data message was received. This is only useful when in the immediate operation sampling mode (see 5.10.1.7).

8.5.2.43 Trigger-to-sample delay uncertainty

Field Type: 30

Field Name: TSError

Data type: single-precision real (Float32, 4 octets)

This field is optional for all TransducerChannels. If it is omitted, the NCAP shall assume a value of zero to indicate that the information is not supplied.

This field contains a single-precision real number defining the uncertainty, expressed in seconds, of the default delay between the trigger and the sample being taken or applied.

The expression of the uncertainty shall be in accordance with the "Combined Standard Uncertainty" defined in NIST Technical Note 1297 [B10].

8.5.2.44 Sampling attribute

Field Type: 31

Field Name: Sampling

The sampling attributes field is required for all TransducerChannels. If it is omitted, the NCAP shall report a fatal TEDS error.

The subfields that make up this type are as follows:

Sampling mode capability

Default sampling mode

8.5.2.45 Sampling mode capability attribute

Field Type: 48

Field Name: SampMode

Data type: unsigned 8 bit integer (UInt8, 1 octet)

The sampling mode capability attributes field is required for all TransducerChannel. If it is omitted, the NCAP shall report a fatal TEDS error.

Copyright © 2007 IEEE. All rights reserved.

This attribute is used to describe which Sampling modes (see 5.10.1) are supported by this TransducerChannel. As shown in Table 54, each allowable sampling mode is assigned a bit in this field. If the bit assigned to a given sampling mode is set to a one, the TransducerChannel is capable of being operated in that mode. If more than one bit is set, the sampling mode is programmable.

Table 54—Sampling mode capability attribute

Bit	Definition
0 (lsb)	Set to a one if the TransducerChannel may be operated in the Trigger initiated mode (see 5.10.1.1).
1	Set to a one if the TransducerChannel may be operated in the Free-running without pre-trigger mode (see 5.10.1.2).
2	Set to a one if the TransducerChannel may be operated in the Free-running with pre-trigger mode (see 5.10.1.3).
3	Set to a one if the TransducerChannel may be operated in the Continuous Sampling mode (see 5.10.1.6).
4	Set to a one if the TransducerChannel may be operated in the Immediate operation sampling mode (see 5.10.1.7).
5	Reserved.
6	Reserved.
7 (msb)	Reserved.

8.5.2.46 Default sampling mode

Field Type: 49

Field Name: SDefault

Data type: unsigned 8 bit integer (UInt8, 1 octet)

The default sampling mode attribute field is required for all TransducerChannels that are capable of being operated in more than one sampling mode (see 8.5.2.45). If only one sampling mode is allowed, the default sampling mode shall be the only available sampling mode. If multiple sampling modes are allowed and this field is omitted or multiple bits are set, the NCAP shall report a fatal TEDS error.

This attribute is used to describe which Sampling modes (see 5.10.1) is the default for this TransducerChannel. As shown in Table 55, each allowable sampling mode is assigned a bit in this field. If the bit assigned to a given sampling mode is set to a one, then that bit identifies the default sampling mode for this TransducerChannel. Only a single bit may be set in this field.

Table 55—Default sampling mode attribute

Bit	Definition
0 (lsb)	Set to a one if the Trigger initiated mode (see 5.10.1.1) is the default sampling mode for this TransducerChannel.
1	Set to a one if the Free-running without pre-trigger mode (see 5.10.1.2) is the default sampling mode for this TransducerChannel.
2	Set to a one if the Free-running with pre-trigger mode (see 5.10.1.3) is the default sampling mode for this TransducerChannel.
3	Set to a one if the Continuous Sampling mode (see 5.10.1.6) is the default sampling mode for this TransducerChannel.
4	Set to a one if the Immediate operation sampling mode (see 5.10.1.7) is the default sampling mode for this TransducerChannel.
5	Reserved.
6	Reserved.
7 (msb)	Reserved.

Copyright © 2007 IEEE. All rights reserved.

8.5.2.47 Buffered attribute

Field Type: 32

Field Name: Buffered

Data type: unsigned 8 bit integer (UInt8, 1 octet)

This field is optional. If it is omitted, the NCAP shall assume that the TransducerChannel has no more than one buffer available.

This attribute describes the Buffered operation modes (see 5.10.3) available for this TransducerChannel. Table 56 lists the allowable values for this attribute.

Table 56—Buffered attribute

Value	Description
0	TransducerChannel has no more than one buffer available.
1	TransducerChannel has multiple buffers available and may only be operated in the buffered mode.
2	TransducerChannel has multiple buffers available and may be operated either buffered or unbuffered. The unbuffered mode of operation is the default.
3	TransducerChannel has multiple buffers available and may be operated either buffered or unbuffered. The buffered mode of operation is the default.
4–255	Reserved.

8.5.2.48 End-of-data-set operation attribute

Field Type: 33

Field Name: EndOfSet

Data type: unsigned 8 bit integer (UInt8, 1 octet)

This field is required for actuator TransducerChannels. It may be omitted for sensors and event sensors. If it is omitted for an actuator with Maximum data repetitions field (see 8.5.2.28) greater than zero, a fatal TEDS error shall be reported. If it is omitted for an actuator with Maximum data repetitions equal to zero, a nonfatal TEDS error shall be reported and the Hold mode shall be assumed.

This attribute describes the End-of-data-set operation modes (see 5.10.4) that this actuator can support. Table 57 lists the allowable values for this attribute.

Table 57—End-of-data-set operation attribute

Value	Description
0	Not applicable.
1	This TransducerChannel holds the last value in the data set until another trigger is received as described in 5.10.4.1.
2	This TransducerChannel recirculates through the last data set until another trigger is received as described in 5.10.4.2.
3	This TransducerChannel may be operated with either the hold mode or the recirculate mode. These two modes are mutually exclusive. The hold mode is the default.
4	This TransducerChannel may be operated with either the hold mode or the recirculate mode. These two modes are mutually exclusive. The recirculate mode is the default.
5–255	Reserved.

Copyright © 2007 IEEE. All rights reserved.

8.5.2.49 Data transmission attribute

Field Type: 34

Field Name: DataXmit

Data type: unsigned 8 bit integer (UInt8, 1 octet)

This field is an optional attribute for sensor or event sensor TransducerChannels. It may be omitted for actuators. If it is omitted for a sensor or event sensor, the NCAP shall assume that this TransducerChannel is only capable of being operated in the only when commanded mode (see 5.10.2.1).

This attribute describes data transmission modes that this TransducerChannel supports (see 5.10.2). Table 58 lists the allowable values for this attribute.

Table 58—Data transmission attribute

Value	Description
0	Reserved.
1	This TransducerChannel is only capable of being operated in the only when commanded modes (see 5.10.2.1).
2	This TransducerChannel is capable of being operated in the Streaming when a buffer is full (see 5.10.2.2) or only when commanded modes.
3	This TransducerChannel is capable of being operated in the Streaming at a fixed interval (see 5.10.2.3) or only when commanded modes.
4	This TransducerChannel is capable of being operated in the only when commanded, Streaming when a buffer is full or Streaming at a fixed interval modes.
5–255	Reserved.

8.5.2.50 Edge-to-report attribute

Field Type: 35

Field Name: EdgeRpt

Data type: unsigned 8 bit integer (UInt8, 1 octet)

This field is a required attribute for an event sensor TransducerChannel. It may be omitted for sensors or actuators. If it is omitted or a reserved value is indicated for an event sensor, the NCAP shall report a fatal TEDS error.

This attribute describes the Edge-to-report operating modes (see 5.10.6) supported by this event sensor. Table 59 lists the allowable values for this attribute.

Copyright © 2007 IEEE. All rights reserved.

Table 59—Edge-to-report options

Value	Meaning
0	Not applicable.
1	This event sensor only reports rising edges.
2	This event sensor only reports falling edges.
3	This event sensor reports on both edges.
4	Reserved.
5	This event sensor is capable of reporting both edges, but the default is to report rising edges.
6	This event sensor is capable of reporting both edges, but the default is to report falling edges.
7	This event sensor is capable of reporting both edges, and the default is to report both edges.
8–255	Reserved.

8.5.2.51 Actuator-halt attribute

Field Type: 36

Field Name: ActHalt

Data type: unsigned 8 bit integer (UInt8, 1 octet)

This field is a required attribute for an actuator TransducerChannel. It may be omitted for sensors or event sensors. If it is omitted or a reserved value is indicated for an actuator, the NCAP shall report a fatal TEDS error.

This attribute describes the Actuator-halt modes (5.10.7) supported by this actuator. This mode determines what the actuator does when it receives a TransducerChannel Idle command (see 7.1.4.2). Table 60 lists the allowable values for this attribute.

Table 60—Actuator-halt operations

Value	Meaning
0	Not applicable
1	Halt Immediate
2	Halt at the end of the data set
3	Ramp to a predefined state
4–255	Reserved

8.5.2.52 Sensitivity direction

Field Type: 37

Field Name: Directon

Data type: unsigned 8 bit integer (UInt8, 1 octet)

This field is optional.

For sensors that measure physical phenomena with three-dimensional spatial properties such as acceleration, velocity, or displacement, this shall identify direction relative to a rectangular coordinate system defined by the sensor manufacturer. The sensor shall produce a positive output when the physical phenomenon is applied to the sensor in the specified direction.

Copyright © 2007 IEEE. All rights reserved.

Direction is determined using the "right-hand rule" with a "right-handed" coordinate system: The fingers of the right-hand point in the +X direction, the palm points in the +Y direction and the thumb points in the +Z direction.

For single-axis sensors, the axis to be used shall be the Z-axis.

The sensor manufacturer shall identify at least two of the three axes by markings on the sensor enclosure.

The direction of the physical phenomena measured by the TransducerChannel shall be defined by the enumeration in Table 61.

Table 61—Sensitivity direction enumeration

Value	Meaning
0	Not applicable
1	+X
2	–X
3	+Y
4	–Y
5	+Z
6	–Z
7–255	Reserved for future expansion

8.5.2.53 Direction angles

Field Type: 38

Field Name: DAngles

Data type: Two single-precision real (Float32, 4 octets) words.

This field is optional.

The two single-precision real words define two angles measured from the reference plane and reference direction marked on the TransducerChannel by the manufacturer. The first number represents ρ in right-hand cylindrical spatial coordinates expressed in radians. The second number represents ϕ in right-hand cylindrical spatial coordinates expressed in radians.

8.5.2.54 Event sensor options

Field Type: 39

Field Name: ESOption

Default value: 0

Data type: unsigned octet integer (UInt8, 1 octet)

This field only applies to event sensor TransducerChannels. It may be omitted for sensors and actuators. If it is omitted for an event sensor, the NCAP shall report a fatal TEDS error.

Copyright © 2007 IEEE. All rights reserved.

An event sensor has the option of changeable pattern, upper threshold, and/or hysteresis. It also has the option of detecting inconsistencies in settings of these parameters as described in 5.6.2.3. This enumeration defines the ability of the transducer module to detect and report these inconsistencies. The options are listed in Table 62.

Table 62—Event sensor options

Value	Meaning
0	Not applicable
1	Pattern/threshold/hysteresis not changeable
2	Changeable and inconsistencies detected
3	Changeable and inconsistencies not detected
4–255	Reserved

8.5.2.55 TEDS access code for units extension

Field Type: 60

Field Name: UnitsExt

Data type: unsigned octet integer (UInt8, 1 octet)

This field is optional.

This field is provided to allow a text-based extension to the Physical Units. It shall not be used as a subsutitute for the Physical Units. Since there can be more than one set of Physical Units in a TransducerChannel TEDS, the contents of this field shall provide the TEDS access code for the text-based TEDS that provides this extension.

8.6 Calibration TEDS

This field is an optional TEDS. The function of the Calibration TEDS is to make available all of the information used by correction software in connection with the TransducerChannel being addressed.

8.6.1 Correction process

Correction is the application of a specified mathematical function upon TransducerChannel data from one or more TransducerChannels and/or data delivered from other software objects. This subclause gives an overview of how the correction process is modeled, in order to aid understanding of how to develop and use the entries in the Calibration TEDSfor correction.

Correction is intended to reconcile two different numbers associated with a TransducerChannel:

The NCAP-side number: This number represents the TransducerChannel's value expressed in the Physical Units field of the TransducerChannel TEDS (8.5.2.6) as modified by the SI units conversion constants (8.6.3.4) in the Calibration TEDS. This number is used to represent the TransducerChannel data in the user's system.

The transducer-side number: This number is read from or written to the TransducerChannel hardware (usually an A/D or D/A Converter).

Copyright © 2007 IEEE. All rights reserved.

The goal of correction depends on the TransducerChannel type of the addressed TransducerChannel. The application of correction, however, is the same regardless of TransducerChannel type:

For sensors, correction takes as input the transducer-side data from the addressed TransducerChannel and possibly data from other TransducerChannels. It produces as output the NCAP-side number.

For actuators, correction takes as input the NCAP-side number for the addressed TransducerChannel, which is the intended next state of the actuator, and possibly data from other TransducerChannels. The output is the transducer-side number.

Either the NCAP-side or the transducer-side value may be used as an input to the correction function of another TransducerChannel, as selected by the Correction input TransducerChannel number field (8.6.3.15).

8.6.1.1 Method

For a sensor, "Correction" is the process by which the constants determined by the calibration process are applied to the output of a sensor to convert the existing number to the form desired by the system. For an actuator, "Correction" is applied to the number provided by the system to convert it to the form required by the actuator.

The calibration process is the process of defining an equation over a segment of the range of the transducer that describes the transfer function of the transducer over that segment. One or more segments are defined for any given calibration. By far the most common number of segments is one, which means that most transducers can be described by a single equation. It is also true that the most commonly used equation is as shown in Equation (5).

$$y = m\,x + b = b + m\,x \tag{5}$$

Two methods are defined in the Calibration TEDS. The first method is based on the fact that most calibrations use a single segment linear conversion, so this special case is defined separately. The second method is the general method that will work with almost any correction function. This general method can be used with multiple segments and equations of degree other than one. It may also use multiple inputs that may be used for such functions as compensating the output of one transducer for variations caused by something other than the quantity being measured.

8.6.1.2 Linear method

This method is the subset of the general method where the correction function takes the form shown in Equation (5) and where only a single segment is required.

8.6.1.3 General method

The method described in this standard is an attempt to develop a single method that can be used for any correction function including the linear method. All commonly used correction functions consist of two different steps. The first step is segmentation. In this step the calibration curve is divided into segments in order to achieve the desired accuracy with a reasonable degree for the polynomial equation. Although the most commonly used segmentation is a single segment, making the decision to use a single segment is still segmentation. The extreme case is a table lookup where a segment is defined for each possible bit combination within the desired range. The second step in correction functions is to define a polynomial curve to fit the response of the transducer within each segment. Again it is recognized that a linear

Copyright © 2007 IEEE. All rights reserved.

equation, i.e., a polynomial of degree one, is the most commonly used polynomial. The case of a table lookup uses a polynomial of degree zero.

A second point to be made is that most correction functions only involve a single input variable. However, if a general solution for this function is to be defined, it is necessary to consider the case where multiple inputs are required for a single output. Common uses for a function with multiple inputs is to temperature compensate the output of a sensor or signal conditioner or to correct for cross-axis sensitivity in strain gage installations.

When put into the correct mathematical form the equation is expressed as shown in Equation (6) and is defined as a multinomial (multivariate polynomial). The "*H*" terms in the equation are offsets that are subtracted from the input to minimize the number that is being raised to a power to minimize the chance of multiplication resulting in a numerical overflow. This value of the offset may fall outside the segment for which it is defined. For those people not familiar with this notation, the equation for the polynomial in a given segment can be expressed as shown in Equation (7), where "*x*" replaces the term "*X* – *H*" in Equation (6).

$$\sum_{i=0}^{D_1}\sum_{j=0}^{D_2}\Lambda\sum_{p=0}^{D_n} C_{i,j,\Lambda,p}[X_1 - H_1]^i[X_2 - H_2]^j \Lambda\,[X_n - H_n]^p \tag{6}$$

D_k is the degree of the input X_k. That is, it is the highest power to which $[X_k - H_k]$ is raised in any term of the multinomial. Note that the degree of each input may be different.

$$y = (A_0 + A_1x_1 + A_2{x_1}^2 \ldots + A_i{x_1}^i)(B_0 + B_1x_2 + B_2{x_2}^2 \ldots + B_j{x_2}^j)(..)(N_0 + N_1x_n + N_2{x_n}^2 \ldots + N_p{x_n}^p) \tag{7}$$

If we consider the case where we have two inputs, the polynomials can be multiplied out to produce the equation shown in Equation (8). Noting that the product of constants results in a different constant, we may write the equation as shown in Equation (9):

$$y = A_0B_0 + A_1B_0x_1 + A_2B_0{x_1}^2 + A_3B_0{x_1}^3 + ..A_iB_0{x_1}^i + A_0B_1x_2 + A_1B_1x_1x_2 + A_2B_1{x_1}^2x_2 + ..A_iB_j{x_1}^i{x_2}^j \tag{8}$$

$$C_0 + C_1x_x + C_2{x_1}^2 + C_3{x_1}^3 + ..C_n{x_1}^i + C_{n+1}x_2 + C_{n+2}x_1x_2 + C_{n+3}{x_1}^2x_2 + ..C_m{x_1}^i{x_2}^j \tag{9}$$

If we added more inputs, it would not change the form of the equation and it would just create more terms.

Equation (7) through Equation (9) are examples of this function in two of its simplest and very familiar forms. represents the form of this equation that is used for a table lookup correction. Equation (10) shows the form that the equation would take if the correction function were a linear function. The second form of Equation (11) is a more common form of writing this equation.

$$y = C_0 \tag{10}$$

$$y = (C_0 + C_1x) = b + mx \tag{11}$$

Segmentation of one or more TransducerChannels divides the input domain into cells with orthogonal boundaries. For example, for a two-dimensional multinomial, the cells are rectangles. Each cell has its own set of coefficients $C_{i,j,\ldots p}$.

Copyright © 2007 IEEE. All rights reserved.

For the two-input correction process shown in Figure 15, if each input has degree 1, then the multinomial for each cell is as follows:

$$C_{0,0} + C_{0,1} \cdot (X_2 - H_2) + C_{1,0} \cdot (X_1 - H_1) + C_{1,1} \cdot (X_1 - H_1) \cdot (X_2 - H_2) \quad (12)$$

As observed in Equation (12), there is a different H_1 for each partition of X_1. That is, H_1 is the same in segments 1, 2, and 3 but different in segments 1 and 4. Likewise, H_2 changes from segment 1 to 2 or from segment 4 to 5. Coefficients $C_{0,0}$ to $C_{1,1}$ are different in each segment. The correction software must determine from the values of X_1 and X_2 into which segment the measurement falls and must choose the coefficients and offsets accordingly.

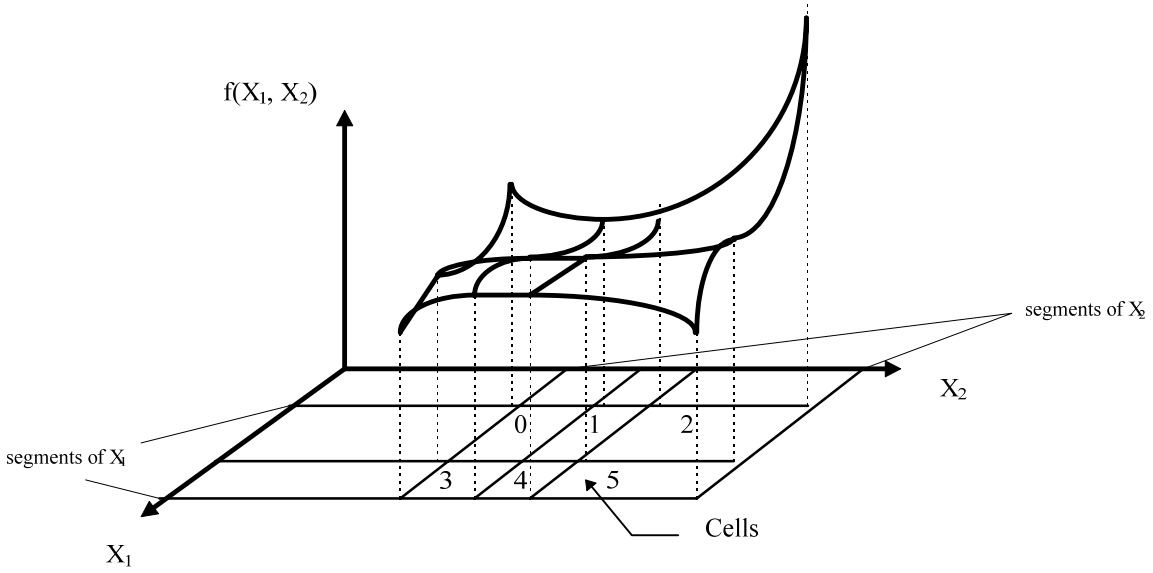


Figure 15—A two-dimensional function partitioned into 2 by 3 cells

8.6.1.4 Application

If the value of the Calibration key field (see 8.5.2.2) in the TransducerChannel TEDS is CAL_SUPPLIED, the correction algorithm shall be performed in the NCAP, a host processor, or elsewhere in the system. It is recommended that it be performed in the NCAP.

If the value of the Calibration key field is TIM_CAL_SUPPLIED or TIM_CAL_SELF, then the correction algorithm shall be performed in the TIM.

It is expected, but not required, that correction software will use a floating-point numeric format for its computations. Conversion to and from the numeric format used by the correction software and all possible TransducerChannel data models is therefore required. (If the correction is done in the TIM, conversion to and from only the data models used in the TIM is required.) Conversion of Calibration TEDSentries may also be necessary in order for the correction software to use them. The method of conversion is beyond the scope of this standard.

The application of the correction process on the NCAP, a host processor, or elsewhere in the system shall be governed by the following rules:

Correction shall be invoked on a sensor after new transducer-side data are read from the TIM.

Copyright © 2007 IEEE. All rights reserved.

Correction shall be invoked on an actuator after new NCAP-side data are provided, and before the corrected transducer-side data are written to the TIM.

The correction engine shall use the values currently available for any other TransducerChannel data required.

The application of the correction process shall not initiate triggering or reading on any TransducerChannel.

The application of the correction process shall not initiate writing on any non-addressed TransducerChannel.

The transducer-side number has a data type specified in the Data model field (8.5.2.24), the Data model length field (8.5.2.25), and the Model significant bits field (8.5.2.26) in the TransducerChannel TEDS.

The application of the correction process on the TIM shall be governed by the following rules:

Correction may be invoked on a sensor when a new sample is acquired or after the TransducerChannel is triggered. If the correction is invoked by the trigger, the TransducerChannel read setup time (8.5.2.35) shall include the time necessary for correction.

Correction shall be invoked on an actuator TransducerChannel after new data are written to the TransducerChannel, before the TransducerChannel is triggered. The TransducerChannel write setup time (8.5.2.34) shall include the time necessary for correction.

The correction engine may use the values currently available in the TIM for any other TransducerChannel data required. Optionally, each input may be processed as it is acquired.

Correction shall not have the effect of triggering any TransducerChannel involved with the correction.

The NCAP-side number has a data type specified in the Data model field (8.5.2.24), the Data model length field (8.5.2.25), and the Model significant bits field (8.5.2.26) in the TransducerChannel TEDS.

Irrespective of where the correction process is applied, it shall be governed by the following rules:

The application of the correction process to one TransducerChannel shall not change the NCAP-side data or the transducer-side data of another TransducerChannel even if the other TransducerChannel is an input to the correction process of the first.

If the data repetition count is greater than zero for the addressed TransducerChannel (that is, vector data), the data repetition count of any other TransducerChannel used in the correction shall be either zero (scalar) or equal to that of the addressed TransducerChannel. The correction shall be applied using vector elements in sequence from each vector input to produce a vector output. Scalar data are used unchanged for the correction of each vector element.

Correction may use or produce data with Physical Units type digital data if it makes sense to do so (for instance if the data are simply “counts”).

The order and availability of the inputs to the correction process must be taken into consideration. The process takes the data that are available when it is required and uses it. If one parameter has been updated and another has not been updated, this can cause variation in the results. It is especially important when using corrected data. The inputs to the process that are corrected values from other TransducerChannels shall be converted before the processing is started on the TransducerChannel of interest.

Copyright © 2007 IEEE. All rights reserved.

8.6.1.5 Conversion between SI units and units output by correction process

The constants provided in the SI units conversion constants (see 8.6.3.4) may be used to convert the numbers in the TEDS fields between the different sets of units.

8.6.1.6 Conversion from units output by correction process to SI units

The conversion to SI units from the units output by the correction process is accomplished by the use of Equation (13).

$$SI = Slope(Value) + Intercept \tag{13}$$

where

Slope	is given in the SI units conversion slope field (see 8.6.3.5) in this TEDS
Intercept	is given in the SI units conversion intercept field (see 8.6.3.6) in this TEDS
Value	is a number in the units output by the corrections process
SI	is a number expressed in SI units

8.6.1.7 Conversion from SI units to units output by correction process

The conversion to the units output by the correction process from SI units is accomplished by the use of Equation (14).

$$Value = \frac{(SI - Intercept)}{Slope} \tag{14}$$

where

Slope	is given in the SI units conversion slope field (see 8.6.3.5) in this TEDS
Intercept	is given in the SI units conversion intercept field (see 8.6.3.6) in this TEDS
Value	is a number in the units output by the corrections process
SI	is a number expressed in SI units

8.6.2 Access

The Calibration TEDS is accessed using a Query TEDS command, a Read TEDS segment command, a write TEDS segment command, or an Update TEDS command. The argument of the command shall specify the TEDS access code of the Calibration TEDS, as defined in Table 17.

This TEDS may be implemented as a read-only TEDS or a read/write TEDS, at the manufacturer's option. However if it is implemented as a read-only TEDS, the TransducerChannel write TEDS segment and TransducerChannel Update TEDS commands shall not apply.

8.6.3 Data block

Table 63 and Figure 16 show the data structure that shall be used for Calibration TEDS. Subsequent subclauses explain each data field in the structure.

Copyright © 2007 IEEE. All rights reserved.

8.6.3.1 TEDS identifier

The TEDS identifier shall comply with the structures defined in 8.3.

This field is required. If this field is omitted or contains illegal values, the NCAP shall report a fatal TEDS error.

8.6.3.2 Last calibration date-time

Field Type: 10

Field Name: LstCalDt

Data type: Time value (TimeInstance, 8 octets)

This field is optional.

This field gives the time and date when the TransducerChannel was last calibrated. It is expressed in the TimeInstance format described in 4.9.2.

8.6.3.3 Calibration interval

Field Type: 11

Field Name: CalInrvl

Data type: Time duration (TimeDuration, 8 octets)

This field is optional.

The calibration interval is the length of time, in seconds, that this TransducerChannel can operate without needing another calibration and produce data that are within the Operational uncertainty specified in 8.6.3.9.

8.6.3.4 SI units conversion constants

Field Type: 12

Field Name: SIConvrt

This field is required. If this field is omitted, the NCAP shall report a nonfatal TEDS error. If the unit is calibrated in SI units, then the SI units conversion slope shall be set to one and the SI units conversion intercept to zero.

This field consists of two subfields:

SI units conversion slope
SI units conversion intercept

NOTE—Not all tables of conversion constants give exactly the same floating-point number for the slope and intercept for this conversion. Some carry more resolution than others, and still others round the numbers differently. When

Copyright © 2007 IEEE. All rights reserved.

comparing the values given in this TEDS with numbers from a table of conversion factors to determine the Physical Units that would be output from the correction process using the constants in this TEDS, this variability must be taken into account.

8.6.3.5 SI units conversion slope

Field Type: 30

Field Name: SISlope

Data type: single-precision real (Float32, 4 octets)

This field is required. If this field is omitted, the NCAP shall report a nonfatal TEDS error.

This field contains the number, that, if multiplied by the output of the correction process and added to the number in the SI units conversion intercept, will result in a number that represents the physical value in SI units.

If the correction process outputs SI units, the value in this field shall be one.

8.6.3.6 SI units conversion intercept

Field Type: 31

Field Name: Intrcpt

Data type: single-precision real (Float32, 4 octets)

This field is required. If this field is omitted, the NCAP shall report a nonfatal TEDS error.

This field contains the number that if added to the SI units conversion slope multiplied by the output of the correction process will result in a number that represents the physical value in SI units.

If the correction process outputs SI units, the value in this field shall be zero.

Copyright © 2007 IEEE. All rights reserved.

Table 63—Structure of the Calibration TEDS data block

Field type	Field name	Description	Type	Required /optional	Data length in octets
—		TEDS length	UInt32		4
0–2	—	Reserved	—	—	—
3	TEDSID	TEDS identifier	UInt8	R	4
4–9	—	Reserved	—	—	—
—		Calibration date related information		—	—
10	LstCalDt	Last calibration date-time	TimeInstance	O	8
11	CalInrvl	Calibration interval	TimeDuration	O	8
Units information					
12	SIConvrt	SI units conversion constants	—	O	—
30	SISlope	SI units conversion slope	Float32	O	4
31	Intrcpt	SI units conversion intercept	Float32	O	4
Operational limits and uncertainty					
13	LowLimit	Operational lower range limit	Float32	O	4
14	HiLimit	Operational upper range limit	Float32	O	4
15	OError	Operational uncertainty	Float32	O	4
Mathematical conversion to be performed on the data before or after correction					
16	OConvert	Post-conversion operation	UInt8	O	1
17	IConvert	Pre-conversion operation	UInt8	O	1
TLV tuple 20 is used when the linear method of conversion is used.					
20	LinOnly	This field is used when only a linear single section conversion is required	—	O	—
TLV tuple 21 provides a description of one TransducerChannel that is involved in the correction specified in this TEDS					
21	XdcrBlk	TransducerChannel description	—	O	—
40	Element	Element number	UInt16	O	1
41	ChanNum	Correction input TransducerChannel	UInt16	O	1
42	ChanKey	Correction input TransducerChannel key	UInt8	O	1
43	Degree	TransducerChannel degree	UInt8	O	1
44	STable	Segment boundary values table	—	—	—
45	OTable	Segment offset values table	Float32Array	O	Note 1
46	LoBndry	Array of low boundary limits	Float32Array	O	Note 1
47	HiBndry	High boundary limit	Float32	O	4
TLV tuple 22 provides a set of coefficients for one segment of the correction specified in this TEDS					
22	CoefBlk	Multinomial coefficient	—	O	—
50	CellNum	Cell number of the segment to which this set of coefficients apply	UInt8	O	1
51	CoefSet	The set of coefficients that applies to this cell	Float32Array	O	Note 2
18–19	—	Reserved	—	—	—
23–29	—	Reserved	—	—	—
48–49	—	Reserved	—	—	—
52–127	—	Reserved	—	—	—
128–255	—	Open to manufacturers	—	—	—
—		Checksum	UInt16	—	2

NOTE 1—Number of segments multiplied by 4.

NOTE 2—((Degree of element 1) + 1) × ((Degree of element 2) + 1) × (…) × ((Degree of element n) + 1) × 4.

Copyright © 2007 IEEE. All rights reserved.

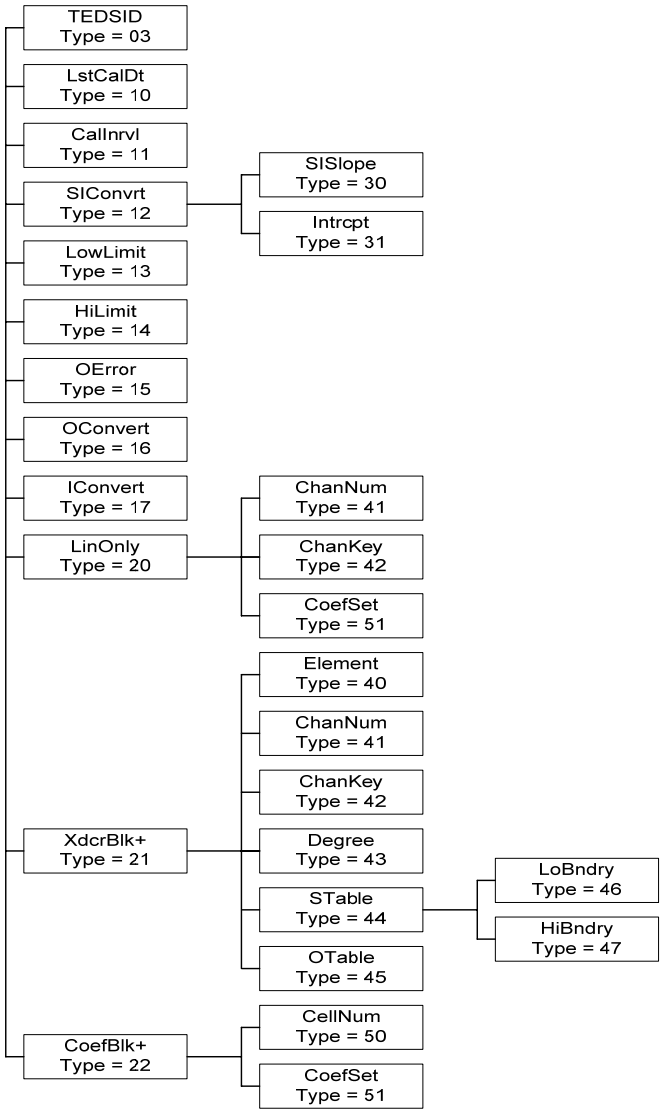


Figure 16—Calibration TEDSStructure

8.6.3.7 Operational lower range limit

Field Type: 13

Field Name: LowLimit

Data type: single-precision real (Float32, 4 octets)

This field is optional. If this field is not provided and the SI units conversion slope (see 8.6.3.5) and SI units conversion intercept (see 8.6.3.6) are one and zero, respectively, the NCAP shall use the Design operational lower range limit specified in the TransducerChannel TEDS (see 8.5.2.7). Otherwise the NCAP shall report a fatal TEDS error.

Copyright © 2007 IEEE. All rights reserved.

For sensors, this shall be the lowest valid value for TransducerChannel data after correction is applied, interpreted in the units appropriate for this Calibration TEDS. If the corrected TransducerChannel data lie below this limit, it may not meet the operational uncertainty specified in 8.6.3.9.

For actuators, this shall be the lowest valid value for TransducerChannel data before correction is applied, interpreted in the units appropriate for this Calibration TEDS. Writing corrected TransducerChannel data below this limit may result in behavior outside the TIM specifications set by the manufacturer.

NOTE—For TransducerChannels that use multiple inputs to produce a single output, this limit, which is expressed in terms of the output, will normally be a nominal value rather than a precise value.

When this parameter is not applicable, it shall be NaN (see 4.5.1).

NOTE—As an example of an application in which range limits do not apply, consider a bank of switches modeled as N-octet data. In this case, both range limit fields shall be set to NaN. This is not to say that range limits do not apply to N-octet data. For example, a 12 bit integer with no expressed units, such as raw ADC output, would also be modeled as N-octet data. In this case, range limits are applicable.

8.6.3.8 Operational upper range limit

Field Type: 14

Field Name: HiLimit

Data type: single-precision real (Float32, 4 octets)

This field is optional. If this field is not provided and the SI units conversion slope (see 8.6.3.5) and SI units conversion intercept (see 8.6.3.6) are one and zero, respectively, the NCAP shall use the Design operational upper range limit specified in the TransducerChannel TEDS (see 8.5.2.19). Otherwise the NCAP shall report a fatal TEDS error.

For sensors, this shall be the highest valid value for TransducerChannel data after correction is applied, interpreted in the units appropriate for this Calibration TEDS. If the corrected TransducerChannel data lie above this limit, it may not meet the operational uncertainty specified in 8.6.3.9.

For actuators, this shall be the highest valid value for TransducerChannel data before correction is applied, interpreted in the units appropriate for this Calibration TEDS. Writing corrected TransducerChannel data above this limit may result in behavior outside the TIM specifications set by the manufacturer.

NOTE—For TransducerChannels that use multiple inputs to produce a single output, this limit, which is expressed in terms of the output, will normally be a nominal value rather than a precise value.

When this parameter is not applicable, it shall be NaN (see 4.5.1).

8.6.3.9 Operational uncertainty

Field Type: 15

Field Name: OError

Data type: single-precision real (Float32, 4 octets)

This field is optional. If this field is not provided, the NCAP shall use the uncertainty under worst-case conditions specified in the TransducerChannel TEDS.

Copyright © 2007 IEEE. All rights reserved.

This field uses the "Combined Standard Uncertainty" defined in NIST Technical Note 1297 [B10]. The value of this field shall be expressed in the units appropriate for this Calibration TEDS.

When this parameter is not applicable, it shall be NaN (see 4.5.1).

8.6.3.10 Post-conversion operation

Field Type: 16

Field Name: OConvert

Data type: unsigned octet integer (UInt8, 1 octets)

This field is optional. If this field is omitted, the NCAP shall assume that no post-conversion operation is required.

The post-conversion operation field contains an enumeration identifying a mathematical operation that shall be performed on the output from the correction process to produce a value in the units specified in the TransducerChannel TEDS. Table 64 lists the allowable values for this field.

8.6.3.11 Pre-conversion operation

Field Type: 17

Field Name: IConvert

Data type: unsigned octet integer (UInt8, 1 octets)

This field is optional. If this field is omitted, the NCAP shall assume that no pre-conversion operation is required.

The pre-conversion operation field contains an enumeration identifying a mathematical operation that shall be performed on the input to the correction process to produce a value in the units specified in the TransducerChannel TEDS. Table 64 lists the allowable values for this field.

Table 64—Pre- or post-correction operation

Value	Meaning
0	No pre- or post-conversion operation is required
1	Inversion: Apply 1/x
2	Log base 10: Apply log10(x)
3	Exponent Base 10: Apply 10x
4	Natural Log : Apply ln (x)
5	Exponent Base e: Apply ex
6–255	Reserved

8.6.3.12 Linear conversion only

Field Type: 20

Field Name: LinOnly

Copyright © 2007 IEEE. All rights reserved.

This field is required if the linear conversion only method is used. This field or fields 21 and 22 shall be provided. If field 20 or 21 and 22 are not provided, the NCAP shall report a fatal TEDS error.

This field describes all of the constants required for a single TransducerChannel when the conversion contains a single section and the conversion is linear. If this field is used, fields 21 and 22 and all associated subfields shall be omitted. It consists of the following subfields:

Correction input TransducerChannel number (8.6.3.15). This field may be omitted if the Correction input TransducerChannel number is the same as the TransducerChannel number to which this Calibration TEDS applies.

Correction input TransducerChannel key (8.6.3.16). If this field is omitted, it shall be assumed that the data will be taken from the transducer side of the correction process for a sensor (see Figure 17). For an actuator it shall be assumed that the input to the correction comes from the NCAP side of the correction process.

Set of coefficients (8.6.3.24). The set of coefficients shall be limited to a slope and intercept.

If field type 21 and 22 are included, then field 20 shall be omitted.

8.6.3.13 TransducerChannel description

Field Type: 21

Field Name: XdcrBlk

This field is required if the general conversion method is used. This field and field 22 or field 20 shall be provided. If field 20 or 21 and 22 are not provided, the NCAP shall report a fatal TEDS error.

If field type 20 is included, fields 21 and 22 shall be omitted.

This field describes all of the constants except the calibration coefficients required for a single TransducerChannel that is part of the correction process. It consists of the following subfields:

Element number
Correction input TransducerChannel
Correction input TransducerChannel key
TransducerChannel degree
Segment boundary values table
Segment offset values table

If multiple TransducerChannels provide inputs to the correction process for the TransducerChannel to which this Calibration TEDSapplies, then each of these input TransducerChannels shall have a TransducerChannel Description in this Calibration TEDS.

8.6.3.14 Element number

Field Type: 40

Field Name: Element

Data type: unsigned 16 bit integer (UInt16, 2 octets)

Copyright © 2007 IEEE. All rights reserved.

If field type 21 is included, this field is required. If field 21 is included and this field is omitted, the NCAP shall report a fatal TEDS error.

The element number is used to determine the cell number. The element numbers determine the order in which the cells are numbered as described in 8.6.3.23.

8.6.3.15 Correction input TransducerChannel number

Field Type: 41

Field Name: ChanNum

Data type: unsigned octet integer (UInt16, 2 octets)

If field type 21 is included, this field is required. If field 21 is included and this field is omitted, the NCAP shall report a fatal TEDS error.

This field contains the TransducerChannel number for this input to the correction process.

8.6.3.16 Correction input TransducerChannel key

Field Type: 42

Field Name: ChanKey

Data type: unsigned octet integer (UInt8, 1 octet)

If field type 21 is included, this field is required. If field 21 is included and this field is omitted, the NCAP shall report a fatal TEDS error.

The key for determining the source of the data value associated with this TransducerChannel. The possible values for the key and their meaning are defined in Table 65.

Table 65—Correction input TransducerChannel key

Value	Meaning
0	The correction input value shall be taken from the transducer-side of the correction process
1	The correction input value shall be taken from the NCAP-side of the correction process
2–255	Invalid

For a TransducerChannel, the following choices are available depending on whether both corrected and uncorrected data are available:

If only uncorrected data are available, the value for the key shall be zero.
If only corrected data are available, the value for the key shall be one.
If both forms are available, the user may select either form for the input to the correction process. This choice depends on factors related to the calibration curves and is beyond the scope of this standard.

Figure 17 illustrates the meaning of the keys. The correction process for TransducerChannel 3 uses data from TransducerChannel 1 as one of its inputs. For the TransducerChannel 3 correction process, the TransducerChannel 1 data may come from either side of the TransducerChannel 1 correction process, depending on the key listed in the TransducerChannel Description for TransducerChannel 1 in the Calibration TEDSof TransducerChannel 3.

Copyright © 2007 IEEE. All rights reserved.

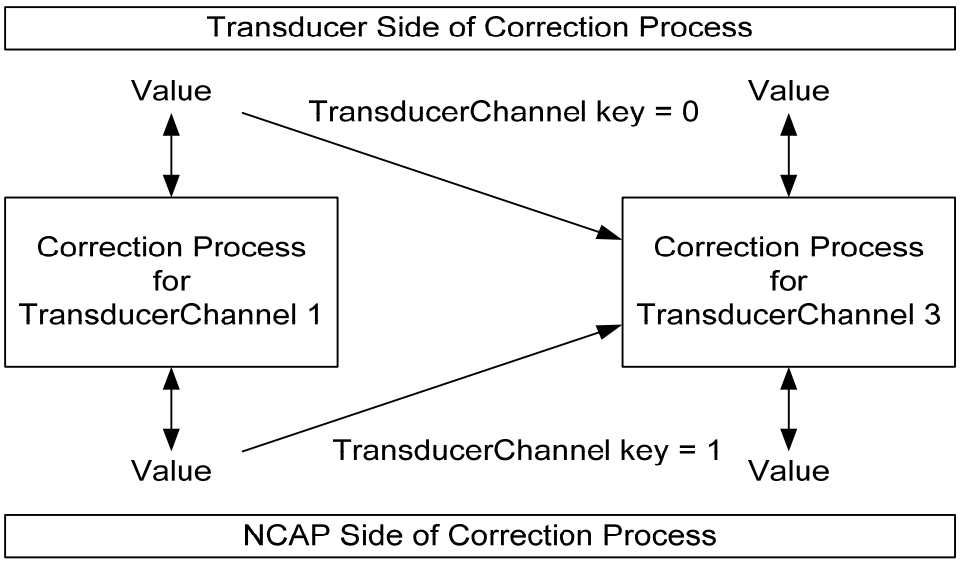


Figure 17—Meaning of correction input TransducerChannel key

8.6.3.17 TransducerChannel degree

Field Type: 43

Field Name: Degree

Data type: unsigned octet integer (UInt8, 1 octet)

If field type 21 is included, this field is required. If field 21 is included and this field is omitted, the NCAP shall report a fatal TEDS error.

The TransducerChannel degree shall be the degree of the corresponding correction input. The degree is the highest power that the input for which this TransducerChannel Description applies is raised in any term of the multinomial.

8.6.3.18 Segment boundary values table

Field Type: 44

Field Name: STable

If field type 21 is included, this field is required. If field 21 is included and this field is omitted, the NCAP shall report a fatal TEDS error.

This field consists of two subgroups:

Array of low boundary limits
High boundary limit

Copyright © 2007 IEEE. All rights reserved.

8.6.3.19 Array of low boundary limits

Field Type: 46

Field Name: LoBndry

Data type: array of single-precision real (Float32Array)

If field type 21 is included, this field is required. If field 21 is included and this field is omitted, the NCAP shall report a fatal TEDS error.

The array of low boundary limits is a one-dimensional array (table) containing the lower boundary for each segment of the input being described by this TransducerChannel Description.

> The elements shall define the domain segment boundaries in ascending numerical order. Note that the domain may be based on either the transducer-side or the NCAP-side data, as indicated in the corresponding Correction input TransducerChannel key (see 8.6.3.16). However, the boundary table values are in a floating-point format, so that if the transducer-side data are used, a conversion process is required.
>
> The first element shall have a value that is less than or equal to the lowest possible value of the input. The last element shall have a value that is equal to the lowest value of the input for the last segment. For the input value (X) to be identified as being within a given segment, it shall satisfy the following equation:
>
> $$B \leq X < B_{(s+1)}$$
>
> where B is the low boundary limit for a segment and $B_{(s+1)}$ is the low boundary limit for the next higher segment.

These values are stored in the TEDS as single-precision real numbers, notwithstanding that they may represent data that are in a different numeric representation.

8.6.3.20 High boundary limit

Field Type: 47

Field Name: HiBndry

Data type: single-precision real (Float32)

If field type 21 is included, this field is required. If field 21 is included and this field is omitted, the NCAP shall report a fatal TEDS error.

The high boundary limits is a single-precision number that is greater than the highest value for the highest segment in the range.

8.6.3.21 Segment offset values table

Field Type: 45

Field Name: OTable

Copyright © 2007 IEEE. All rights reserved.

Type: array of single-precision real (Float32)

If field type 21 is included, this field is required. If field 21 is included and this field is omitted, the NCAP shall report a fatal TEDS error.

The Segment offset values table is a one-dimensional array (table) containing one offset value for each segment of the input being described by this TransducerChannel Description.

Note that O_s (thus, H_k) may be based on either the transducer-side or the NCAP-side data for the input, as indicated in the corresponding Correction input TransducerChannel key (see 8.6.3.16).

These values are stored in the TEDS as single-precision real numbers, notwithstanding that they may represent data that are in a different numeric representation.

8.6.3.22 Multinomial coefficient block

Field Type: 22

Field Name: CoefBlk

This field is required if the general conversion method is used. This field and field 22 or field 20 shall be provided. If field 20 or 21 and 22 are not provided, the NCAP shall report a fatal TEDS error.

If field type 20 is included, fields 21 and 22 shall be omitted.

This field consists of two sub-blocks:

Cell number
Set of coefficients

8.6.3.23 Cell number

Field Type: 50

Field Name: CellNum

Data type: unsigned 16 bit integer (UInt16, 2 octets)

If field type 22 is included, this field is required. If field 22 is included and this field is omitted, the NCAP shall report a fatal TEDS error.

The cells are numbered from 0 to m. Cell 0 is the cell with the lowest-valued segments of all input TransducerChannels. Numbering continues, taking the next higher segment of domain X_n (the domain of values from the highest numbered element). Upon reaching the last segment for domain X_k, the next cell covers the lowest segment of domain X_k again, and the next higher segment of domain X_{k-1}. For example, for a two TransducerChannel correction with two segments on the first correction input (A1, A2) and three segments on the second correction input (B1, B2, B3). In the terms used in this TEDS, input A would be identified as Element 0 and input B as Element 1. The cells shall be numbered in the following segment order: (A1,B1)=0, (A1,B2)=1, (A1,B3)=2, (A2,B1)=3, (A2,B2)=4, and (A2,B3)=5.

Copyright © 2007 IEEE. All rights reserved.

8.6.3.24 Set of coefficients

Field Type: 51

Field Name: CoefSet

Data type: array of single-precision real (Float32)

This field is required. If field type 20 is used, it shall only contain two values, an intercept, b or C_0, followed by a slope, m or C_1. This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

The set of coefficients is a one-dimensional array (table) containing the coefficient for each term in the equation. The set of multinomial coefficients shall correspond to the cell identified by the cell number within the coefficient block.

NOTE—Each element in the array is identified by a subscripted name. If we name the elements C and give them subscripts, the first subscript represents the degree of the TransducerChannel with element number 0 and may vary between 0 and the value given for that element in the TransducerChannel Degree field. The second subscript is for the TransducerChannel with element number 1. This continues with the last subscript identifying the entry with the highest element number.

Each entry in the table is a coefficient $C_{i,j,\ldots p}$, used in the multinomial:

$$\sum_{i=0}^{D(1)} \sum_{j=0}^{D(2)} \cdots \sum_{p=0}^{D(n)} C_{i,j,\cdots,p} [X_1 - H_1]^i [X_2 - H_2]^j \cdots [X_n - H_n]^p \tag{15}$$

The coefficients shall be stored in the table beginning with $C_{0,0,\ldots,0}$ and incrementing the rightmost subscript. When any subscript reaches its limit, begin again at zero and increment the subscript to the next input to the left.

$C_{0,0,\ldots 0,0}$; $C_{0,0,\ldots 0,0}$; … $C_{0,0,\ldots 0,Dn}$

$C_{0,0,\ldots 1,0}$; $C_{0,0,\ldots 1,1}$; … $C_{0,0,\ldots D(n-1),Dn}$

…

$C_{D1,D2,\ldots D(n-1),0}$; $C_{D1,D2,\ldots D(n-1),1}$; … $C_{D1,D2,\ldots D(n-1),Dn}$

All coefficient blocks within a Calibration TEDS shall have the same length. Coefficients that are not required for one segment but that are required for another shall be set to zero.

8.7 Frequency Response TEDS

This field is an optional TEDS. The function of the Frequency Response TEDS shall be to make available the information regarding the frequency response of the TransducerChannel for the user.

The Frequency Response TEDS provides a characterization of the frequency and phase response of the TransducerChannel. A partial list of the factors affecting frequency response is sensor, analog signal conditioning, anti-aliasing filter, and digital signal processing. The TEDS gives the end-to-end response

Copyright © 2007 IEEE. All rights reserved.

from the analog sensor to the digital output. This characterization assumes that the TIM output is being read at a rate that supports the frequency response described.

The Frequency Response TEDS is normalized at the reference frequency; that is, it is implicitly assumed that the transducer data are referenced to this frequency.

8.7.1 Access

The Frequency Response TEDS is accessed using a Query TEDS command, a Read TEDS segment command, a write TEDS segment command, or a Update TEDS command. The argument of the command shall specify the TEDS access code of the Frequency Response TEDS, as defined in Table 17.

This TEDS may be implemented as a read-only TEDS or a read/write TEDS, at the manufacturer's option. However, if it is implemented as a read-only TEDS, the TransducerChannel write TEDS segment and TransducerChannel Update TEDS commands shall not apply.

8.7.2 Data block

Table 66 and Figure 18 show the data structure for the Frequency Response TEDS. Subsequent subclauses explain each data field in the structure.

Table 66—Structure of the Frequency Response TEDS data block

Field type	Field name	Description	Type	# octets
—		TEDS length	UInt32	4
0–2	—	Reserved	—	—
3	TEDSID	TEDS identifier	UInt8	4
4–9	—	Reserved	—	—
10	RefFreq	Reference frequency	Float32	4
11	RefAmp	Test amplitude	Float32	4
12	RefPhase	Phase at the reference frequency	Float32	4
The following fields comprise a structure that defines one data point. The structure is repeated n times, once for each data point				
13	Points	Points in the table	—	—
14–127	—	Reserved	—	—
128–255	—	Open to manufacturers	—	—
—	—	Checksum	UInt16	2

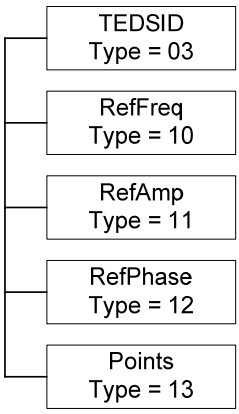


Figure 18—Frequency Response TEDS structure

Copyright © 2007 IEEE. All rights reserved.

8.7.2.1 TEDS identifier

The TEDS identifier shall comply with the structures defined in 8.3.

This field is required. If this field is omitted or contains illegal values, the NCAP shall report a fatal TEDS error.

8.7.2.2 Reference frequency

Field Type: 10

Field Name: RefFreq

Data type: single-precision real (Float32, 4 octets)

This field is required. If it is omitted, the NCAP shall report a fatal TEDS error.

This field identifies the reference frequency at which the amplitude is defined as being unity. The units on this field shall be Hertz.

8.7.2.3 Test amplitude

Field Type: 11

Field Name: RefAmp

Data type: single-precision real (Float32, 4 octets)

This field is required. If it is omitted, the NCAP shall report a fatal TEDS error.

This field identifies the input amplitude that was used to obtain the response information. The units on this field shall be the same as defined in the Physical units field of the TransducerChannel TEDS (see 4.11 or 8.5.2.6).

8.7.2.4 Phase at the reference frequency

Field Type: 12

Field Name: RefPhase

Data type: single-precision real (Float32, 4 octets)

This field is required. If it is omitted, the NCAP shall report a fatal TEDS error.

This field identifies the phase shift of the TransducerChannel output at the Reference frequency in field 2 (see 8.7.2.2). The units for this parameter shall be radians.

8.7.2.5 Points in the table

Field Type: 13

Copyright © 2007 IEEE. All rights reserved.

Field Name: Points

Data type: Three single-precision real (Float32, 4 octets) values (12 octets)

This field is required. If it is omitted, the NCAP shall report a fatal TEDS error.

This field defines a data point in the response table. Each data point is made up of three subfields:

Frequency
Amplitude response
Phase response

This field may be repeated for as many points as the manufacturer considers adequate to define the frequency response of the TransducerChannel.

Table 67 shows the structure of the TLV tuple for this field.

Table 67—Structure of each point

Field	Definition
Field type	Always 13 for this field.
Length	Always 12 because there are three floating-point numbers in the value field so 12 octets are required.
Frequency	This field identifies the frequency for which the amplitude and phase information in the following two fields are applicable. The units on this field shall be Hertz.
Amplitude	This field identifies the amplitude of the TransducerChannel output relative to the amplitude at the Reference frequency specified in field 2. The amplitude response is defined as the amplitude at the Frequency divided by the amplitude at the Reference frequency.
Phase	This field identifies the phase shift of the TransducerChannel output at the Frequency. The units for this parameter shall be radians.

8.8 Transfer Function TEDS

This field is an optional TEDS. It provides a series of constants that can be used to describe the transfer function of the transducer. Factors affecting the transfer function are sensor, analog signal conditioning, anti-aliasing filter, and digital signal processing. The transfer function gives the end-to-end response from the analog sensor to the digital output. It is intended to allow the NCAP or other element in the system to compensate for the frequency response of the transducer.

The Transfer Function TEDS is normalized at the reference frequency; i.e., it is implicitly assumed that the transducer data are referenced to this frequency.

8.8.1 Compensation process

When the transfer function describing the relation between the input and the output of the TransducerChannel is known, the inverse function may be used to linearize or compensate the frequency response function of the total system.

Frequency response compensation is the application of a specified mathematical function upon TransducerChannel data. This subclause gives an overview of how the compensation process is modeled, in order to aid understanding of how to develop and use the entries in the Transfer Function TEDS for compensation.

The goal of compensation depends on the TransducerChannel type. The application of compensation, however, is the same regardless of TransducerChannel type:

Copyright © 2007 IEEE. All rights reserved.

For sensors, compensation takes as input the transducer-side or pre-compensation data. It produces as output the NCAP-side or post-compensation number.

For actuators, compensation takes as input the NCAP-side number, that is, the intended next state of the actuator. The output is the transducer-side number compensated for the phase and frequency response of the circuit and actuator.

8.8.1.1 Method

The compensation function is defined as an inverse transfer function. If the transfer function is given as $H(f)$ for the TransducerChannel, then the TransducerChannel output should be multiplied by $1/H(f)$ to give the compensated output. (This requires that there are no zeros in the transfer function within the frequency range of application.)

$H(f)$ is represented as a factorized product of a number of elements normalized at the reference frequency specified in the Reference frequency field of this TEDS (see 8.8.3.2). Equation (16) is the mathematical representation of this function.

$$H(f) = \frac{H_1(f)}{\left|H_1(f_{ref})\right|}\frac{H_2(f)}{\left|H_2(f_{ref})\right|} * .... * \frac{H_N(f)}{\left|H_N(f_{ref})\right|} \tag{16}$$

forces $\left|H(f_{ref})\right| = 1$ but maintains the phase at the reference frequency. The form of each element $H_i(f)$ shall be as described in 8.8.3.1 through 8.8.3.23.

Alternatively, $H(f)$ is described as a rational function, normally referred to as the z-transform, as shown in Equation (17).

$$H(z) = \frac{A_0 + A_1 z^{-1} + A_2 z^{-2} + ...A_{\mathrm{n}} z^{-\mathrm{n}}}{1 + B_0 + B_1 z^{-1} + B_2 z^{-2} ....B_{\mathrm{m}} z^{-\mathrm{m}}} \tag{17}$$

where

$$z^{-1}(\omega) = \mathrm{e}^{-j\omega \mathrm{T}}$$
$$\omega = 2\pi f$$
$$j = \sqrt{-1}$$

and T is a time constant that is the delay or time between samples.

8.8.1.2 Application

When the stream of data coming from the TransducerChannel is processed (for example, by a fast Fourier transformation into the frequency domain), the resulting spectrum may be divided by $H(f)$ to give the corrected spectrum. An inverse transformation can bring this back to the time domain. From $H(f)$, other functions like the impulse response function can be derived. This may then be used directly on the time data because a convolution makes the desired correction.

A digital filter may also be used to make the compensation. From the z-transform, this is straightforward. The factorized form may also be used to calculate the needed coefficients.

Copyright © 2007 IEEE. All rights reserved.

8.8.2 Access

The Transfer Function TEDS is accessed using a Query TEDS command, a Read TEDS segment command, a write TEDS segment command, or a Update TEDS command.The argument of the command shall specify the TEDS access code of the Transfer Function TEDS, as defined in Table 17.

This TEDS may be implemented as a read-only TEDS or a read/write TEDS, at the manufacturer's option. However, if it is implemented as a read-only TEDS, the TransducerChannel write TEDS segment and TransducerChannel Update TEDS commands shall not apply.

8.8.3 Data block

Table 68 and Figure 19 show the structure of the data block for this TEDS. Subclauses 8.8.3.1 through 8.8.3.23 explain each data field in the structure. The Reference frequency field is always required. The remaining fields in this TEDS are required or not depending on the method chosen to define the transfer function.

8.8.3.1 TEDS identifier

See the description of the TEDS Header in 8.3

This field is required. If this field is omitted or contains illegal values, the NCAP shall report a fatal TEDS error.

8.8.3.2 Reference frequency

Field Type: 10

Field Name: RefFreq

Data type: single-precision real (Float32, 4 octets)

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

This field identifies the reference frequency at which the amplitude is defined as being 1. The units on this field shall be Hertz.

Copyright © 2007 IEEE. All rights reserved.

Table 68—Structure of the Transfer Function TEDS data block

Field	Field name	Description	Type	# octets
—		TEDS length	UInt32	4
0–2	—	Reserved	—	—
3	TEDSID	TEDS identification header	UInt8	4
4–9	—	Reserved	—	—
Transfer function related information				
10	RefFreq	Reference frequency	Float32	4
11	OneZero	Single zero TF_SZ	Float32	4
12	OnePole	Single pole TF_SP	Float32	4
13	ZeroPole	Single zero with a dependant pole	—	—
14	PoleZero	Single pole with a dependent zero	—	—
15	ComplexZ	Complex zero	—	—
16	ComplexP	Complex pole	—	—
17	OneZZPol	Single zero at zero and a single pole	—	—
18	CRSlope	Constant Relative Slope	Float32	4
19	SampleT	Sample/Delay Time	Float32	4
20	DependP	Single pole dependent on a zero TF_SPm (x)	Float32	4
21	DependZ	Single zero dependent on a pole TF_SZm (x)	Float32	4
22	ComplexZF	Complex zero frequency	Float32	4
23	ComplexZQ	Complex zero quality factor	Float32	4
24	ComplexPF	Complex pole frequency	Float32	4
25	ComplexPQ	Complex pole quality factor	Float32	4
26–29	—	Reserved	—	—
30	NCoeff	Numerator coefficients (A0, A1, … An)	Array of Float32	n*4
31	DCoeff	Denominator coefficients (B0, B1, … Bm)	Array of Float32	m*4
32–127	—	Reserved	—	—
12–255	—	Open for manufacturers use	—	—
		Checksum		
NOTE—The number of coefficients, n or m, for fields 30 and 31 may be determined by dividing the tuple length by 4.				

Copyright © 2007 IEEE. All rights reserved.

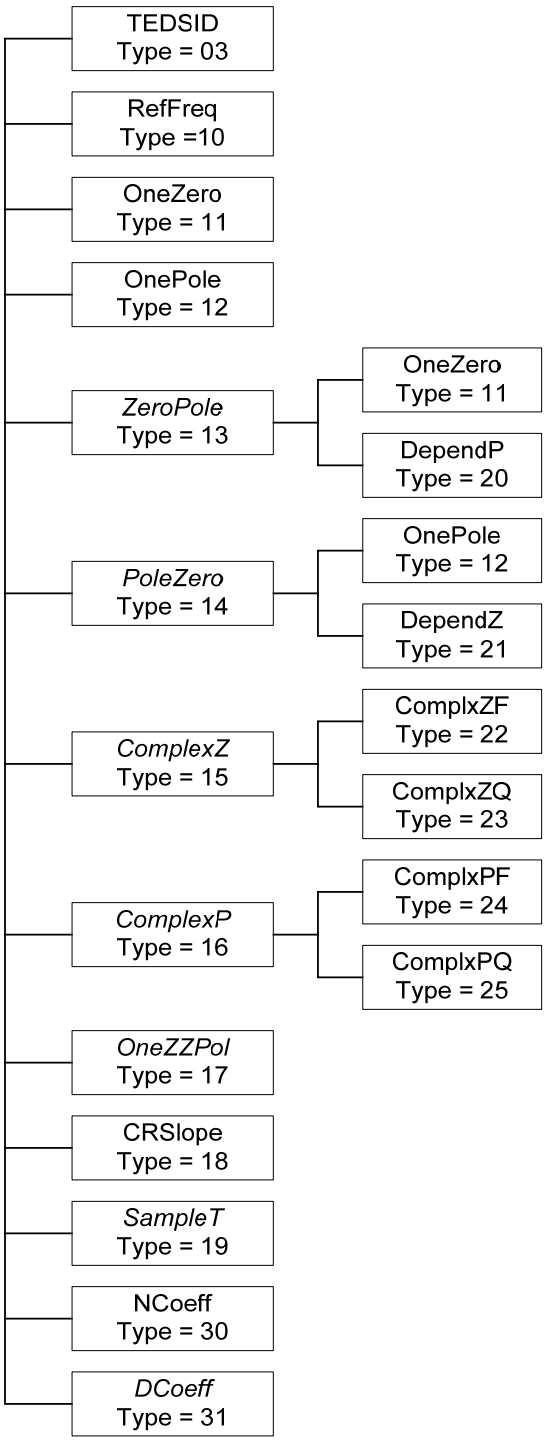


Figure 19—Transfer Function TEDS structure

Copyright © 2007 IEEE. All rights reserved.

8.8.3.3 Single zero

Field Type: 11

Field Name: OneZero

Data type: single-precision real (Float32, 4 octets)

This field is optional. If this field is omitted, the transfer function does not contain a single zero.

The value is used as F_{sz} in the transfer function shown in Equation (18), which directly describes a high-pass filter of first order with the –3 dB point at F_{sz}.

$$H(f, F_{SZ}) = (1 + \frac{j.f}{F_{SZ}}) \tag{18}$$

8.8.3.4 Single pole

Field Type: 11

Field Name: OnePole

Data type: single-precision real (Float32, 4 octets)

This field is optional. If this field is omitted, the transfer function does not contain a single pole.

The value is used as F_{sp} in the transfer function shown in Equation (19) that directly describes a low-pass filter of first order with the –3 dB point at F_{sp}.

$$H(f, F_{SP}) = \frac{1}{(1 + \frac{j.f}{F_{SP}})} \tag{19}$$

8.8.3.5 Single zero of a zero/pole pair

Field Type: 13

Field Name: ZeroPole

This field is optional. If this field is omitted, the transfer function does not contain a zero/pole pair.

This field is composed of two subfields. One is the same as the single zero described in 8.8.3.3, and the other is the single pole dependent on a zero as described in 8.8.3.7.

8.8.3.6 Single zero

This field is the same as the field single zero (see 8.8.3.3).

Copyright © 2007 IEEE. All rights reserved.

This field is required if field 13 is present. If this field is omitted and field 13 is present, the NCAP shall report a fatal TEDS error.

8.8.3.7 Single pole dependent on a zero

Field Type: 20

Field Name: DependP

Data type: single-precision real (Float32, 4 octets)

This field is required if field 13 is present. If this field is omitted and field 13 is present, the NCAP shall report a fatal TEDS error.

Each value in this field is used as "*x*" in the transfer function given in Equation (20).

$$H(f,x,v) = \frac{1}{(1+\frac{j \cdot f}{x \cdot v})} \tag{20}$$

where *v* is the value "F_{sz}" from the associated element in the single zero of a zero/pole pair field (see 8.8.3.5).

8.8.3.8 Single pole with a dependent zero

Field Type: 14

Field Name: PoleZero

This field is optional. If this field is omitted, the transfer function does not contain a pole/zero pair.

This field is composed of two subfields. One is the same as the single pole described in 8.8.3.3, and the other is the Single zero dependent on a pole as described in 8.8.3.10.

8.8.3.9 Single pole

This field is the same as the single pole field described in 8.8.3.3.

This field is required if field 14 is present. If this field is omitted and field 14 is present, the NCAP shall report a fatal TEDS error.

8.8.3.10 Single zero dependent on a pole

Field Type: 21

Field Name: DependZ

Data type: single-precision real (Float32, 4 octets)

This field is required if field 14 is present. If this field is omitted and field 14 is present, the NCAP shall report a fatal TEDS error.

Copyright © 2007 IEEE. All rights reserved.

The value in this field is used as "*x*" in the transfer function shown in Equation (21).

$$H(f,x,v) = (1 + \frac{j \cdot f}{x \cdot v}) \tag{21}$$

where *v* is the value "F_{sp}" from the associated element in the single pole with a dependent zero field.

8.8.3.11 Complex zero

Field Type: 15

Field Name: ComplexZ

This field is optional. If this field is omitted, the transfer function does not contain a complex zero.

This field has two subfields, Complex zero frequency and Complex zero quality factor.

8.8.3.12 Complex zero frequency

Field Type: 22

Field Name: ComplexZF

Data type: array of single-precision real (Float32, 4 octets each).

This field is required if field 15 is present. If this field is omitted and field 15 is present, the NCAP shall report a fatal TEDS error.

This value may be used as F_{zres} in the transfer function given in Equation (22).

$$H(f,F_{\text{zres}},Q_{\text{z}}) = (1 + \frac{j \cdot f}{Q_{\text{z}} \cdot F_{\text{zres}}} + (\frac{j \cdot f}{F_{\text{zres}}})^2) \tag{22}$$

where parameter Q_z has to be given in the associated element of the Complex zero quality factor field.

The units on this field shall be Hertz.

8.8.3.13 Complex zero quality factor

Field Type: 23

Field Name: ComplexZQ

Data type: single-precision real (Float32, 4 octets).

This field is required if field 15 is present. If this field is omitted and field 15 is present, the NCAP shall report a fatal TEDS error.

This field provides the parameter Q_z for the transfer function of a complex zero in the associated element of the Complex zero field.

Copyright © 2007 IEEE. All rights reserved.

This quantity is “unitless.”

8.8.3.14 Complex pole

Field Type: 16

Field Name: ComplexZ

This field is optional. If this field is omitted, the transfer function does not contain a complex pole.

This field has two subfields, Complex pole frequency and Complex pole quality factor.

8.8.3.15 Complex pole frequency

Field Type: 24

Field Name: ComplexPF

Data type: single-precision real (Float32, 4 octets).

This field is required if field 16 is present. If this field is omitted and field 16 is present, the NCAP shall report a fatal TEDS error.

This value may be used as F_{zres} in the transfer function.

$$H(f, F_{\text{zres}}, Q_{\text{p}}) = (1 + \frac{j \cdot f}{Q_{\text{p}} \cdot F_{\text{zres}}} + (\frac{j \cdot f}{F_{\text{zres}}})^2) \tag{23}$$

where the parameter Q_{p} has to be given in the associated element of the array found in the Complex pole quality factor field (see 8.8.3.16).

The units on this field shall be Hertz.

8.8.3.16 Complex pole quality factor

Field Type: 25

Field Name: ComplexPQ

Data type: single-precision real (Float32, 4 octets).

This field is required if field 16 is present. If this field is omitted and field 16 is present, the NCAP shall report a fatal TEDS error.

This field gives the parameter Qp for the transfer function of a complex pole. This is typically used to describe the behavior of a single degree-of-freedom system like the spring mass system in an accelerometer or the membrane with air damping in a microphone. The parameters are the resonance frequency and Q (or quality factor) of the response curve.

This quantity is “unitless.”

Copyright © 2007 IEEE. All rights reserved.

8.8.3.17 Single zero at zero and a single pole

Field Type: 17

Field Name: OneZZPol

Data type: single-precision real (Float32, 4 octets).

This field is optional. If this field is omitted, the transfer function does not contain a Single zero at zero and a single pole.

This value may be used as F_{hp} in the transfer function shown in Equation (24).

$$H(f, F_{hp}) = \frac{\frac{j \cdot f}{F_{hp}}}{(1 + \frac{j \cdot f}{F_{hp}})} \tag{24}$$

directly describes a high-pass filter of first order with the –3 dB point at F_{hp}. The units on this field shall be Hertz.

8.8.3.18 Constant relative slope

Field Type: 18

Field Name: CRSlope

Data type: single-precision real (Float32, 4 octets).

This field is optional. If this field is omitted, the transfer function does not contain a constant relative slope.

The value in this field is used as a in the transfer function given in Equation (25).

$$H(f, a, F_{ref}) = (\frac{j \cdot f}{F_{ref}})^{\frac{a}{\ln(10)}} \tag{25}$$

where a is the relative change per frequency decade.

F_{ref} is found in the Reference frequency field. (Typically most piezoelectric ceramics (lead titanate-zirconate) have a value of a in the order of –0.02 or 2% drop in sensitivity per frequency decade. This is linked to simultaneous frequency independent phase lag, which is found to be –0.78 degrees.)

8.8.3.19 Sample/delay time

Field Type: 19

Field Name: SampleT

Data type: single-precision real (Float32, 4 octets).

Copyright © 2007 IEEE. All rights reserved.

This field is optional. If this field is omitted, the transfer function does not contain a digital filter.

This is the time between samples or the delay time used in a digital filter. The units for this field shall be seconds.

8.8.3.20 Numerator coefficients

Field Type: 30

Field Name: NCoeff

Data type: array of single-precision real (Float32, 4 octets each).

This field is optional. If this field is omitted, the transfer function does not contain a "z" transform. If this field is not present but field 31 is present, the NCAP shall report a fatal TEDS error.

When the alternative form of the transfer function (the "z" transform) is used, this field contains the list of coefficients required in the numerator of the equation. See the z-transform discussion in 8.8.1.1 for more details.

NOTE—The number of coefficients in this field may be obtained by dividing the "tuple length" by four.

8.8.3.21 Denominator coefficients

Field Type: 31

Field Name: DCoeff

Data type: array of single-precision real (Float32, 4 octets each).

This field is optional. If this field is omitted, the transfer function does not contain a "z" transform. If this field is not present but field 30 is present, the NCAP shall report a fatal TEDS error.

When the alternative form of the transfer function (the "z" transform) is used, this field contains the list of coefficients required in the denominator of the equation. See the ztransform discussion in 8.8.1.1 for more details.

NOTE—The number of coefficients in this field may be obtained by dividing the "tuple length" by four.

8.9 Text-based TEDS

This field is a class of optional TEDS. The function of these TEDS is to provide information for display to an operator. Six TEDS are listed in Table 17 that fall into this category. They are the Meta-Identification TEDS, TransducerChannel Identification TEDS, Calibration-Identification TEDS, Commands TEDS and the Location and Title TEDS, and the Geographic Location TEDS. For general descriptions of these TEDS, see 5.5.2.6 through 5.5.2.9.

Copyright © 2007 IEEE. All rights reserved.

8.9.1 Access

They are accessed using a Query TEDS command, a Read TEDS segment command, a write TEDS segment command, or an Update TEDS command. The argument of the command shall specify the TEDS access code of the TEDS as defined in Table 17.

These TEDS may be implemented as read-only TEDS or as read/write TEDS at the manufacturer's option. If it is implemented as a read-only TEDS, the TIM write TEDS segment or TransducerChannel write segment and TIM Update TEDS or TransducerChannel Update TEDS commands shall not apply.

8.9.2 Data block

Text-based TEDS are structures that encapsulate one or more blocks of textual information. Each block of text is presented in a specific language. The first part of this TEDS is a directory to allow a processor to locate and read a single language. The field XMLText contains the displayable information, encoded as XML. The manufacturer determines the number of languages implemented. Table 69 and Figure 20 show the contents of this TEDS.

Table 69—Structure of a Text-based TEDS data block

Field type	Field name	Description	Type	# octets
—		TEDS length	UInt32	4
0–2	—	Reserved	—	—
3	TEDSID	TEDS identification header	UInt8	4
4–9	—	Reserved	—	—
The following three fields comprise a language header. The header is repeated N times, once for each language supported.				
10	NumLang	The number N of different language blocks in this TEDS	UInt8	1
11	DirBlock	Language block description This block is repeated N times	—	—
20	LangCode	Language code from ISO 639 (two letters in USASCII)	UInt8	2
21	Offset	Language offset	UInt32	4
22	Length	Language length = LL	UInt32	4
23	Compress	Enumeration identifying the compression technique used	UInt8	1
12	SubSum	Nondisplayable data checksum	UInt16	2
The following two fields comprise a structure containing text in one language. The structure is repeated N times, once for each language defined.				
—	XMLText	XML-based text block	text	LL - 2
—	XMLSum	Text block checksum	UInt16	2
13–19	—	Reserved	—	—
23–127	—	Reserved	—	—
128–255	—	Open to manufacturers	—	—
—		Checksum	UInt16	2

Copyright © 2007 IEEE. All rights reserved.

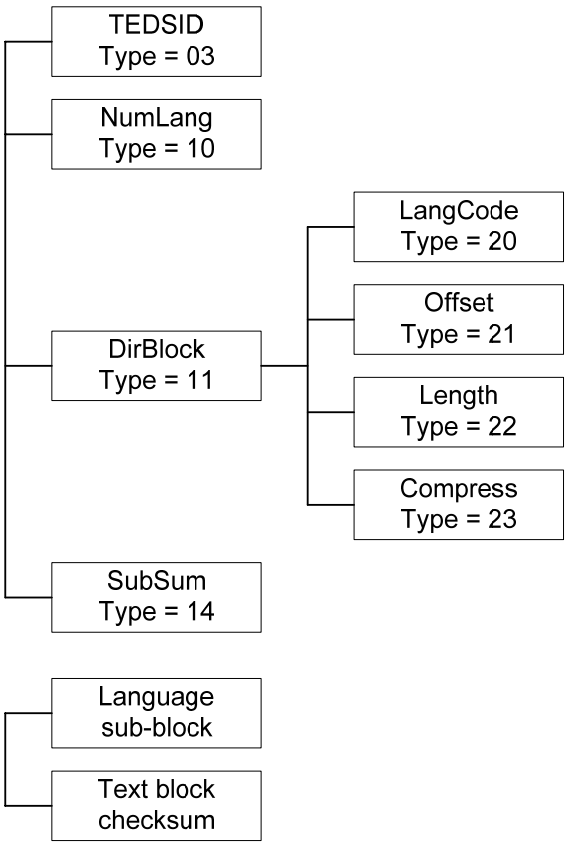


Figure 20—Text-based TEDS structure

8.9.2.1 TEDS identifier

The TEDS identifier shall comply with the structures defined in 8.3.

This field is required. If this field is omitted or contains illegal values, the NCAP shall report a fatal TEDS error.

8.9.2.2 Number of languages

Field Type: 10

Field Name: NumLang

Data type: This field contains a single octet used for counting (UInt8, 1 octet).

If this field is omitted, the NCAP shall assume that only one language is present.

This field contains a number identifying the number of languages in the TEDS.

Copyright © 2007 IEEE. All rights reserved.

8.9.2.3 Language directory block

Field Type: 11

Field Name: DirBlock

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

This field is made up of subfields as follows:

Language code

Language offset

Language sub-block length

Compression technique enumeration

8.9.2.4 Language code

Field Type: 20

Field Name: LangCode

Data type: A two-octet character string encoded in USASCII.

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

This field indicates the language in which the text fields in the TEDS are written.

The value corresponds to the alphabetic list of two-letter language symbols in the ISO 639: 1988-04-01 (E/F) standard. Table 70 lists some of the possible languages. Languages and dialects not listed in ISO 639 shall not be used in the TEDS text fields.

Table 70—Examples of enumerations of language codes

ISO 639 language code	Meaning (informative)
Reserved	—
aa	Afar
da	Danish
de	German
en	English
es	Spanish
eu	Basque
fi	Finnish
fr	French
ga	Irish
it	Italian
nl	Dutch
no	Norwegian
pl	Polish
pt	Portuguese
ru	Russian
sv	Swedish
vi	Vietnamese
zu	Zulu

Copyright © 2007 IEEE. All rights reserved.

8.9.2.5 Language offset

Field Type: 21

Field Name: Offset

Data type: Unsigned 32-bit integer for counting (UInt32, 4 octets)

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

This field is used to locate the displayable information and is not normally displayed.

This field indicates the offset from the beginning of the TEDS at which the XML-related information data sub-block is located for the specified language.

8.9.2.6 Language sub-block length

Field Type: 22

Field Name: Length

Data type: Unsigned 32-bit integer for counting (UInt32, 4 octets)

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

This field is used to locate the displayable information and is not normally displayed.

This field indicates the number of octets in the language sub-block, including the check sum. (In the case of character fields, the length shall be the number of octets and not the number of characters, since more than 1 octet may be needed to encode a character. The interpretation of the character strings needs the number of octets per character to determine the length in characters of a particular string.)

8.9.2.7 Compression technique enumeration

Field Type: 23

Field Name: Compress

Data type: unsigned octet integer (UInt8, 1 octet)

This field is optional. If this field is omitted, the system shall assume that no compression is employed.

This field is used to identify the compression algorithm used with this language text block. Table 71 lists the acceptable values for this field.

Copyright © 2007 IEEE. All rights reserved.

Table 71—Enumeration identifying compression algorithms

Enumeration	Description
0	This enumeration means that no compression is used in this language block of this TEDS
1	WinZip
2	GZip
3	Reserved
4–127	Reserved
128–255	Open to manufacturers

8.9.2.8 Non-displayable data checksum

Field Type: 14

Field Name: SubSum

Data type: Unsigned 16 bit integer for counting (UInt16, 2 octets)

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

This field contains the checksum for all octets preceding it in the TEDS. The checksum shall be the one's complement of the sum (modulo 2^{16}) of all the data structure's preceding octets, including the Length (see 8.1.1) field and excluding the Non-displayable data checksum field.

8.9.2.9 XML-based text block

Data type: text

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

This field contains the information to be displayed by an XML-savvy application. A proposed schema for all text-based TEDS used in IEEE Std 1451.2-1997 or described in this standard is presented in Annex D.

8.9.2.10 Text block checksum

This field contains the checksum for all octets in the preceding XML-based text block field. The checksum shall be the one's complement of the sum (modulo 2^{16}) of all the octets in the XML text block.

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

8.10 End User Application Specific TEDS

This field is an optional TEDS that provides storage for application-dependent data that the user wants to keep with the TIM or TransducerChannel. The user shall determine the content and function of the End User Application Specific TEDS. This TEDS shall be able to be both read and written.

8.10.1 Access

The End User Application Specific TEDS may be associated with the TIM or a TransducerChannel. It is accessed using a Query TEDS command, a Read TEDS segment command, a write TEDS segment

Copyright © 2007 IEEE. All rights reserved.

command, or an Update TEDS command. The argument of the command shall specify the End User Application Specific TEDS access code as defined in Table 17.

8.10.2 Data block

Table 72 shows the structure of the data block for this TEDS. The content and structure of the Data block field is user defined.

Table 72—Structure of the End User Application Specific TEDS data block

Field	Description	Type	Type	# octets
—	TEDS length	UInt32	UInt32	4
0–2	—	Reserved	—	—
3	TEDSID	TEDS identification header	UInt8	UInt8
4–9	—	Reserved	—	—
10	EndUserData	Variable		Variable
—	Checksum	UInt16		2

The manufacturer shall determine the size of this TEDS. It is recommended that the size of this TEDS accommodate a Data block field of at least 256 octets.

8.10.2.1 TEDS identifier

The TEDS identifier shall comply with the structures defined in 8.3.

This field is required. If this field is omitted or contains illegal values, the NCAP shall report a fatal TEDS error.

8.10.2.2 Data block content

Field Type: 10

Field Name: EndUserData

The contents of the data block are up to the user and are not defined in this standard.

8.11 User's Transducer Name TEDS

This field is a required TEDS for the TIM and is recommended for all Transducers. The User's Transducer Name TEDS provides a place to store the name by which the system or the end user will know this transducer.

NOTE—The User's Transducer Name TEDS is intended to support "Object Tags," as defined in IEEE Std 1451.1-1999 or other similar uses.

8.11.1 Access

The User's Transducer Name TEDS may be associated with the TIM or a TransducerChannel. It is accessed using a Query TEDS command, a Read TEDS segment command, a write TEDS segment command, or an Update TEDS command. The argument of the command shall specify the TEDS access code of the User's Transducer Name TEDS, as defined in Table 17.

Copyright © 2007 IEEE. All rights reserved.

This TEDS shall be implemented as read/write TEDS.

8.11.2 Data block

Table 73 shows the structure of the data block for this TEDS. The content and structure of the Data block field is user defined.

Table 73—Structure of the user transducer name TEDS data block

Field type	Field name	Description	Data type	# octets
—		Length	UInt32	4
0–2	—	Reserved	—	—
3	TEDSID	TEDS Identification Header	UInt8	4
4	Format	Format description of this TEDS	UInt8	1
5	TCName	TIM or TransducerChannel Name	—	
—		Checksum	UInt16	2

The manufacturer shall determine the size of this TEDS, but as a minimum, the TEDS shall be large enough to accommodate a Data block field of at least 160 octets with a 32-character parameter name plus a TEDS identification header and a possible text-based TEDS header. The user is not required to use the text-based TEDS header.

8.11.2.1 TEDS identifier

The TEDS identifier shall comply with the structures defined in 8.3.

This field is required. If this field is omitted, the NCAP shall report a fatal TEDS error.

8.11.2.2 Format

Field Type: 10

Field Name: Format

Data type: unsigned octet integer (UInt8, 1 octet)

This field is required. If this field is omitted or contains illegal values, the NCAP shall report a fatal TEDS error.

The values for format flag are defined in Table 74.

Table 74—Enumeration of format flag field

Value	Meaning
0	User defined
1	Text-based TEDS using the format defined in 8.9
2–255	Reserved for future expansion

If the format field indicates that this data block is user defined, the content and structure of the Data block field is user defined. If it is text based, then the content of this data block shall conform with the structure defined in 8.9.

NOTE—The use of multiple languages in this TEDS may cause problems with a user's application.

Copyright © 2007 IEEE. All rights reserved.

8.11.2.3 Data block content

Field Type: 11

Field Name: EndUserData

The contents of the data block are up to the user and are not defined in this standard.

8.12 Manufacturer-defined TEDS

Manufacturer-defined TEDS may be in any format required by the manufacturer's application software. A generic system shall not attempt to parse these TEDS or to interpret their content in any manner. Unless the TEDS is text based as determined by the response to the Query TEDS command (see 7.1.1.1), the system shall simply read these TEDS and shall pass the contents to the application that called it. If the TEDS is text based, it shall conform to the structure defined in 8.9. For a manufacturer-defined TEDS that is being sent to the TIM, the system shall take the information, apply the length field and checksum fields, and transmit it to the TIM.

8.12.1 Access

A manufacturer-defined TEDS may be associated with the TIM or a TransducerChannel. It is accessed using a Query TEDS command, a Read TEDS segment command, a write TEDS segment command, or an Update TEDS command. The argument of the command shall specify the TEDS access code of the TEDS, as defined by the manufacturer.

This TEDS may be implemented as a read-only TEDS or as a read/write TEDS at the manufacturer's option. If it is implemented as a read-only TEDS, the TIM write TEDS segment or TransducerChannel write segment and TIM Update TEDS or TransducerChannel Update TEDS commands shall not apply.

8.12.2 Data block

The content and structure of this TEDS is defined by the manufacturer and is out of scope for this standard.

8.12.2.1 TEDS identifier

The TEDS identifier shall comply with the structures defined in 8.3.

This field is required. If this field is omitted or contains illegal values, the NCAP shall report a fatal TEDS error.

8.12.2.2 Data block contents

The contents and structure of this field are controlled by the manufacturer.

Copyright © 2007 IEEE. All rights reserved.

8.13 PHY TEDS

The PHY TEDS is a required TEDS. The function of the PHY TEDS shall be to make available at the interface all of the information needed to gain access to any channel, plus the information common to all channels. PHY TEDS octets are constant and read-only.

This TEDS is not described further in this standard. See the particular IEEE 1451 standard you are interested in to find the details for this TEDS.

8.13.1 Access

The PHY TEDS is accessed using a Query TEDS command, a Read TEDS segment command, a Write TEDS segment command, or an Update TEDS command. The argument of the command shall specify the TEDS access code of the PHY TEDS, as defined in Table 17.

This TEDS should be implemented as a read-only TEDS to prevent it from being changed in the field because changes could cause unpredictable behavior. If it is implemented as a read-only TEDS, the TIM write TEDS segment and TIM Update TEDS commands shall not apply.

8.13.2 Data block

The content and structure of this TEDS is defined in another member of the IEEE 1451 family of standards and is out of scope for this standard.

8.13.2.1 TEDS identifier

The TEDS identifier shall comply with the structures defined in 8.3.

This field is required. If this field is omitted or contains illegal values, the NCAP shall report a fatal TEDS error.

8.13.2.2 Data block contents

The contents and structure of this field are controlled by other standards in the IEEE 1451 family of standards.

9. Introduction to the IEEE 1451.0 API

Two APIs are defined in this standard. This clause defines the common aspects of the APIs. The “Transducer Services Interface” defined in Clause 10 is an NCAP-only API used by measurement and control applications to access the IEEE 1451.0 layer. This API contains methods to read and write TransducerChannels, read and write TEDS, and send configuration, control, and operation commands to the TIMs. Optionally, an interface is also defined that would be implemented by the application to support non-blocking read and write operations and to receive data from measurement streams.

These API definitions are provided for systems that have visible interfaces and that the structure for monolithic TIMs and NCAPs, i.e., those with a single set of hardware and software without regard to distinguishing separate interfaces between IEEE 1451.0 functionality and IEEE 1451.X functionality, is

Copyright © 2007 IEEE. All rights reserved.

optional as long as the messages at visible interfaces conform to the rest of the standard. The definition of these APIs is to facilitate modular design to the extent that multiple suppliers can provide different functionality and yet have the various parts integrate seamlessly.

The "Module Communication Interface" is between the standard and another member of the IEEE 1451 family. It is a symmetric interface that would be implemented on both the NCAP and TIM sides. This API contains methods that would be implemented by the IEEE 1451.X layer and called by this standard to initiate communication operations. Similarly, this API contains methods that would be implemented by this standard that are called by the IEEE 1451.X layer to deliver communication payloads.

The relationship between the interfaces and other entities in the IEEE 1451 family is illustrated in the reference model in Figure 1 and Figure 2.

9.1 API goals

The basic goals to be achieved by the use of these API are as follows:

Provide APIs that are well matched to the needs of IEEE 1451-based measurement systems composed of NCAPs and TIMs.

Provide APIs to simplify the interaction between measurement and control applications on the NCAP and TIMs. Key services are as follows:

- TIM discovery
- Transducer access
- Transducer management
- TEDS management

Provide a communication abstraction that is independent of the underlying IEEE 1451.X technology.

Accommodate the wide range of known IEEE 1451.X native communication technologies, and allow IEEE 1451.X groups to use the most appropriate mechanism.

Accommodate the wide range of known CPU and RAM memory resources available on NCAPs and TIMs, which includes PCs functioning as NCAPS to very low-end "PIC-like" 8 bit microprocessors.

Provide escape mechanisms where the IEEE 1451.X layer can intercept a communication invocation and thereby bypass network operations.

Provide a pass-through mechanism where knowledgeable applications can send custom commands through IEEE 1451.0 layers to be handled by the local or remote IEEE 1451.X layers.

Provide an escape mechanism where applications can send proprietary commands "straight through" to the application-specific side of the TIM without interpretation in the IEEE 1451.0 or IEEE 1451.X subsystems.

The interface to services described in this clause is presented as operations. Operation signatures are presented using a variant of the Interface Definition Language (IDL), defined in ISO/IEC 14750: 1999-03-15. The variation is summarized as follows: The IDL signature of an operation described in this clause shall only use data types defined within this standard. All IDL specifications within this clause are prefaced with "IDL:" in bold type to facilitate automated extraction from an electronic copy of this standard.

Copyright © 2007 IEEE. All rights reserved.

9.2 API design decisions

9.2.1 API described in IDL and text

To meet the language-neutral goal, IDL will be used to describe the API's functions, parameters, and results. An accompanying textual description will be provided to handle calling semantics.

Figure 21 illustrates the top-level structure of the IEEE 1451.0 APIs. A top-level IDL module named "IEEE1451Dot0" is defined. The "nested modules" listed in Table 75 are defined.

Table 75—Modules in the API

Module	Description
TransducerServices	This is the public API that measurement and control applications use to interact with the IEEE 1451.0 layer. It contains classes and interfaces for discovering registered TIMs, accessing TransducerChannels to make measurements or write actuators, managing TIM access, and reading and writing TEDS.
ModuleCommunications	This is the API that IEEE 1451.X implementers use to provide NCAP to TIM communications. Both "point-to-point" and "network" interfaces and callbacks are specified.
Args	This package contains all IEEE 1451.0 arguments. The ArgumentArray data structure is defined here.
Util	This package contains utility classes and interfaces for the conversion of ArgumentArrays to/from OctetArrays. This is provided via the Codec interface. IEEE 1451.X implementers that need to change the encoding would register alternative Codecs.

Copyright © 2007 IEEE. All rights reserved.

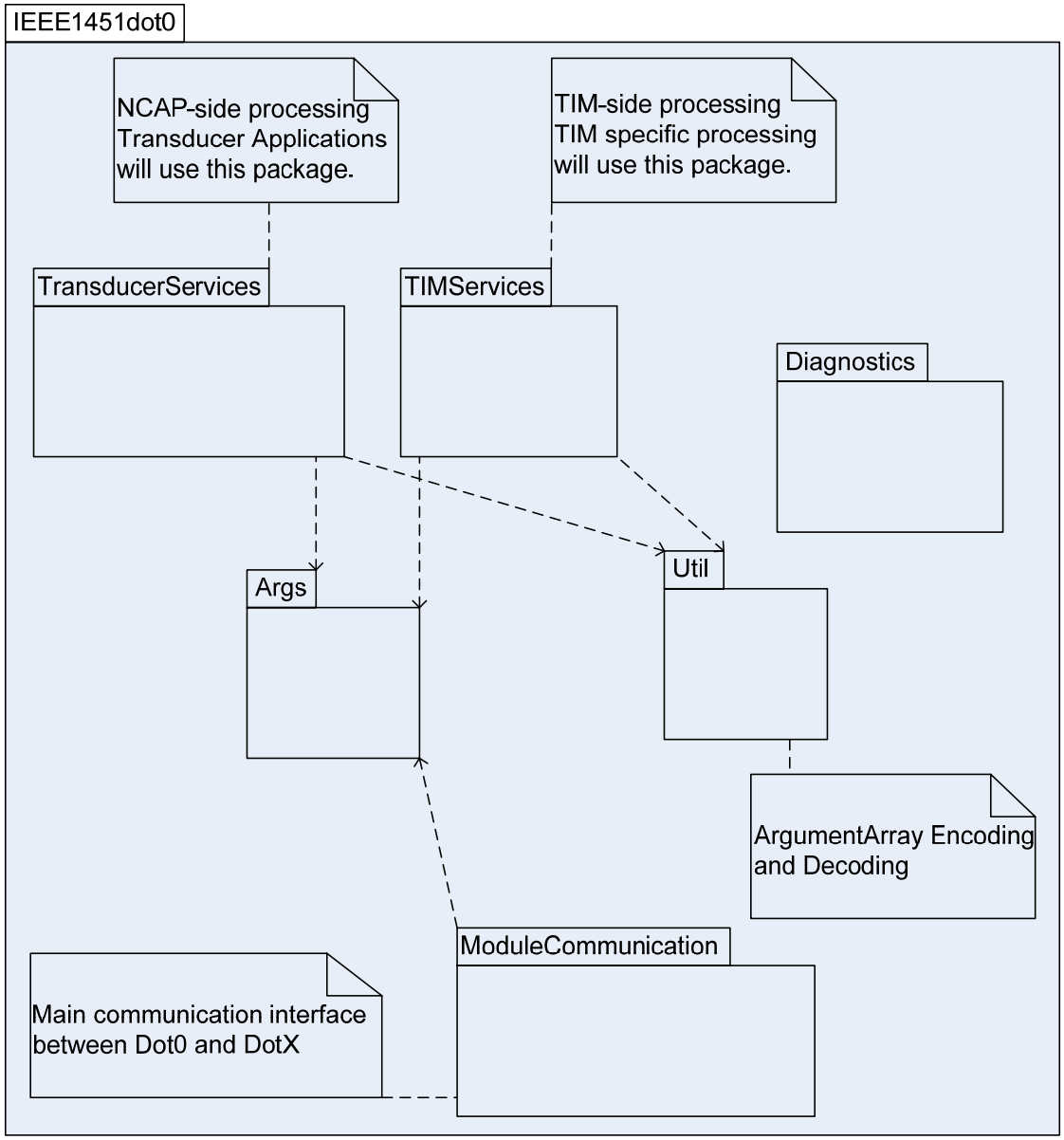


Figure 21—Structure of the API

9.2.2 IEEE 1451.0 OctetArray Payloads

To minimize the knowledge that IEEE 1451.X needs to be concerned with, all transmitted information from the perspective of this standard will be bundled together into a payload. The payload will be encoded as an OctetArray.

Except in cases where IEEE 1451.X wishes to intercept an IEEE 1451.0 message, IEEE 1451.X should consider the payload as "opaque." See 6.2 and 6.3 for the location of the length information necessary for decoding of the data. This is handled inside the encoder/decoder class.

From the perspective of this standard, the OctetArray and destination addressing represents the logical "on-the-link" format. IEEE 1451.X is expected to package the OctetArray into appropriate network packets for

Copyright © 2007 IEEE. All rights reserved.

the given IEEE 1451.X technology. For example, appropriate network headers and CRCs may be added. Also, IEEE 1451.X is responsible for segmenting the OctetArray into appropriately sized network packets and for reassembling them back into the OctetArray on the remote node. Similarly, all issues with encryption, authentication, compression, and flow control are IEEE 1451.X responsibilities.

9.2.3 IEEE 1451.0 IDL data structures and ArgumentArray usage

To facilitate the use of payloads in the IEEE 1451.0 and upper measurement application layers, this standard also specifies IDL data structures. Specific implementations will map the IDL data structures to appropriate language-dependent data structures for the implementation language at hand (for example, a C structure or C++ or Java class). These language-dependent data structures guarantee proper octet alignment to allow straightforward access to all data attributes. For example, alignment of floating-point numbers on 4-octet memory boundaries will be needed on most platforms.

Following the example of IEEE Std 1451.1-1999, the most common data structure inside IEEE 1451.0 and application layers is the generic ArgumentArray, which is an array of Arguments. Arguments are provided for all the basic types (for example, UInt8, UInt16, or Float32) and primitive arrays of basic types (for example, UInt8Array, Float32Array, or StringArray). In addition, some specific Arguments that are useful in measurement applications are also defined (for example, Units, TimeInstance, and TimeDuration).

The ArgumentArray data structure is a very flexible mechanism to compose and pass arbitrary data types through the system without requiring compile time knowledge. For example, a measurement from a TransducerChannel will be encoded into an appropriate argument based on information provided in the TEDS (for example, data model, data repetitions, and calibration information). This argument will be contained within an ArgumentArray of length 1. A more complex example would be reading from a TransducerChannel proxy that defines a group of TransducerChannels to be read together. Each TransducerChannel in the proxy would be represented as a separate Argument of the correct type based on the TEDS information. These Arguments would be grouped together into an ArgumentArray that represents the data from the proxy.

To facilitate the communication of ArgumentArrays across the IEEE 1451.X layer, a generic Encode/Decode mechanism is provided via a Codec interface. These methods convert ArgumentArrays to/from OctetArrays. In most cases, this standard is responsible for calling these methods and the IEEE 1451.X layer will only deal with the OctetArray forms. IEEE 1451.X only needs to transfer the OctetArray "payload" from the initiating to the receiving destinations. The encoding and decoding operations are provided as a library, and an IEEE 1451.X is free to substitute a different implementation if it is needed. See Clause 7 and Clause 8 for details.

In rare cases, IEEE 1451.X may need to deviate from the IEEE 1451.0 recommended OctetArray format. In this case, IEEE 1451.X would provide a different Encode/Decode library.

9.2.4 Parameter ownership assumptions

Unless otherwise specified, parameters and returned results are owned by the initiating object or caller. If the "called" object needs to hold onto a parameter, it must make a local copy. The initiating object has the responsibility to free any memory resources for parameters and returned result after the call returns.

9.3 IEEE1451Dot0

```
IDL: module IEEE1451Dot0 { };
```

All IEEE 1451.0 interfaces are inside the IDL module "IEEE1451Dot0".

Copyright © 2007 IEEE. All rights reserved.

9.3.1 IEEE1451Dot0::Args

IDL: `module Args { };`

Fundamental and special data types are contained within the Args module. The fundamental data types and data types used in other parts of the standard are defined in Clause 4.

Figure 22 illustrates the relationships between the data types and the Argument and ArgumentArray classes.

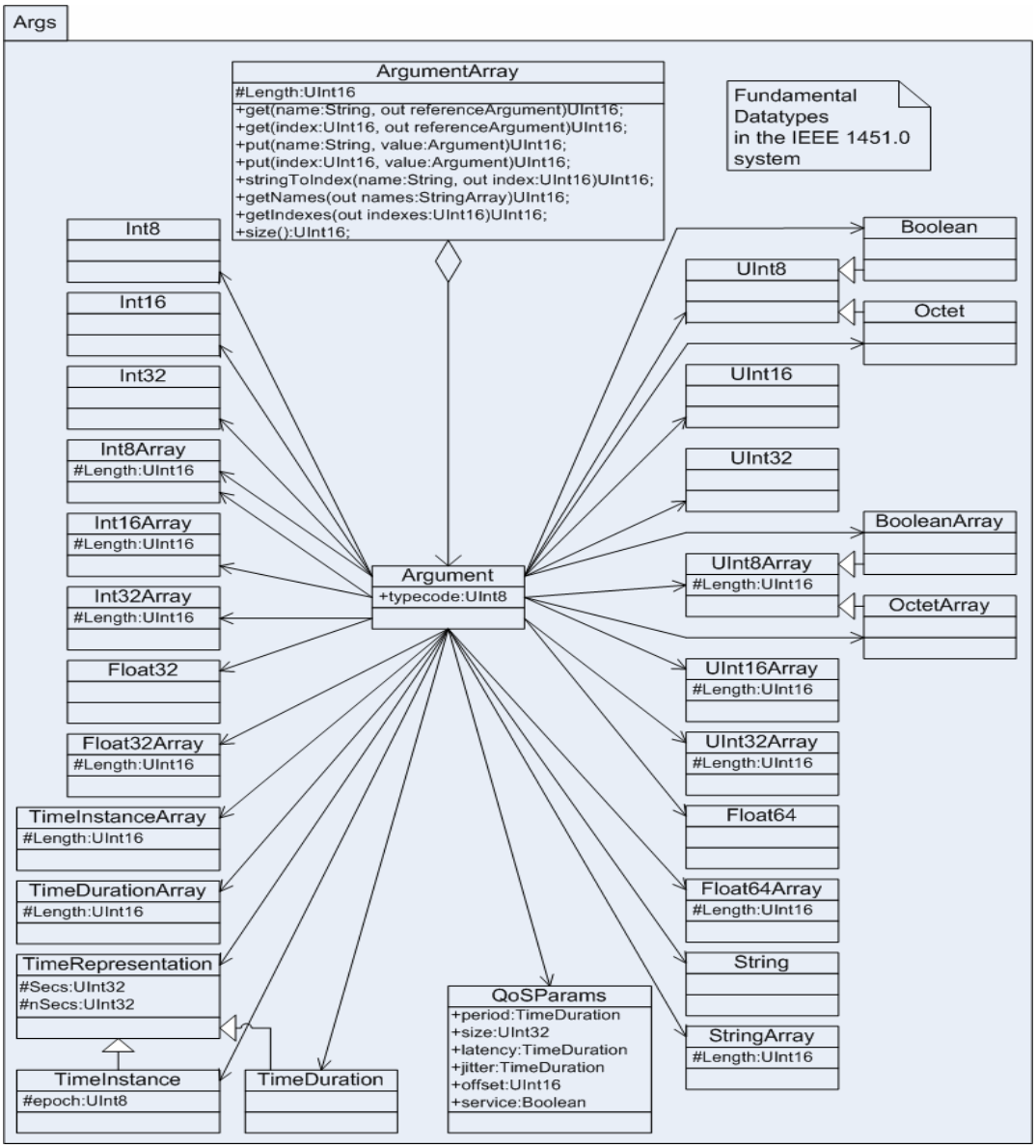


Figure 22—Arguments

Copyright © 2007 IEEE. All rights reserved.

9.3.1.1 Arrays of basic types

Variable-length arrays of basic types are provided as IDL sequences. Array data types contain both the length of the data and the data. The length is a UInt16 quanity.

```
IDL:   typedef sequence<Int8>     Int8Array;
typedef sequence<Int16>           Int16Array;
typedef sequence<Int32>           Int32Array;
typedef sequence<UInt8>           UInt8Array;
typedef sequence<UInt16>          UInt16Array;
typedef sequence<UInt32>          UInt32Array;
typedef sequence<Float32>         Float32Array;
typedef sequence<Float64>         Float64Array;
typedef sequence<_String>         StringArray;
typedef sequence<_Octet>          OctetArray;
typedef sequence<_Boolean>        BooleanArray;
typedef sequence<TimeInstance>    TimeInstanceArray;
typedef sequence<TimeDuration>    TimeDurationArray;
```

9.3.1.2 Error codes

All error codes are represented as UInt16 quantities. Five entities are involved in a communication transaction: the local IEEE 1451.0 layer, the local IEEE 1451.X layer, the remote IEEE 1451.X layer, the remote IEEE 1451.0 layer, and the remote application layer. The error code source is encoded in the upper 3 bits. The error code enumeration is encoded in the lower bits. Bits are numbered from most significant to least significant bit 15 to bit 0 as shown in Table 76.

Table 76—Bit assignments for error codes

Bits	Used by
Bits 15 through 13	(The 3 most significant bits): The error code source information is encoded in this location as described in Table 77
Bits 12 through 0	Error code enumeration; see Table 78

Error code source values are specified in Table 77. The value column contains the bits 15 through 13 evaluated as a 3 bit unsigned integer.

Table 77—Error code source enumeration

Value	Source of the error
0	Error from the local IEEE 1451.0 layer
1	Error from the local IEEE 1451.X layer
2	Error from the remote IEEE 1451.X layer
3	Error from the remote IEEE 1451.0 layer
4	Error from the remote application layer
5	Reserved
6	Reserved
7	Open to manufacturers

Error code values are specified in Table 78. In this case bits 12 through 0 are evaluated as a 13 bit unsigned integer.

Copyright © 2007 IEEE. All rights reserved.

Table 78—Error code enumerations

Enumerations	Error code name	Description
0	NO_ERROR	No error, operation successful
1	INVALID_COMMID	Invalid “commId”
2	UNKNOWN_DESTID	Unknown “destId”
3	TIMEOUT	Operation time-out
4	NETWORK_FAILURE	Destination unreachable network failure
5	NETWORK_CORRUPTION	Corrupt communication network failure
6	MEMORY	Local out-of-memory error
7	QOS_FAILURE	Network quality-of-service violation
8	MCAST_NOT_SUPPORTED	Multicast not supported or operation invalid for multicast
9	UNKNOWN_GROUPID	Unknown “groupId”
10	UNKNOWN_MODULEID	Unknown “moduleId”
11	UNKNOWN_MSGID	Unknown “msgId”
12	NOT_GROUP_MEMBER	destId not in the group
13	ILLEGAL_MODE	The mode parameter is not valid
14	LOCKED_RESOURCE	The resource being accessed is locked
15	FATAL_TEDS_ERROR	An error in the TEDS makes the device unusable
16	NON-FATAL_TEDS_ERROR	The value in a field in the TEDS is unusable, but the device will still function
17	CLOSE_ON_LOCKED_RESOURCE	A warning error code returned to signal that a close on a locked resource was performed
18	LOCK_BROKEN	If a non-blocking read or write, or measurement stream, is in progress, the callback will be invoked with this error code
19	NETWORK_RESOURCE_EXCEEDED	IEEE 1451.X has reached network resource limits
20	MEMORY_RESOURCE_EXCEEDED	IEEE 1451.X has reached memory resource limits
21–4095	Reserved	
4096–8191	Open to manufacturers	

9.3.1.3 IEEE1451Dot0::Args::QosParam

“Quality of Service” attributes are organized into a data structure for efficiency. The QoSParams structure contains the information listed in Table 79.

```
IDL: struct QoSParam {
   _Boolean        service;
   TimeDuration    period;
   UInt16          transmitSize;
   UInt16          replySize;
   TimeDuration    accessLatency;
   TimeDuration    transmitLatency;
}
```


Copyright © 2007 IEEE. All rights reserved.

Table 79—QoSParams descriptions

Parameter	Type	Description
Service	_Boolean	A "true" indicates guaranteed QoS. A "false" indicates "best effort."
Period	TimeDuration	Indicates the period of communication. A zero value should be used for nonperiodic conditions.
transmitSize	UInt32	This parameter specifies the number of octets transmitted for each periodic communication.
replySize	UInt32	For two-way communications, this parameter specifies the number of octets sent as a reply in each periodic communication. A value of 0 indicates a one-way communication.
accessLatency	TimeDuration	This parameter specifies the extra time latency that the IEEE 1451.0 layer can tolerate for the IEEE 1451.X layer to start transmission of a communication before IEEE 1451.X should report a communication exception. A zero value shall be interpreted as access latency not specified.
transmitLatency	TimeDuration	This parameter specifies the extra time latency that the IEEE 1451.0 layer can tolerate for the IEEE 1451.X layer to complete transmission of a communication before IEEE 1451.X should report a communication exception. A zero value shall be interpreted as transmit latency not specified.

9.3.1.4 IEEE1451Dot0::Args::TypeCode

Each valid type in the IEEE 1451.0 ArgumentArray has a unique typecode.

```
IDL: enum TypeCode {
   UNKNOWN_TC,

   // Simple types
   UInt8_TC, UINT16_TC, UINT32_TC,
   FLOAT32_TC, FLOAT64TC, STRING_TC,
   OCTET_TC, BOOLEAN_TC,
   TIME_INSTANCE_TC, TIME_DURATION_TC,
   QOS_PARAMS_TC,

   // Arrays of simple types. Note no QOS array
   UInt8_ARRAY_TC, UINT16_ARRAY_TC, UINT32_ARRAY_TC,
   FLOAT32_ARRAY_TC, FLOAT64_ARRAY_TC, STRING_ARRAY_TC,
   OCTET_ARRAY_TC, BOOLEAN_ARRAY_TC,
TIME_INSTANCE_ARRAY_TC, TIME_DURATION_ARRAY_TC
};
```

9.3.1.5 IEEE1451Dot0::Args::Argument

This is a generic data container. It is represented as an IDL discriminated union. However, implementations in language with run-time type checking may choose a simpler representation.

```
IDL: union Argument switch (TypeCode) {
case UNKNOWN_TC:            Boolean        valueError;
case UInt8_TC:              UInt8          valueInt8;
case UINT16_TC:             UInt16         valueUInt16;
case UINT32_TC:             UInt32         valueUInt32;
case FLOAT32_TC:            Float32        valueFloat32;
case FLOAT64_TC:            Float64        valueFloat64;
case STRING_TC:             _String        valueString;
case OCTET_TC:              _Octet         valueOctet;
case BOOLEAN_TC:            _Boolean       valueBoolean;
case TIME_INSTANCE_TC:      TimeInstance   valueTimeInstance;
case TIME_DURATION_TC:      TimeDuration   valueTimeDuration;
case QOS_PARAMS_TC:         QosParams      valueQosParams;
case UInt8_ARRAY_TC:        UInt8Array     valueInt8Array;
```


Copyright © 2007 IEEE. All rights reserved.

```
case UINT16_ARRAY_TC:         UInt16Array    valueUInt16Array;
case UINT32_ARRAY_TC:         UInt32Array    valueUInt32Array;
case FLOAT32_ARRAY_TC:        Float32Array   valueFloat32Array;
case FLOAT64_ARRAY_TC:        Float64Array   valueFloat64Array;
case STRING_ARRAY_TC:         StringArray    valueStringArray;
case OCTET_ARRAY_TC:          OctetArray     valueOctetArray;
case BOOLEAN_ARRAY_TC:        BooleanArray   valueBooleanArray;
case TIME_INSTANCE_ARRAY_TC: TimeInstanceArray
                                             valueTimeInstanceArray;
case TIME_DURATION_ARRAY_TC: TimeDurationArray
                                             valueTimeDurationArray;
};
```

9.3.1.6 IEEE1451Dot0::Args::ArgumentArray

IDL: `interface ArgumentArray { };`

This is a generic array data container. All Arguments are owned by the ArgumentArray. When the ArgumentArray is deleted, it will free all memory consumed by the Arguments. Table 80 provides a list of methods associated with ArgumentArrays.

Table 80—ArgumentArray

IEEE1451dot0::Args::ArgumentArray
UInt16 getByName(in _String name, out Argument reference);
UInt16 getByIndex(in UInt16 index, out Argument reference);
UInt16 putByName(in _String name, in Argument value);
UInt16 putByIndex(in UInt16 index, in Argument value);
UInt16 stringToIndex(in String name, out UInt16 index);
UInt16 getNames(out StringArray names);
UInt16 getIndexes(out UInt16Array indexes);
UInt16 size();

9.3.1.7 IEEE1451Dot0::Args::ArgumentArray::get

IDL: `UInt16 getByName ( in _String name, out Argument reference);`

This method provides a lookup-by-name feature. See Clause 7 and Clause 8 for appropriate attribute names. A reference to the desired Argument is returned. Note that the caller should consider this a "read-only" reference and must make a local copy if needed.

Parameters:
The "name" parameter is the name for the desired attribute.

The [out] "reference" parameter is a reference to the desired Argument.

Return result: Error code

9.3.1.8 IEEE1451Dot0::Args::ArgumentArray::get

IDL: `UInt16 getByIndex( in UInt16 index, out Argument reference);`

Copyright © 2007 IEEE. All rights reserved.

This method provides a lookup-by-index feature. See Clause 7 and Clause 8 for appropriate conversions between names and indices. A reference to the desired Argument is returned. Note that the caller should consider this a "read-only" reference and must make a local copy if needed.

Parameters:
The "index" parameter is the array index beginning with zero.

The [out] "reference" parameter is a reference to the desired Argument.

Return result: Error code

9.3.1.9 IEEE1451Dot0::Args::ArgumentArray::put

IDL: `UInt16 putByName( in String name, in Argument value);`

This method provides a set-by-name feature. See Clause 7 and Clause 8 for appropriate conversions between names and indices. The ArgumentArray will assume ownership of the provided Argument. The caller shall be careful not to free the memory associated with that Argument. If an Argument is already in the ArgumentArray with the same name, it will be deleted.

Parameters:
The "name" parameter is the name for the desired attribute.

The "value" Argument parameter will be stored into the ArgumentArray.

Return result: Error code

9.3.1.10 IEEE1451Dot0::Args::ArgumentArray::put

IDL: `UInt16 putByIndex( in UInt16 index, in Argument value);`

This method provides a set-by-index feature. See Clause 7 and Clause 8 for appropriate conversions between names and indices. The ArgumentArray will assume ownership of the provided Argument. The caller shall be careful not to free the memory associated with that Argument. If an Argument is already in the ArgumentArray at the same index, it will be deleted.

Parameters:
The "index" parameter is the array index beginning with 0.

The "value" Argument parameter will be stored into the ArgumentArray.

Return result: Error code

9.3.1.11 IEEE1451Dot0::Args::ArgumentArray::stringToIndex

IDL: `UInt16 stringToIndex( in _String name, out UInt16 index);`

This method provides a conversion mechanism from names to array index. It is usually more efficient to perform a lookup-by-index rather than a lookup-by-name. This method allows the application to perform the string search and compare operation once and then use the index for future array access.

Parameters:
The "name" parameter is the desired name.

The [out] "index" parameter is the array index beginning with zero.

Copyright © 2007 IEEE. All rights reserved.

Return result: Error code

9.3.1.12 IEEE1451Dot0::Args::ArgumentArray::getNames

IDL: `UInt16 getNames( out StringArray names);`

This method returns a StringArray of "names" for each element in the ArgumentArray. Each could be used in the get( ) method.

Parameters:
The [out] "names" parameter is the StringArray.

Return result: Error code

9.3.1.13 IEEE1451Dot0::Args::ArgumentArray::getIndexes

IDL: `UInt16 getIndexes( out UInt16Array indexes);`

This method returns a UInt16Array of "indexes" for each element in the ArgumentArray. Each could be used in the get( ) method.

Parameters:
The [out] "indexes" parameter is the UInt16Array.

Return result: Error code

9.3.1.14 IEEE1451Dot0::Args::ArgumentArray::size

IDL: `UInt16 size();`

This method returns the number of elements in the ArgumentArray.

Return result: Number of elements.

9.3.2 IEEE1451Dot0::Util

IDL: `module Util { };`

Utility classes and interfaces are organized into this module.

9.3.2.1 IEEE1451Dot0::Util::Codec

IDL: `interface Codec { };`

This interface is optionally provided by the IEEE 1451.X layer to provide custom Encoding and Decoding of ArgumentArrays to/from OctetArrays. If registered, it will be invoked by the IEEE 1451.0 layer automatically. Table 81 lists the interfaces in this grouping.

Copyright © 2007 IEEE. All rights reserved.

Table 81—Codec

IEEE1451dot0::Util::Codec
Args::UInt16 encodeCommand(in Args::UInt16 channelId, in Args::UInt8 cmdClassId, in Args::UInt8 cmdFunctionId, in Args::ArgumentArray inArgs, out Args::OctetArray payload);
Args::UInt16 decodeCommand(in Args::OctetArray payload, out Args::UInt16 channelId, out Args::UInt8 cmdClassId, out Args::UInt8 cmdFunctionId, out Args::ArgumentArray inArgs);
Args::UInt16 encodeResponse(in Args::_Boolean successFlag, in Args::ArgumentArray outArgs, out Args::OctetArray payload);
Args::UInt16 decodeResponse(in Args::OctetArray payload, out Args::_Boolean successFlag, out Args::ArgumentArray outArgs);
Args::UInt16 argumentArray2OctetArray(in Args::ArgumentArray inArgs, out Args::OctetArray payload);
Args::UInt16 octetArray2ArgumentArray(in Args::OctetArray payload, out Args::ArgumentArray outArgs);

9.3.2.2 IEEE1451Dot0::Util::Codec::encodeCommand

```
IDL: Args::UInt16 encodeCommand(
in  Args::UInt16        destId,
in  Args::UInt16         channelId,
in  Args::UInt8          cmdClassId,
in  Args::UInt8          cmdFunctionId,
in  Args::ArgumentArray  inArgs,
out Args::OctetArray     payload);
```

This method is provided by the IEEE 1451.X layer and is called by the IEEE 1451.0 layer to encode a command into an OctetArray. See Clause 6 for details on the encoding conventions. This is called on the initiating node before invocation of the ModuleCommunication::P2PComm::write( ) or ModuleCommunication::NetComm::writeMsg( ) calls.

Propose that dot 0 read & use the Commands TEDS to encode & decode unknown commands.

Parameters:
The “channelId” parameter is the desired channel identifier.

The “cmdClassId” is the desired command class.

The “cmdFunctionId” is the desired command function code.

The “inArgs” parameter contains the command specific input arguments.

The [out] “payload” is the encoded OctetArray.

Return result: Error code

9.3.2.3 IEEE1451Dot0::Util::Codec::decodeCommand

```
IDL: Args::UInt16 decodeCommand(
in  Args::OctetArray    payload,
out Args::UInt16        channelId,
out Args::UInt8         cmdClassId,
out Args::UInt8         cmdFunctionId,
out Args::ArgumentArray inArgs);
```

This method is provided by the IEEE 1451.X layer and is called by the IEEE 1451.0 layer to decode an OctetArray into command arguments. See Clause 6 for details on the encoding conventions. This is called on the receiving node after invocation of the ModuleCommunication::P2PComm::read( ) or ModuleCommunication::NetComm::readMsg( ) calls.

Parameters:

Copyright © 2007 IEEE. All rights reserved.

The “payload” is the OctetArray.

The [out] “channelId” parameter is the desired channel identifier.

The [out] “cmdClassId” is the desired command class.

The [out] “cmdFunctionId” is the desired command function code.

The [out] “inArgs” parameter contains the command specific input arguments.

Return result: Error code

9.3.2.4 IEEE1451Dot0::Util::Codec::encodeResponse

```
IDL: Args::UInt16 encodeResponse(
in  Args::_Boolean         successFlag,
in  Args::ArgumentArray    outArgs,
out Args::OctetArray       payload);
```

This method is provided by the IEEE 1451.X layer and called by the IEEE 1451.0 layer to encode a response into an OctetArray. See Clause 7 and Clause 8 for details on the encoding conventions. This is called on the receiving node before invocation of the ModuleCommunication::P2PComm::write( ) or ModuleCommunication::NetComm::writeRsp( ) calls.

Parameters:
The “successFlag” parameter is the desired success code.

The “outArgs” parameters are the command response specific output arguments.

The [out] “payload” is the encoded OctetArray.

Return result: Error code

9.3.2.5 IEEE1451Dot0::Util::Codec::decodeResponse

```
IDL: Args::UInt16 decodeResponse(
in  Args::OctetArray      payload,
out Args::_Boolean        successFlag,
out Args::ArgumentArray   outArgs);
```

This method is provided by the IEEE 1451.X layer and is called by the IEEE 1451.0 layer to decode an OctetArray into response arguments. See Clause 7 and Clause 8 for details on the encoding conventions. This is called on the initiating node after invocation of the ModuleCommunication::P2PComm::read( ) or ModuleCommunication::NetComm::readMsg( ) calls.

Parameters:
The “payload” is the OctetArray.

The [out] “successFlag” parameter is the command success flag.

The [out] “outArgs” parameter are the command specific output arguments.

Return result: Error code

9.3.2.6 IEEE1451Dot0::Util::Codec::encodeArgumentArray

```
IDL: Args::UInt16 encodeArgumentArray(
```


Copyright © 2007 IEEE. All rights reserved.

```
in  Args::ArgumentArray   inArgs,
out Args::OctetArray      payload);
```

This method is provided by the IEEE 1451.X layer and is called by the IEEE 1451.0 layer to encode an ArgumentArray into an OctetArray. See Clause 7 and Clause 8 for details on the encoding conventions. Make sure that this works!!

Parameters:
The "inArgs" parameter is the input ArgumentArray.

The [out] "payload" is the encoded OctetArray.

Return result: Error code

9.3.2.7 IEEE1451Dot0::Util::Codec::decodeOctetArray

```
IDL: Args::UInt16 decodeOctetArray(
in  Args::OctetArray      payload,
out Args::ArgumentArray   outArgs);
```

This method is provided by the IEEE 1451.X layer and is called by the IEEE 1451.0 layer to decode an OctetArray into an ArgumentArray. See Clause 7 and Clause 8 for details on the encoding conventions.

Parameters:
The "payload" is the OctetArray.

The [out] "outArgs" parameter contains the output ArgumentArray.

Return result: Error code

9.3.2.8 IEEE1451Dot0::Util::CodecManagement

```
IDL: Interface CodecManagement { };
```

This class is provided by the IEEE 1451.0 layer and is optionally called by the IEEE 1451.X layer to register alternate Encode/Decode implementations with the IEEE 1451.0 layer. Table 82 lists the methods available in this class.

Table 82—CodecManagement

IEEE1451dot0::Util::CodecManagement
Args::UInt16 register(in Args::UInt8 moduleId, in Codec customCodec);
Args::UInt16 unregister(in Args::UInt8 moduleId);

9.3.2.9 IEEE1451Dot0::Util::CodecManagement::register

```
IDL: Args::UInt16 register(
in Args::UInt8    moduleId,
in Codec          customCodec);
```

This method is provided by the IEEE 1451.0 layer and is optionally called by the IEEE 1451.X layer to register a custom Encoder/Decoder instance. Only a single instance may be registered for each module.

Parameters:
The "moduleIds" parameter is the desired IEEE 1451.X ID.

Copyright © 2007 IEEE. All rights reserved.

The "customCodec" parameter is the Codec instance to register. This instance should not be destructed until after the unregister( ) call is made.

Return result: Error code

9.3.2.10 IEEE1451Dot0::Util::CodecManagement::unregister

```
IDL: Args::UInt16 unregister( in Args::UInt8 moduleId);
```

This method is provided by the IEEE 1451.0 layer and is optionally called by the IEEE 1451.X layer to unregister a custom Encoder/Decoder instance. After this call, the default Encoder/Decoder will be used.

Parameters:
The "moduleIds" parameter is the desired IEEE 1451.X ID.

Return result: Error code

10. Transducer services API

```
IDL: module TransducerServices { };
```

The Transducer services API provides the interface between the applications running on the NCAP and the functions defined by this standard.

All interfaces primarily used by "measurement and control" applications are inside this IDL module.

The "TransducerServices" module is subdivided into five interfaces as listed in Table 83. The first four interfaces are implemented by this standard and are called by the measurement application. If the application desires advanced optional features, it will need to implement the "AppCallback" interface, which this standard will invoke.

Table 83—Transducer Services API classes and interfaces

Interface	Description
TimDiscovery	Methods for applications to discover available IEEE 1451.X communications modules, TIMs, and TransducerChannels are organized in this interface.
TransducerAccess	When an application desires to access sensor and actuator TransducerChannels, it will use methods on this interface.
TransducerManager	Applications that need more control over TIM access will use methods on this interface. For example, to lock the TIM for exclusive use and to send arbitrary commands to the TIM.
TedsManager	Applications use methods on this interface to read and write TEDS. This class also manages the NCAP-side TEDS cache information.
CommManager	Handles access to the communication module on the local device.
AppCallback	Applications that need advanced features need to implement this interface. For example, this allows the application to configure measurement streams and the IEEE 1451.0 layer will invoke appropriate callbacks in the application.

10.1 IEEE1451Dot0::TransducerServices::TimDiscovery

```
IDL: interface TimDiscovery { };
```


Copyright © 2007 IEEE. All rights reserved.

The TimDiscovery interface is provided by the IEEE 1451.0 layer and is called by the application to provide a common mechanism to discover available TIMs and TransducerChannels. The methods are listed in Table 84 and discussed in 10.1.1 through 10.1.3.

Table 84—TimDiscovery

IEEE1451dot0::TransducerServices::TimDiscovery
Args::UInt16 reportCommModule(out Args::UInt8Array moduleIds);
Args::UInt16 reportTims(in Args::UInt8 moduleId, out Args::UInt16Array timIds);
Args::UInt16 reportChannels(in Args::UInt16 timId, out Args::UInt16Array channelIds, out Args::StringArray names);

10.1.1 IEEE1451Dot0::TransducerServices::TimDiscovery::reportCommModule

```
IDL: Args::UInt16 reportCommModule( out Args::UInt8Array moduleIds);
```

This method reports the available communication module interfaces that have been registered with this standard. See the IEEE1451Dot0::ModuleCommunication::NetRegistration::registerModule( ) method, which the IEEE 1451.X layer will invoke when it is ready for operation. In that method, the IEEE 1451.0 layer will assign a unique moduleId to each IEEE 1451.X interface. Note that the NCAP may have:

A single IEEE 1451.X interface of a given technology (for example, Clause 7 of IEEE Std 1451.5-2007 [B4]) .

Multiple interfaces of the same technology (for example, IEEE 1451.2-RS232 on COM1 and COM2).

Multiple IEEE 1451.X interfaces of different technologies (for example, Clause 7 of IEEE Std 1451.5-2007 [B4] and IEEE 1451.3 multidrop).

Parameters:
The [out] “moduleIds” parameter is returned by the IEEE 1451.0 layer to the application. This array contains all the known communication modules on this NCAP.

Return result: Error code

10.1.2 IEEE1451Dot0::TransducerServices::TimDiscovery::reportTims

```
IDL: Args::UInt16 reportTims(
in Args::UInt8          moduleId,
out Args::UInt16Array   timIds);
```

This returns the known TIM devices on this interface. See 11.6.2, the IEEE1451Dot0::ModuleCommunication::Registration::registerDestination method, which the IEEE 1451.X layer will invoke when registering new TIMs to the NCAP.

Parameters:
The “moduleId” parameter is the desired IEEE 1451.X communication module ID.

The [out] “timIds” parameter is returned to the application and contains all known TIMs on this IEEE 1451.X module.

Return result: Error code

Copyright © 2007 IEEE. All rights reserved.

10.1.3 IEEE1451Dot0::TransducerServices::TimDiscovery::reportChannels

```
IDL: Args::UInt16 reportChannels(
in  Args::UInt16         timId,
out Args::UInt16Array    channelIds,
out Args::StringArray    names);
```

This returns the TransducerChannel list and names for this TIM. This information is retrieved from the cached TEDS.

Parameters:
The “timId” parameter is the desired TIM.

The [out] “channelIds” parameter is returned to the application and contains all known TransducerChannels on this TIM.

The [out] “names” parameter is returned to the application and contains the TransducerChannel names.

Return result: Error code

10.2 IEEE1451Dot0::TransducerServices::TransducerAccess

```
IDL: interface TransducerAccess { };
```

The TransducerAccess interface is provided by the IEEE 1451.0 layer and is called by the application to provide access to TransducerChannels. For most applications, they will primarily be interacting with this interface to perform TIM read and write operations. To keep this interface small, more advanced methods are placed in the TransducerManager interface. Each method is listed in Table 85.

Table 85—TransducerAccess methods

IEEE1451dot0::TransducerServices::TransducerAccess
Args::UInt16 open(in Args::UInt16 timId, in Args::UInt16 channelId, out Args::UInt16 transCommId);
Args::UInt16 openQoS(in Args::UInt16 timId, in Args::UInt16 channelId, inout Args::QoSParams qosParams, out Args::UInt16 transCommId);
Args::UInt16 openGroup(in Args::UInt16Array timIds, in Args::UInt16Array channelIds, out Args::UInt16 transCommId);
Args::UInt16 openGroupQoS(in Args::UInt16Array timIds, in Args::UInt16Array channelIds, inout Args::QoSParams qosParams, out Args::UInt16 transCommId);
Args::UInt16 close(in Args::UInt16 transCommId);
Args::UInt16 readData (in Args::UInt16 transCommId, in Args::TimeDuration time-out, in Args::UInt8 SamplingMode, out Args::ArgumentArray result);
Args::UInt16 writeData (in Args::UInt16 transCommId, in Args::TimeDuration time-out, in Args::UInt8 SamplingMode, in Args::ArgumentArray value);
Args::UInt16 startReadData(in Args::UInt16 transCommId, in Args::TimeInstance triggerTime, in Args::TimeDuration time-out, in Args::UInt8 SamplingMode, in AppCallback callback, out Args::UInt16 operationId);
Args::UInt16 startWriteData(in Args::UInt16 transCommId, in Args::TimeInstance triggerTime, in Args::TimeDuration time-out, in Args::UInt8 SamplingMode, in Args::ArgumentArray value, in AppCallback callback, out Args::UInt16 operationId);
Args::UInt16 startStream(in Args::UInt16 transCommId, in AppCallback callback, out Args::UInt16 operationId);
Args::UInt16 cancel(in Args::UInt16 operationId);

10.2.1 IEEE1451Dot0::TransducerServices::TransducerAccess::open

```
IDL: Args::UInt16 open(
```


Copyright © 2007 IEEE. All rights reserved.

```
in  Args::UInt16      timId,
in  Args::UInt16      channelId,
out Args::UInt16      transCommId);
```

This method opens a communication channel to the desired TIM/TransducerChannel and returns a "transCommId" that will be used in subsequent calls. The default Quality of Service is used.

Parameters:
The "timId" specifies the desired TIM

The "channelId" specifies the desired TransducerChannel. This field allows addressing a single TransducerChannel, a TransducerChannel proxy, a group of TransducerChannels, or all TransducerChannels connected to an NCAP. To address the TIM, use a TransducerChannel of zero. See 5.3 for details.

The [out] "transCommId" parameter is returned by the IEEE 1451.0 layer to the application. This identifier will be used on subsequent calls.

Return result: Error code

10.2.2 IEEE1451Dot0::TransducerServices::TransducerAccess::openQoS

```
IDL: Args::UInt16 openQoS(
in    Args::UInt16      timId,
in    Args::UInt16      channelId,
inout Args::QoSParams   qosParams,
out   Args::UInt16      transCommId);
```

This method opens a communication channel to the desired TIM/TransducerChannel and returns a "transCommId" that will be used in subsequent calls. Special Quality of Service communications are used. If the call fails, the "qosParams" will be modified and returned to the application in order to provide a "hint" on what QoS may be acceptable.

Parameters:
The "timId" specifies the desired TIM.

The "channelId" specifies the desired TransducerChannel. This field allows addressing a single TransducerChannel, a TransducerChannel proxy, a group of TransducerChannels or all TransducerChannels connected to an NCAP. To address the TIM use a TransducerChannel of zero. See 5.3 for details.

The inout "qosParams" is the desired quality of service parameters. See 9.3.1.3 for details.

The [out] "transCommId" parameter is returned by the IEEE 1451.0 layer to the application. This identifier will be used on subsequent calls.

Return result: Error code

10.2.3 IEEE1451Dot0::TransducerServices::TransducerAccess::openGroup

The purpose of this method is to assign a number of TIMs to multicast addresses that will later be used to assign TransducerChannels to AddressGroups. See 5.3.2 for a description of AddressGroups.

```
IDL: Args::UInt16 openGroup(
in  Args::UInt16Array    timIds,
in  Args::UInt16Array    channelIds,
out Args::UInt16         transCommId);
```


Copyright © 2007 IEEE. All rights reserved.

This method opens a group communication channel to the desired set of TIMs/TransducerChannels and returns a "transCommId" that will be used in subsequent calls. The default Quality of Service is used.

There is a one-to-one correspondence between the positions for the timIds and channelIds arrays. The TransducerChannels may be on the same or different TIMs. All TIMs shall be attached to the same communication module. If there are multiple channelIds for a given timId, the timId shall be repeated for each channelId within that TIM so that the two lists are of equal length.

Parameters:
The "timIds" specifies the desired TIMs.

The "channelIds" specifies the desired TransducerChannels. This field allows addressing a single TransducerChannel, a TransducerChannel proxy, a group of TransducerChannels, or all TransducerChannels connected to an NCAP. See 5.3 for details.

The [out] "transCommId" parameter is returned by the IEEE 1451.0 layer to the application. This identifier will be used on subsequent calls.

Return result: Error code

10.2.4 IEEE1451Dot0::TransducerServices::TransducerAccess::openGroupQoS

The purpose of this method is to assign a number of TIMs to multicast addresses that will later be used to assign TransducerChannels to AddressGroups. See 5.3.2 for a description of AddressGroups.

```
IDL: Args::UInt16 openGroupQoS(
in    Args::UInt16Array   timIds,
in    Args::UInt16Array   channelIds,
inout Args::QoSParams     qosParams,
out   Args::UInt16        transCommId);
```

This method opens a group communication channel to the desired TIMs/TransducerChannels and returns a "transCommId" that will be used in subsequent calls. Special Quality of Service communications are used. If the call fails, the "qosParams" will be modified and returned to the application in order to provide a "hint" on what QoS may be acceptable.

There is a one-to-one correspondence between the positions for the timIds and channelIds arrays. The TransducerChannels may be on the same or different TIMs. All TIMs must be attached to the same communication module.

Parameters:
The "timIds" specifies the desired TIMs

The "channelIds" specifies the desired TransducerChannels. This field allows addressing a single TransducerChannel, a TransducerChannel proxy, a group of TransducerChannels, or all TransducerChannels connected to an NCAP. See 5.3 for details.

The inout "qosParams" is the desired quality of service parameters. See 9.3.1.3 for details.

The [out] "transCommId" parameter is returned by the IEEE 1451.0 layer to the application. This identifier will be used on subsequent calls.

Return result: Error code

10.2.5 IEEE1451Dot0::TransducerServices::TransducerAccess::close

```
IDL: Args::UInt16 close( in Args::UInt16 transCommId);
```


Copyright © 2007 IEEE. All rights reserved.

This method closes a transducer communication session. The application shall consider the transCommId as invalid. Note that a subsequent "open" call may return the old value.

See TransducerManager::unlock( ) for information on calling close( ) on a locked transCommId.

Parameters:
The "transCommId" parameter indicates which communication to close.

Return result: Error code

10.2.6 IEEE1451Dot0::TransducerServices::TransducerAccess::readData

```
IDL: Args::UInt16 readData(
in  Args::UInt16         transCommId,
in  Args::TimeDuration   timeout,
in  Args::UInt8          SamplingMode,
out Args::ArgumentArray  result);
```

This method performs a blocking read of the specified TransducerChannel(s). The ArgumentArray can have many attributes as discussed in Clause 7 and Clause 8; each attribute is represented by a separate "Argument" in the ArgumentArray. The application can control which attributes are returned through the use of the TransducerManager::configureAttributes( ) call.

In cases where this is a read of a single TransducerChannel, there will always be a "result" Argument that contains the TransducerChannel reading. The type of this Argument is determined by the TransducerChannel's data model and if the TransducerChannel has a Calibration TEDS. For example, a read of a simple TransducerChannel that does not specify Calibration TEDS will always be in the TransducerChannel's native format (for example, UInt8 or Float32Array). If the TransducerChannel does specify NCAP-side correction via the Calibration TEDS, the data type will always be Float32 or Float32Array.

In cases where this is a read of a group of TransducerChannels, there will always be a nested "result" ArgumentArray that contains an Argument for each TransducerChannel in the group. These will be accessed in numerical order beginning with array position "0". This organization corresponds to the order of TIM/TransducerChannel pairs in the openGroup( ) or openGroupQoS( ) call. The data type for each returned Argument will be like the single TransducerChannel read discussed in the previous paragraph. .

Parameters:
The "transCommId" parameter indicates which transducer communication session to use.

The "timeout" parameter specifies how long to wait to perform the reading without generating a time-out error. Note a time-out can occur due to communication or trigger failures.

The "SamplingMode" specifies the triggering mechanism. See 5.11 and 7.1.2.4 for details.

The [out] "result" ArgumentArray is the returned values.

Return result: Error code

10.2.7 IEEE1451Dot0::TransducerServices::TransducerAccess::writeData

```
IDL: Args::UInt16 writeData(
in Args::UInt16          transCommId,
in Args::TimeDuration    timeout,
in Args::UInt8           SamplingMode,
in Args::ArgumentArray   value);
```


Copyright © 2007 IEEE. All rights reserved.

This method performs a blocking write of the specified TransducerChannels. The ArgumentArray will have many attributes as discussed in Clause 7; each attribute will be represented by a separate "Argument" in the ArguemntArray.

In cases where this is a write of a single TransducerChannel, the caller shall provide a "value" Argument that contains the TransducerChannel's value. The caller should provide the result in a compatible data type to what the TransducerChannel requires as specified in the TransducerChannel TEDS. The IEEE 1451.0 layer on the NCAP will perform simple conversions among all numeric data types. Note that this may lose precision if the resulting data type is smaller. For example, if the actuator required a UInt8, a provided Float32 would be downconverted to a UInt8 with appropriate loss of precision before being passed to the TransducerChannel. In cases where the NCAP will be performing correction (as specified in the Calibration TEDSfor this TransducerChannel), the value's data type shall be numeric. It will be converted to a Float32 or Float32Array type before passing through the correction engine. The output of the correction engine will be converted to the form required by the actuators Data Model as defined in the TransducerChannel TEDS.

In cases where this is a write to a group of TransducerChannels, there is always a nested "value" ArgumentArray that contains an Argument for each TransducerChannel in the group. These will be accessed in numerical order beginning with array position "0." This organization corresponds to the order of TIM/TransducerChannel pairs in the openGroup( ) or openGroupQoS( ) call. The data type for each Argument shall follow the rules for the single TransducerChannel write case discussed in the previous paragraph

Parameters:
The "transCommId" parameter indicates which transducer communication session to use.

The "timeout" parameter specifies how long to wait to perform the reading without generating a time-out error. Note a time-out can occur due to communication or trigger failures.

The "SamplingMode" specifies the triggering mechanism. See 5.11 and 7.1.2.4 for details.

The "value" ArgumentArray is the provided actuator input values.

Return result: Error code

10.2.8 IEEE1451Dot0::TransducerServices::TransducerAccess::startReadData

```
IDL: Args::UInt16 startReadData(
in  Args::UInt16         transCommId,
in  Args::TimeInstance   triggerTime,
in  Args::TimeDuration   timeout,
in  Args::UInt8          SamplingMode,
in  AppCallback          callback,
out Args::UInt16         operationId);
```

This method begins a non-blocking read of the specified TransducerChannels. When the read completes, the AppCallback::measurementUpdate( ) callback will be invoked on the callback object.

Parameters:
The "transCommId" parameter indicates which transducer communication session to use.

The "triggerTime" parameter specifies when to begin the read operation. A value specified in the past will result in an immediate time-out failure. A value of secs == 0, nsecs == 0 is a special case that implies read immediately.

The "timeout" parameter specifies how long to wait after initiating the read operation without generating a time-out error. Note a time-out can occur due to communication or trigger failures.

Copyright © 2007 IEEE. All rights reserved.

The “SamplingMode” specifies the triggering mechanism. See 5.11 and 7.1.2.4 for details.

The “callback” parameter is the interface to invoke when the read has completed. It will also be invoked upon failures.

The [out] “operationId” parameter is an identifier that can be used to cancel the read request.

Return result: Error code

10.2.9 IEEE1451Dot0::TransducerServices::TransducerAccess::startWriteData

```
IDL: Args::UInt16 startWriteData(
in  Args::UInt16            transCommId,
in  Args::TimeInstance      triggerTime,
in  Args::TimeDuration      timeout,
in  Args::UInt8             SamplingMode,
in  Args::ArgumentArray     value,
in  AppCallback             callback,
out Args::UInt16            operationId);
```

This method begins a non-blocking write of the specified TransducerChannels. When the write completes, the AppCallback::actuationComplete( ) callback will be invoked on the callback object.

Parameters:
The “transCommId” parameter indicates which transducer communication session to use.

The “triggerTime” parameter specifies when to begin the write operation. A value specified in the past will result in an immediate time-out failure. A value of secs == 0, nsecs == 0 is a special case that implies write immediately.

The “timeout” parameter specifies how long to wait after initiating the write operation without generating a time-out error. Note a time-out can occur due to communication or trigger failures.

The “SamplingMode” specifies the triggering mechanism. See 5.11 and 7.1.2.4 for details.

The “value” ArgumentArray is the provided actuator input values. See 10.2.7 write( ) for details.

The “callback” parameter is the interface to invoke when the write has completed. It will also be invoked upon failures.

The [out] “operationId” parameter is an identifier that may be used to cancel the write request.

Return result: Error code

10.2.10 IEEE1451Dot0::TransducerServices::TransducerAccess::startStream

```
IDL: Args::UInt16 startStream(
in  Args::UInt16      transCommId,
in  AppCallback       callback,
out Args::UInt16      operationId);
```

This method begins operation of a measurement stream. The transCommId shall be created with either the openQoS( ) or the openGroupQoS( ) call. In the later case, all TransducerChannels shall be from the same TIM. Each time new measurements are available from the stream, the AppCallback::measurementUpdate( ) callback will be invoked on the callback object.

Parameters:
The “transCommId” parameter indicates which transducer communication session to use.

Copyright © 2007 IEEE. All rights reserved.

The "callback" parameter is the interface to invoke when a data set has been written to an actuator or received from a sensor. It will also be invoked upon failures.

The [out] "operationId" parameter is an identifier that may be used to cancel the measurement stream.

Return result: Error code

10.2.11 IEEE1451Dot0::TransducerServices::TransducerAccess::cancel

IDL: `Args::UInt16 cancel( in Args::UInt16 operationId);`

This method will cancel a blocking read, blocking write, or measurement stream. The callback will be invoked with a CANCEL status error code.

Parameters:
The "operationId" parameter specifies the operation to cancel.

Return result: Error code

10.3 IEEE1451Dot0::TransducerServices::TransducerManager

IDL: `Interface TransducerManager { };`

The TransducerManager interface (see Table 86) is provided by this system and is called by the application to provide access to more advanced features. For most applications, they will not interact with this interface but will primarily be interacting with the TransducerAccess interface to perform TransducerChannel read and write operations. Advanced methods are placed in the TransducerManager interface to keep the TransducerAccess class small.

Table 86—TransducerManager interface methods

IEEE1451dot0::TransducerServices::TransducerManager
Args::UInt16 lock(in Args::UInt16 transCommId, in Args::TimeDuration time-out);
Args::UInt16 unlock(in Args::UInt16 transCommId);
Args::UInt16 reportLocks(out Args::UInt16Array transCommIds);
Args::UInt16 breakLock(in Args::UInt16 transCommId);
Args::UInt16 sendCommand(in Args::UInt16 transCommId, in Args::TimeDuration time-out, in Args::UInt8 cmdClassId, in Args::UInt8 cmdFunctionId, in Args::ArgumentArray inArgs, out Args::ArgumentArray outArgs);
Args::UInt16 startCommand(in Args::UInt16 transCommId, in Args::TimeInstance triggerTime, in Args::TimeDuration time-out, in Args::UInt8 cmdClassId, in Args::UInt8 cmdFunctionId, in Args::ArgumentArray inArgs, in AppCallback callback, out Args::UInt16 operationId);
Args::UInt16 configureAttributes(in Args::UInt16 transCommId, in Args::StringArray attributeNames);
Args::UInt16 trigger(in Args::UInt16 transCommId, in Args::TimeInstance triggerTime, in Args::TimeDuration time-out, in Args::UInt16 SamplingMode);
Args::UInt16 startTrigger(in Args::UInt16 transCommId, in Args::TimeInstance triggerTime, in Args::TimeDuration time-out, in Args::UInt16 SamplingMode, in AppCallbackcallback, out Args::UInt16 operationId);
Args::UInt16 clear(in Args::UInt16 transCommId, in Args::TimeDuration time-out, in Args::UInt8 clearMode);
Args::UInt16 registerStatusChange(in Args::UInt16 transCommId, in Args::TimeDuration time-out, in AppCallback callback, out Args::UInt16 operationId);
Args::UInt16 unregisterStatusChange(in Args::UInt16 transCommId);

Copyright © 2007 IEEE. All rights reserved.

10.3.1 IEEE1451Dot0::TransducerServices::TransducerManager::lock

```
IDL: Args::UInt16 lock(
in Args::UInt16          transCommId,
in Args::TimeDuration    timeout);
```

This method will lock the TIM/TransducerChannels represented by the transCommId. This will prevent other applications from accessing those resources. To prevent deadlocks in multi-threaded environments, it is the application's responsibility to lock resources in agreed upon order.

The implementation shall allow multiple locks by the same calling thread without blocking.

In cases where the transCommId specifies a group, all TIM/TransducerChannels in that group will be locked sequentially in the order specified in the openGroup( ) or openGroupQoS( ) call.

Parameters:
The "transCommId" specifies the desired transducer communication session.

The "timeout" specifies the duration to wait when acquiring the lock. A value of secs == 0, nsecs == 0 implies no wait and can be used to test for an existing lock. A value of secs == 0, nsecs == –1 implies wait forever. Using a value of "wait forever" is extremely dangerous as it can create deadlocks.

Return result: Error code

10.3.2 IEEE1451Dot0::TransducerServices::TransducerManager::unlock

```
IDL: Args::UInt16 unlock( in Args::UInt16 transCommId);
```

This method will unlock the TIM/TransducerChannels represented by the transCommId. This will allow other applications to access those resources.

In cases where the application has called lock( ) multiple times with the same calling thread, the application shall ensure that unlock( ) is called the same number of times. When the last unlock( ) is invoked, the resources are now available.

The implementation shall allow an alternative calling thread to invoke unlock( ). This alternative only is valid when lock( ) has been called a single time. An example would be a non-blocking operation. The initiating thread calls open( ), lock( ), and the non-blocking read( ). When the read completes, the AppCallback::measurementUpdate( ) callback will be invoked. That thread may then call unlock( ).

A call to close( ) will result in unlock( ) being called the correct number of times. A warning error code will be returned to signal that a close on a locked resource was performed.

Parameters:
The "transCommId" specifies the desired transducer communication session.

Return result: Error code

10.3.3 IEEE1451Dot0::TransducerServices::TransducerManager::reportLocks

.
```
IDL: Args::UInt16 unlock(out Args::UInt16Array transCommIds);
```

This method will report all transCommIds that are currently locked.

Copyright © 2007 IEEE. All rights reserved.

Parameters:
The [out] "transCommIds" returns an array of locked IDs.

Return result: Error code

10.3.4 IEEE1451Dot0::TransducerServices::TransducerManager::breakLock

```
IDL: Args::UInt16 breakLock( in Args::UInt16 transCommId);
```

This method will break a lock. If a non-blocking read or write or measurement stream is in progress, the callback will be invoked with an appropriate error code. See Table 78 for the list of error codes.

Parameters:
The "transCommId" parameter specifies the transducer communication session to unlock.

Return result: Error code

10.3.5 IEEE1451Dot0::TransducerServices::TransducerManager::sendCommand

```
IDL: Args::UInt16 sendCommand(
in  Args::UInt16            transCommId,
in  Args::TimeDuration      timeout,
in  Args::UInt8             cmdClassId,
in  Args::UInt8             cmdFunctionId,
in  Args::ArgumentArray     inArgs,
out Args::ArgumentArray     outArgs);
```

This method will perform a blocking operation. The format of input and output arguments are command dependent. The caller shall make sure to use the correct data types for each input argument.

If this is a custom command, the application must use Command TEDS and this argument array must contain the octetArray containing the command.

Parameters:
The "transCommId" parameter specifies the transducer communication session.

The "timeout" is the maximum time to wait before a time-out error. A value of secs == 0, nsecs == –1 means "wait forever."

The "cmdClassId" specifies the desired command class code. See Table 15 for details.

The "cmdFunctionId" specifies the desired command function code. See Clause 7 for details.

The "inArgs" are the input arguments in ArgumentArray form.

The [out] "outArgs" are returned output arguments.

Return result: Error code

10.3.6 IEEE1451Dot0::TransducerServices::TransducerManager::sendCommandRaw

```
IDL: Args::UInt16 sendCommandRaw(
in  Args::UInt16          transCommId,
in  Args::TimeDuration    timeout,
in  Args::UInt8           cmdClassId,
in  Args::UInt8           cmdFunctionId,
```


Copyright © 2007 IEEE. All rights reserved.

```
in  Args::OctetArray     inArgs,
out Args::OctetArray     outArgs);
```

This method will perform a blocking operation. The format of input and output arguments are command dependent. The caller shall make sure to use the correct data types for each input argument.

If this is a custom command, the application must use Command TEDS and this argument array must contain the octetArray containing the command.

Parameters:
The "transCommId" parameter specifies the transducer communication session.

The "timeout" is the maximum time to wait before a time-out error. A value of secs == 0, nsecs == –1 means "wait forever."

The "cmdClassId" specifies the desired command class code. See Table 15 for details.

The "cmdFunctionId" specifies the desired command function code. See Clause 7 for details.

The "inArgs" are the input arguments in ArgumentArray form.

The [out] "outArgs" are returned output arguments.

Return result: Error code

10.3.7 IEEE1451Dot0::TransducerServices::TransducerManager::startCommand

```
IDL: Args::UInt16 startCommand(
in  Args::UInt16             transCommId,
in  Args::TimeInstance       triggerTime,
in  Args::TimeDuration       timeout,
in  Args::UInt8              cmdClassId,
in  Args::UInt8              cmdFunctionId,
in  Args::ArgumentArray      inArgs,
in  AppCallback              callback,
out Args::UInt16             operationId);
```

This method starts a non-blocking operation. The format of input arguments are command dependent. The caller shall make sure to use the correct data types for each input argument.

Parameters:
The "transCommId" parameter specifies the transducer communication session.

The "triggerTime" parameter specifies when to begin the operation. A value specified in the past will result in an immediate time-out failure. A value of secs == 0, nsecs == 0 is a special case that implies immediate action.

The "timeout" is the maximum time to wait before a time-out error. A value of secs == 0, nsecs == –1 means "wait forever."

The "cmdClassId" specifies the desired command class code. See Table 15 for details.

The "cmdFunctionId" specifies the desired command function code. See Clause 7 for details.

The "inArgs" are the input arguments in ArgumentArray form. These are command dependent.

The "callback" specifies the callback interface. The AppCallback::commandComplete( ) method will be invoked.

The [out] "operationId" is the returned operation ID.

Return result: Error code

Copyright © 2007 IEEE. All rights reserved.

10.3.8 IEEE1451Dot0::TransducerServices::TransducerManager::configureAttributes

```
IDL: Args::UInt16 configureAttributes(
in Args::UInt16          transCommId,
in Args::StringArray     attributeNames);
```

This method configures a transCommId for read or measurement stream operations. It specifies which attributes to include in the returned ArgumentArray. See Clause 7 and Clause 8 for details on appropriate names.

Parameters:
The “transCommId” parameter specifies the transducer communication session.

The “attributeNames” specifies the names of desired attributes.

Return result: Error code

10.3.9 IEEE1451Dot0::TransducerServices::TransducerManager::trigger

```
IDL: Args::UInt16 trigger(
in Args::UInt16          transCommId,
in Args::TimeInstance    triggerTime,
in Args::TimeDuration    timeout,
in Args::UInt16          SamplingMode);
```

This method performs a blocking trigger on the specified transCommId.

Parameters:
The “transCommId” parameter specifies the transducer communication session.

The “triggerTime” parameter specifies when to begin the operation. A value specified in the past will result in an immediate time-out failure. A value of secs == 0, nsecs == 0 is a special case that implies immediate action.

The “timeout” is the maximum time to wait before a time-out error. A value of secs == 0, nsecs == –1 means “wait forever.”

The “SamplingMode” specifies the trigger mode. See 5.11 and 7.1.2.4 for details.

Return result: Error code

10.3.10 IEEE1451Dot0::TransducerServices::TransducerManager::startTrigger

```
IDL: Args::UInt16 startTrigger(
in Args::UInt16          transCommId,
in Args::TimeInstance    triggerTime,
in Args::TimeDuration    timeout,
in Args::UInt16          SamplingMode,
in AppCallback           callback,
out Args::UInt16         operationId);
```

This method begins a non-blocking trigger on the specified transCommId.

Parameters:
The “transCommId” parameter specifies the transducer communication session.

Copyright © 2007 IEEE. All rights reserved.

The "triggerTime" parameter specifies when to begin the operation. A value specified in the past will result in an immediate time-out failure. A value of secs == 0, nsecs == 0 is a special case that implies immediate action.

The "timeout" is the maximum time to wait before a time-out error. A value of secs == 0, nsecs == –1 means "wait forever."

The "SamplingMode" specifies the trigger mode. See 5.11 and 7.1.2.4 for details.

The "callback" specifies the callback interface. The AppCallback::triggerComplete( ) method will be invoked.

The [out] "operationId" is the returned operation ID.

Return result: Error code

10.3.11 IEEE1451Dot0::TransducerServices::TransducerManager::clear

```
IDL: Args::UInt16 clear(
in Args::UInt16         transCommId,
in Args::TimeDuration   timeout,
in Args::UInt8          clearMode);
```

This method performs a clear on the specified transCommId.

Parameters:
The "transCommId" parameter specifies the transducer communication session.

The "timeout" is the maximum time to wait before a time-out error. A value of secs == 0, nsecs == –1 means "wait forever."

The "clearMode" specifies the clear mode as shown in Table 87.

Table 87—Clear mode options

Enumeration	Description
0	Reserved
1	Clear all
2	Clear communications channel
3	Clear buffers
4	Reset TIM state machine
5	Clear TEDS cache
6–127	Reserved
128–255	Open to manufacturers

Return result: Error code

10.3.12 IEEE1451Dot0::TransducerServices::TransducerManager::registerStatusChange

```
IDL: Args::UInt16 registerStatusChange(
in Args::UInt16         transCommId,
in Args::TimeDuration   timeout,
in AppCallback          callback,
out Args::UInt16        operationId);
```

This method registers an application callback for TIM status change events on the specified transCommId.

Parameters:
The "transCommId" parameter specifies the transducer communication session.

Copyright © 2007 IEEE. All rights reserved.

The "timeout" is the maximum time to wait before a time-out error. A value of secs == 0, nsecs == –1 means "wait forever."

The "callback" specifies the callback interface. The AppCallback::statusChange( ) method will be invoked.

The [out] "operationId" is the returned operation ID.

Return result: Error code

10.3.13 IEEE1451Dot0::TransducerServices::TransducerManager: :unregisterStatusChange

IDL: `Args::UInt16 unregisterStatusChange( in Args::UInt16 transCommId);`

This method unregisters an application callback for TIM status change events on the specified transCommId.

Parameters:
The "transCommId" parameter specifies the transducer communication session.

Return result: Error code

10.4 IEEE1451Dot0::TransducerServices::TedsManager

IDL: `interface TedsManager { };`

The TedsManager interface is provided by the IEEE 1451.0 layer and is called by the application to provide access to the TEDS. The methods in this interface are listed in Table 88.

Table 88—TedsAccess methods

IEEE1451Dot0::TransducerServices::TedsManager
Args::UInt16 readTeds(in Args::UInt16 transCommId, in Args::TimeDuration time-out, in Args::UInt8 tedsType, out Args::ArgumentArray teds);
Args::UInt16 writeTeds(in Args::UInt16 transCommId, in Args::TimeDuration time-out, in Args::UInt8 tedsType, in Args::ArgumentArray teds);
Args::UInt16 readRawTeds(in Args::UInt16 transCommId, in Args::TimeDuration time-out, in Args::UInt8 tedsType, out Args::OctetArray rawTeds);
Args::UInt16 writeRawTeds(in Args::UInt16 transCommId, in Args::TimeDuration time-out, in Args::UInt8 tedsType, in Args::OctetArray rawTeds);
Args::UInt16 updateTedsCache(in Args::UInt16 transCommId, in Args::TimeDuration time-out, in Args::UInt8 tedsType);

10.4.1 IEEE1451Dot0::TransducerServices::TedsManager::readTeds

```
IDL: Args::UInt16 readTeds(
in  Args::UInt16         transCommId,
in  Args::TimeDuration   timeout,
in  Args::UInt8          tedsType,
out Args::ArgumentArray  teds);
```

This method will read the desired TEDS block from the TEDS cache. If the TEDS is not available from the cache, it will read the TEDS from the TIM. The TEDS information is returned in an ArgumentArray.

Copyright © 2007 IEEE. All rights reserved.

Parameters:
The “transCommId” specifies the desired transducer communication session.

The “timeout” specifies the duration to wait before returning a time-out error if no response is received. A value of secs == 0, nsecs == 0 implies no wait and may be used to only read from the cache. A value of secs == 0, nsecs == –1 implies wait forever.

The “tedsType” parameter specifies what TEDS to return. See Table 17 for TEDS access codes.

The [out] “teds” ArgumentArray contains the TEDS information. The values may be retrieved by attribute name. See Clause 8 for TEDS field names.

Return result: Error code

10.4.2 IEEE1451Dot0::TransducerServices::TedsManager::writeTeds

```
IDL: Args::UInt16 writeTeds(
in Args::UInt16          transCommId,
in Args::TimeDuration    timeout,
in Args::UInt8           tedsType,
in Args::ArgumentArray   teds);
```

This method will write the desired TEDS block to the TIM. The TEDS cache is also updated if the write succeeds. The provided TEDS information is encoded in an ArgumentArray. It will be converted internally to the correct “tuple” form and will be transferred to the TIM in an OctetArray.

The ArgumentArray shall include all the required TEDS fields for the type of TEDS being written. An error will be returned if a required TEDS field is missing.

Parameters:
The “transCommId” specifies the desired transducer communication session.

The “timeout” specifies the duration to wait before returning a time-out error if no response is received. A value of secs == 0, nsecs == 0 implies no wait and may be used to only read from the cache. A value of secs == 0, nsecs == –1 implies wait forever.

The “tedsType” parameter specifies what TEDS to return. See Table 17 for TEDS access codes.

The [out] “teds” ArgumentArray contains the TEDS information. The values may be retrieved by attribute name. See Clause 8 for TEDS field names.

Return result: Error code

10.4.3 IEEE1451Dot0::TransducerServices::TedsManager::readRawTeds

```
IDL: Args::UInt16 readRawTeds(
in  Args::UInt16         transCommId,
in  Args::TimeDuration   timeout,
in  Args::UInt8          tedsType,
out Args::OctetArray     rawTeds);
```

This method will read the desired TEDS block from the TEDS bypassing the TEDS cache. The TEDS information is returned in its raw OctetArray form. The TEDS cache will not be updated.

Copyright © 2007 IEEE. All rights reserved.

Parameters:
The “transCommId” specifies the desired transducer communication session.

The “timeout” specifies the duration to wait before returning a time-out error if no response is received. A value of secs == 0, nsecs == 0 implies no wait and may be used to only read from the cache. A value of secs == 0, nsecs == –1 implies wait forever.

The “tedsType” parameter specifies what TEDS to return. See Table 17 for TEDS access codes.

The [out] “rawTeds” OctetArray contains the raw TEDS information in “tuple” form.

Return result: Error code

10.4.4 IEEE1451Dot0::TransducerServices::TedsManager::writeRawTeds

```
IDL: Args::UInt16 writeRawTeds(
in Args::UInt16         transCommId,
in Args::TimeDuration   timeout,
in Args::UInt8          tedsType,
in Args::OctetArray     rawTeds);
```

This method will write the desired TEDS block to the TIM bypassing the TEDS cache. The provided TEDS information is encoded in “tuple” form in an OctetArray. No verification of the OctetArray will be performed.

CAUTION
Be sure to include all required TEDS fields in the OctetArray because what is written by this method will replace the entire TEDS.

Parameters:
The “transCommId” specifies the desired transducer communication session.

The “timeout” specifies the duration to wait before returning a time-out error if no response is received. A value of secs == 0, nsecs == –1 implies wait forever.

The “tedsType” parameter specifies what TEDS to write. See Table 17 for TEDS access codes.

The “rawTeds” OctetArray contains the raw TEDS information in “tuple” form.

Return result: Error code

10.4.5 IEEE1451Dot0::TransducerServices::TedsManager::updateTedsCache

```
IDL: Args::UInt16 updateTedsCache(
in Args::UInt16         transCommId,
in Args::TimeDuration   timeout,
in Args::UInt8          tedsType);
```

This method will update the TEDS cache. The TEDS checksum will be read from the TIM and compared with the cached TEDS checksum. If the checksums differ, the TEDS will be read from the TIM and stored in the cache.

Parameters:
The “transCommId” specifies the desired transducer communication session.

Copyright © 2007 IEEE. All rights reserved.

The "timeout" specifies the duration to wait before returning a time-out error if no response is received. A value of secs == 0, nsecs == –1 implies wait forever.

The "tedsType" parameter specifies what TEDS to read. See Table 17 for TEDS access codes.

Return result: Error code

10.5 IEEE1451Dot0::TransducerServices::CommManager

```
IDL: interface CommManager { };
```

The CommManager interface is provided by the IEEE 1451.0 layer and is called by the application to provide a common mechanism to manage available communications on an NCAP. The methods are listed in Table 89 and discussed in 10.5.1.

Table 89—CommManager

IEEE1451dot0::TransducerServices::CommManager
Args::UInt16 getCommModule(in Args::UInt8 moduleId, out ModuleCommunication::Comm commObject, out Args::UInt8 type, out Args::UInt8 technologyId);

10.5.1 IEEE1451Dot0::TransducerServices::CommManager::getCommModule

```
IDL: Args::UInt16 getCommModule(
in  Args::UInt8                  moduleId,
out ModuleCommunication::Comm    commObject,
out Args::UInt8                  type,
out Args::UInt8                  technologyId);
```

This returns the abstract "Comm" object for applications that need to bypass IEEE 1451.0 processing and interact directly with the underlying communications object. By consulting the "type" parameter, the application can safely downcast to either the "P2PComm" or the "NetComm" object.

Applications must use extreme caution when accessing the underlying "Comm" objects as incorrect usage may compromise the IEEE 1451.0 layer. This method is provided to allow expansion beyond the IEEE 1451.0 architecture.

Parameters:
The "moduleId" parameter is the desired communications module ID.

The [out] "commObject" parameter is returned to the application and is a reference to the underlying object.

The [out] "type" parameter is returned to the application in order to allow a safe downcast. Valid values are represented in the [out] "technologyId", which specifies the underlying IEEE 1451.X technology. See Table 90.

The [out] "technologyId" specifies the underlying IEEE 1451.X technology. See Table 99.

Table 90—Comm type enumerations

Enumerations	Type code name	Description
0	P2P_TYPE	Specifies a P2PComm object
1	NET_COMM_TYPE	Specifies a NetComm object
2–255	Reserved	

Return result: Error code

Copyright © 2007 IEEE. All rights reserved.

10.6 IEEE1451Dot0::TransducerServices::AppCallback

IDL: `interface AppCallback { };`

The AppCallback interface is provided by applications and is called by the IEEE 1451.0 layer to provide access to non-blocking I/O and measurement streams. The interface methods are listed in Table 91.

Table 91—Application callback interface methods

IEEE1451Dot0::TransducerServices::AppCallback
Args::UInt16 measurementUpdate(in Args::UInt16 operationId, in Args::ArgumentArray measValues, in Args::UInt16 status);
Args::UInt16 actuationComplete (in Args::UInt16 operationId, in Args::UInt16 status);
Args::UInt16 statusChange(in Args::UInt16 operationId, in Args::UInt16 status);
Args::UInt16 commandComplete(in Args::UInt16 operationId, in Args::ArgumentArray outArgs, in Args::UInt16 status);
Args::UInt16 triggerComplete(in Args::UInt16 operationId, in Args::UInt16 status);

10.6.1 IEEE1451Dot0::TransducerServices::AppCallback::measurementUpdate

```
IDL: Args::UInt16 measurementUpdate(
in Args::UInt16           operationId,
in Args::ArgumentArray    measValues,
in Args::UInt16           status);
```

This method will be invoked following a startRead( ) or startStream( ) call. For non-blocking operations, it provides measurements back to the application. For the stream case, this callback will be invoked every time new measurement data are available.

Parameters:
The “operationId” specifies the desired operation ID that was returned in the startRead( ) or startStream( ) call.

The “measValues” contains the measurement information. The values may be retrieved by attribute name. See Clause 7 for attribute names. See 10.2.6 read( ) for more details.

The “status” specifies the error code from the non-blocking read or stream operation.

Return result: The application shall return a status code back to the IEEE 1451.0 layer. See 9.3.1.2 for error codes.

10.6.2 IEEE1451Dot0::TransducerServices::AppCallback::actuationComplete

```
IDL: Args::UInt16 actuationComplete (
in Args::UInt16      operationId,
in Args::UInt16      status);
```

This method will be invoked following a startWrite( ) call. For non-blocking operations, it provides status information back to the application.

Copyright © 2007 IEEE. All rights reserved.

Parameters:
The "operationId" specifies the desired operation ID that was returned in the startWrite( ) call.

The "status" specifies the error code from the non-blocking write operation.

Return result: The application shall return a status code back to the IEEE 1451.0 layer. See 9.3.1.2 for error codes.

10.6.3 IEEE1451Dot0::TransducerServices::AppCallback::statusChange

```
IDL: Args::UInt16 statusChange(
in Args::UInt16      operationId,
in Args::UInt16      status);
```

This method will be invoked following a registerStatusChange( ) call.

Parameters:
The "operationId" specifies the desired operation ID that was returned in the registerStatusChange( ) call.

The "status" specifies the TIM or TransducerChannel status information.

Return result: The application shall return a status code back to the IEEE 1451.0 layer. See 9.3.1.2 for error codes.

10.6.4 IEEE1451Dot0::TransducerServices::AppCallback::commandComplete

```
IDL: Args::UInt16 commandComplete(
in Args::UInt16          operationId,
in Args::ArgumentArray   outArgs,
in Args::UInt16          status);
```

This method will be invoked following a startCommand( ). It provides the output ArgumentArray back to the application.

Parameters:
The "operationId" specifies the desired operation ID that was returned in the startRead( ) or startStream( ) call.

The "outArgs" contains the returned ArgumentArray. This information is specific to each command.

The "status" specifies the error code from the non-blocking send command operation.

Return result: The application shall return a status code back to the IEEE 1451.0 layer. See 9.3.1.2 for error codes.

10.6.5 IEEE1451Dot0::TransducerServices::AppCallback::triggerComplete

```
IDL: Args::UInt16 triggerComplete(
in Args::UInt16         operationId,
in Args::UInt16         status);
```

This method will be invoked following a startTrigger( ). It provides status information to inform the application when the trigger has completed.

Copyright © 2007 IEEE. All rights reserved.

Parameters:
The "operationId" specifies the desired operation ID that was returned in the startTrigger( ) call.

The "status" specifies the error code from the non-blocking trigger command operation.

Return result: The application shall return a status code back to the IEEE 1451.0 layer. See 9.3.1.2 for error codes.

11. Module Communications API

IDL: `module ModuleCommunication { };`

The Module Communications API provides the interface between the functions defined by the IEEE 1451.0 layer and the communications functions defined by another member of the IEEE 1451 family of standards.

These IEEE 1451.0 interfaces are inside the IDL module "IEEE1451Dot0".

NOTE—Throughout this clause, the nomenclature "IEEE 1451.0" or "1451.0" refers to a device or part of a device that is in compliance with this standard. The term "IEEE 1451.X" or "1451.X" refers to a device or part of a device that is in compliance with IEEE Std 1451.2-1997, IEEE Std 1451.3-2003, IEEE Std 1451.5-2007 [B4], IEEE P1451.6™ [B3], or other similar standard. IEEE Std 1451.1-1999 and IEEE Std 1451.4-2004 are excluded from this list.

Table 92—Transducer Services API classes and interfaces

Interface	Description
Comm	The Comm abstract interface provides the mechanisms to control the "life cycle" of an IEEE 1451.X instance.
P2PComm	The P2PComm interface is provided to perform point-to-point communication operations.
NetComm	The NetComm interface is provided to perform network communication operations.
Registration	The Registration interface provides methods to register an IEEE 1451.X module with the IEEE 1451.0 layer.
P2PRegistration	The P2PRegistration interface provides methods to register specific TIMs with the IEEE 1451.0 layer.
NetRegistration	The NetRegistration interface provides methods to register specific TIMs and groups of TIMs with the IEEE 1451.0 layer.
Recieve	The abstract Receive interface does not define any generic methods. It is provided for future expansion.
P2PRecieve	The P2PReceive interface provides methods for a IEEE 1451.X point-to-point communications module to notify the IEEE 1451.0 layer that a message has been received. It also provides the method for aborting an operation.
NetRecieve	The NetReceive interface provides methods for IEEE 1451.X network communications module to notify the IEEE 1451.0 layer that a message has been received. It also provides the method for aborting an operation.

11.1 IEEE1451Dot0::ModuleCommunication::Comm

IDL: `interface Comm { };`

The Comm abstract interface is provided by the IEEE 1451.X layer and is called by the IEEE 1451.0 layer to provide a common mechanism to control the "life cycle" of an IEEE 1451.X instance. The Comm methods are listed in Table 93.

Copyright © 2007 IEEE. All rights reserved.

Table 93—Comm methods

IEEE1451Dot0::ModuleCommunication::Comm
Args::UInt16 init( )
Args::UInt16 shutdown( )
Args::UInt16 sleep(in Args::TimeDuration duration)
Args::UInt16 wakeup()
Args::UInt16 setLocalConfiguration(in Args::ArgumentArray params)
Args::UInt16 getLocalConfiguration(out Args::ArgumentArray params);
Args::UInt16 sendLocalCommand(in Args::UInt8 cmdClassId, in Args::UInt8 cmdFunctionId, in Args::ArgumentArray inArgs, out Args::ArgumentArray outArgs);
Args::UInt16 describe(out Args::UInt8 logicalType, out Args::UInt8 physicalType, out Args::_String name);

11.1.1 IEEE1451Dot0::ModuleCommunication::Comm::init

IDL: `Args::UInt16 init( );`

This method will be called at startup and when the IEEE 1451.0 layer needs to reset this IEEE 1451.X instance to the power-on state. In dynamic cases, each IEEE 1451.X instance shall call the appropriate Registration::Register1451.X( ) method to register this instance with the IEEE 1451.0 layer.

Parameters:
None.

Return result: Error code

11.1.2 IEEE1451Dot0::ModuleCommunication::Comm::shutdown

IDL: `Args::UInt16 shutdown( );`

This method will be called to coordinate an orderly shutdown of this IEEE 1451.X instance. In dynamic cases, each IEEE 1451.X instance shall call the Registration::unRegister1451.X( ) method to un-register this instance from the IEEE 1451.0 syste,.

Parameters:
None.

Return result: Error code

11.1.3 IEEE1451Dot0::ModuleCommunication::Comm::sleep

IDL: `Args::UInt16 sleep(in Args::TimeDuration duration);`

This method is called by the IEEE 1451.0 layer to put this IEEE 1451.X instance into the sleep (i.e., low-power) state. This will be when the IEEE 1451.0 layer does not intend to communicate through this IEEE 1451.X interface for a significant period of time.

Parameters:
The “duration” parameter is the amount of time to sleep. A negative duration indicates the IEEE 1451.X device should sleep until the wakeup method is invoked.

Return result: Error code

Copyright © 2007 IEEE. All rights reserved.

11.1.4 IEEE1451Dot0::ModuleCommunication::Comm::wakeup

```
IDL: Args::UInt16 wakeup();
```

This method is called by the IEEE 1451.0 layer to wakeup this IEEE 1451.X instance from the sleep (i.e., low-power) state. This will be when the IEEE 1451.0 layer has previously put this IEEE 1451.X interface to sleep and now needs to resume communication.

Parameters:
None.

Return result: Error code

11.1.5 IEEE1451Dot0::ModuleCommunication::Comm::performOperation

```
IDL: Args::UInt16 performOperation(
in  Args::UInt16          operationId,
in  Args::TimeDuration    timeout,
in  Args::ArgumentArray inArgs,
out Args::ArgumentArray outArgs);
```

This low-level mechanism sends an arbitrary command to the local IEEE 1451.X layer. This method will perform a blocking operation. The format of input and output arguments is command dependent. The caller shall make sure to use the correct data types for each input argument.

Parameters:
The “operationId” specifies the desired command class code. The IEEE 1451.X layer will define legal operations.

The “timeout” parameter specifies the maximum amount of time the caller is willing to block before a time-out error should be returned. A value of secs == 0, nsecs == –1 means “wait forever.”

The “inArgs” are the input arguments in ArgumentArray form.

The [out] “outArgs” are returned output arguments.

Return result: Error code

11.1.6 IEEE1451Dot0::ModuleCommunication::Comm::setLocalConfiguration(delete)

```
IDL: Args::UInt16 setLocalConfiguration (in Args::ArgumentArray params
);
```

This method is called by the IEEE 1451.0 layer to configure this local instance of the IEEE 1451.X interface. The contents of the “params” ArgumentArray are IEEE 1451.X specific.

Parameters:
The “params” parameter is an ArgumentArray of configuration state variables. Each state variable of interest to the IEEE 1451.X layer should be retrieved from this array to set up the IEEE 1451.X instance.

Return result: Error code

Copyright © 2007 IEEE. All rights reserved.

11.1.7 IEEE1451Dot0::ModuleCommunication::Comm::getLocalConfiguration(delete)

```
IDL: Args::UInt16 getLocalConfiguration (outout Args::ArgumentArray
params );
```

This method is called by the IEEE 1451.0 layer to retrieve the configuration for local instance of the IEEE 1451.X interface. The contents of the "params" ArgumentArray are IEEE 1451.X specific.

Parameters:
The [out] "params" parameter is an ArgumentArray of configuration state variables. Each state variable of interest to the IEEE 1451.X layer should be returned in this array to allow setup of the IEEE 1451.X instance in the future.

Return result: Error code

11.1.8 IEEE1451Dot0::ModuleCommunication::Comm::sendLocalCommand(Delete)

```
IDL: Args::UInt16 sendLocalCommand(
in  Args::UInt8          cmdClassId,
in  Args::UInt8          cmdFunctionId,
in  Args::ArgumentArray inArgs,
out Args::ArgumentArray outArgs);
```

This low-level mechanism sends an arbitrary command to the local IEEE 1451.X layer. This method will perform a blocking operation. The format of input and output arguments is command dependent. The caller shall make sure to use the correct data types for each input argument.

Parameters:
The "cmdClassId" specifies the desired command class code. See Table 15 for details.

The "cmdFunctionId" specifies the desired command function code. See Clause 7 for details.

The "inArgs" are the input arguments in ArgumentArray form.

The [out] "outArgs" are returned output arguments.

Return result: Error code

11.1.9 IEEE1451Dot0::ModuleCommunication::Comm::describe

```
IDL: UInt16 describe (
out UInt8          logicalType,
out UInt8          physicalType,
out _String name );
```

This generic mechanism describes the type of the underlying IEEE 1451.X object.

Parameters:
The [out] "logicalType" specifies the type of interface. See Table 94 for the details.

The [out] "physicalType" specifies the type of the physical layer. See Table 99 for details.

The [out] "name" is a human readable name that was provided during the registration process.

Return result: Error code

Copyright © 2007 IEEE. All rights reserved.

Table 94—Logical interface type

Enumeration	Logical type
0	Reserved
1	Point to point (P2P)
2	Network (Net)
3–127	Reserved
128–255	Open to manufacturers

11.2 IEEE1451Dot0::ModuleCommunication::P2PComm

IDL: `interface P2PComm { };`

The P2PComm interface is appropriate in cases where the node only receives and optionally replies to incoming messages (i.e., it never initiates communications). Also, the P2PComm interface may be used in the case where the node is only initiating communication to a single destination.

The P2PComm interface is provided by the IEEE 1451.X layer and is called by the IEEE 1451.0 layer to perform communication operations. The point-to-point Comm methods are listed in Table 95.

Table 95—Point-to-point Comm methods

IEEE1451Dot0::ModuleCommunication::P2PComm
Args::UInt16 read(in Args::TimeDuration time-out, inout Args::UInt32 len, out Args::OctetArray payload, out Args::_Boolean last);
Args::UInt16 write(in Args::TimeDuration time-out, in Args::OctetArray payload, in Args::_Boolean last);
Args::UInt16 flush( );
Args::UInt16 readSize(out Args::UInt32 cacheSize);
Args::UInt16 setPayloadSize(in Args::UInt32 size);
Args::UInt16 abort( );
Args::UInt16 commStatus(out Args::UInt16 statusCode);
Args::UInt16 setRemoteConfiguration(in Args::TimeDuration time-out, in Args::ArgumentArray params);
Args::UInt16 getRemoteConfiguration(in Args::TimeDuration time-out, out Args::ArgumentArray params);
Args::UInt16 sendRemoteCommand(in Args::TimeDuration time-out, in Args::UInt8 cmdClassId, in Args::UInt8 cmdFunctionId, in Args::ArgumentArray inArgs, out Args::ArgumentArray outArgs);

11.2.1 IEEE1451Dot0::ModuleCommunication::P2PComm::read

```
IDL: Args::UInt16 read(
in  Args::TimeDuration      timeout,
in  Args::UInt32            maxLen,
out Args::OctetArray        payload,
out Args::_Boolean          last);
```

This method is called by the IEEE 1451.0 layer to retrieve information on the current communication operation. This method is always called by the IEEE 1451.0 layer on the receiving node for both one-way and two-way communication sequences. For two-way, this method is also called on the initiating node to retrieve the response.

In cases where the IEEE 1451.0 layer makes multiple read( ) calls for large payloads, the “len” value returns the length of each transfer, not the total length. See 11.2.4 readSize( ) method to manage the cached length.

Parameters:

Copyright © 2007 IEEE. All rights reserved.

The “timeout” parameter specifies the maximum amount of time the caller is willing to block before a time-out error should be returned. A value of secs == 0, nsecs == –1 means “wait forever.”

The “maxLen” parameter indicates the maximum number of octets to be transferred by the IEEE 1451.X layer.

The [out] “payload” parameter is the OctetArray provided by the IEEE 1451.0 layer to the IEEE 1451.X layer.The IEEE 1451.X layer will transfer available data into this array.

The [out] “last” parameter indicates if additional calls to read( ) should be performed by the IEEE 1451.0 layer. A false value indicates that the IEEE 1451.X layer has more octets to communicate to the IEEE 1451.0 layer and that the IEEE 1451.0 layer must make additional read( ) calls. A true value indicates that the complete payload has been transferred from the IEEE 1451.X layer to the IEEE 1451.0 layer.

Return result: Error code

11.2.2 IEEE1451Dot0::ModuleCommunication::P2PComm::write

```
IDL: Args::UInt16 write(
in Args::TimeDuration      timeout,
in Args::OctetArray        payload,
in Args::_Boolean          last);
```

This method is called by the IEEE 1451.0 layer to begin or continue a communication operation. It is always called on the initiating node to begin communication. In two-way communications, it is also called on the receiving node to provide the response.

Parameters:
The “timeout” parameter specifies the maximum amount of time the caller is willing to block before a time-out error should be returned. A value of secs == 0, nsecs == –1 means “wait forever.”

The “payload” parameter is the OctetArray to be communicated.

The “last” parameter indicates if additional calls to write( ) will be made by the IEEE 1451.0 layer to provide additional portions of the payload. A false value indicates that the IEEE 1451.0 layer has more octets to send and will call write( ) additional times. A true value indicates that the complete payload has been transferred from the IEEE 1451.0 layer to the IEEE 1451.X layer.

Return result: Error code

11.2.3 IEEE1451Dot0::ModuleCommunication::P2PComm::flush

```
IDL: Args::UInt16 flush();
```

This method is called by the IEEE 1451.0 layer to flush any cached information to the remote side. In most cases, this call is not used because the IEEE 1451.X layer will always perform the transfer when the write( ) call is invoked with “last” set to true.

Parameters:
None.

Return result: Error code

Copyright © 2007 IEEE. All rights reserved.

11.2.4 IEEE1451Dot0::ModuleCommunication::P2PComm::readSize

IDL: `Args::UInt16 readSize( out Args::UInt32 cacheSize);`

This method is called by the IEEE 1451.0 layer to retrieve the number of octets available to be immediately read. This represents the size of the cached data. This value may be less than the size of the complete payload if the IEEE 1451.X layer has performed segmentation.

Parameters:
The [out] "cacheSize" returns the number of octets available to be immediately read.

Return result: Error code

11.2.5 IEEE1451Dot0::ModuleCommunication::P2PComm::setPayloadSize

IDL: `Args::UInt16 setPayloadSize( in Args::UInt32 size);`

This method is called by the IEEE 1451.0 layer to set the total number of octets available in the full payload. This method should be called when the IEEE 1451.0 layer expects to make multiple calls to write( ) to pass the payload to the IEEE 1451.X layer in pieces. In cases where a single write( ) call is made (i.e., last is true on the first call), this call is not needed as the IEEE 1451.X layer shall internally call this method to record the payload size.

Parameters:
The "size" parameter specifies the full size of the payload.

Return result: Error code

11.2.6 IEEE1451Dot0::ModuleCommunication::P2PComm::abort

IDL: `Args::UInt16 abort();`

This method is called by the IEEE 1451.0 layer on the initiating node to abort or reset a communication channel. If the IEEE 1451.X layer has already initiated communication to the remote node, the IEEE 1451.X layer shall make an effort to abort the remote processing. The "commId" may be used for subsequent communications after this call. This method will not close the communication channel.

Parameters:
None.

Return result: Error code

11.2.7 IEEE1451Dot0::ModuleCommunication::P2PComm::commStatus

IDL: `Args::UInt16 commStatus( out Args::UInt16 statusCode );`

This method is called by the IEEE 1451.0 layer on the initiating node to retrieve status information on the local and remote state machine. Table 96 provides a list of the available status.

Parameters:
The [out] "statusCode" parameter returns the status information for the local (upper octet) and remote (lower octet) state machines.

Copyright © 2007 IEEE. All rights reserved.

Return result: Error code

Table 96—State machine status codes

Status enumeration	State
1	Idle
2	InitiatorWriting
3	InitiatorClosing
4	InitiatorReceivePending
5	InitiatorReading
6	InitiatorAborting
7	ReceiverIncomingMsg
8	ReceiverReading
9	ReceiverWriting
10	ReceiverOutgoingMsg
11	ReceiverAborting
12–127	Reserved
128–255	Open to manufacturers

11.2.8 IEEE1451Dot0::ModuleCommunication::P2PComm::setRemoteConfiguration

```
IDL: Args::UInt16 setRemoteConfiguration (
in Args::TimeDuration      timeout,
in Args::ArgumentArray     params );
```

This method is called by the IEEE 1451.0 layer on the initiating node to set the configuration on the remote node. In cases where the params array needs to be transmitted to the remote node, it is recommended to use the standard Encoder/Decoder mechanism to convert the ArgumentArray to/from an OctetArray.

Parameters:
The “timeout” is the maximum time to wait before a time-out error. A value of secs == 0, nsecs == –1 means “wait forever.”

The “params” parameter is an ArgumentArray of configuration state variables. Each state variable of interest to the IEEE 1451.X layer should be retrieved from this array to set up the remote IEEE 1451.X instance.

Return result: Error code

11.2.9 IEEE1451Dot0::ModuleCommunication::P2PComm::getRemoteConfiguration

```
IDL: Args::UInt16 getRemoteConfiguration (
in Args::TimeDuration      timeout,
out Args::ArgumentArray    params );
```

This method is called by the IEEE 1451.0 layer on the initiating node to get the configuration from the remote node. In cases where the params array needs to be transmitted from the remote node, it is recommended to use the standard Encoder/Decoder mechanism to convert the ArgumentArray to/from an OctetArray.

Parameters:
The “timeout” is the maximum time to wait before a time-out error. A value of secs == 0, nsecs == –1 means “wait forever.”

Copyright © 2007 IEEE. All rights reserved.

The [out] "params" parameter is an ArgumentArray of configuration state variables. Each state variable of interest to the IEEE 1451.X layer should be stored in this array.

Return result: Error code

11.2.10 IEEE1451Dot0::ModuleCommunication::P2PComm::sendRemoteCommand

```
IDL: Args::UInt16 sendRemoteCommand(
in  Args::TimeDuration      timeout,
in  Args::UInt8             cmdClassId,
in  Args::UInt8             cmdFunctionId,
in  Args::ArgumentArray     inArgs,
out Args::ArgumentArray     outArgs );
```

This low-level mechanism sends an arbitrary command to the remote IEEE 1451.X layer. It is recommended that the IEEE 1451.X layer use the standard Encoder/Decoder to convert the ArgumentArrays to/from OctetArrays. This method will perform a blocking operation. The format of input and output arguments is command dependent. The caller shall make sure to use the correct data types for each input argument.

Parameters:
The "timeout" is the maximum time to wait before a time-out error. A value of secs == 0, nsecs == –1 means "wait forever."

The "cmdClassId" specifies the desired command class code. See Table 15 for details.

The "cmdFunctionId" specifies the desired command function code. See Clause 7 for details.

The "inArgs" are the input arguments in ArgumentArray form.

The [out] "outArgs" are returned output arguments.

Return result: Error code

11.3 IEEE1451Dot0::ModuleCommunication::NetComm

```
IDL: interface NetComm { };
```

The NetComm interface is appropriate when the node needs to initiate access to one or more than one destination.

The NetComm interface is provided by the IEEE 1451.X layer and is called by the IEEE 1451.0 layer to perform communication operations. The methods that implement this interface are listed in Table 97.

Copyright © 2007 IEEE. All rights reserved.

Table 97—Network Comm interface methods

IEEE1451Dot0::ModuleCommunication::NetComm
Args::UInt16 open(in Args::UInt16 destId, in Args::_Boolean twoWay, out Args::UInt16 maxPayloadLen, out Args::UInt16 commId);
Args::UInt16 openQoS(in Args::UInt16 destId, in Args::_Boolean twoWay, out Args::UInt16 maxPayloadLen, out Args::UInt16 commId, inout Args::QoSParams qosParams);
Args::UInt16 close(in Args::UInt16 commId);
Args::UInt16 readMsg(in Args::UInt16 commId, in Args::TimeDuration time-out, inout Args::UInt32 len, out Args::OctetArray payload, out Args::_Boolean last);
Args::UInt16 readRsp(in Args::UInt16 commId, in Args::TimeDuration time-out, in Args::UInt16 msgId, in Args::UInt32 maxLen, out Args::OctetArray payload, out Args::_Boolean last);
Args::UInt16 writeMsg(in Args::UInt16 commId, in Args::TimeDuration time-out, in Args::OctetArray payload, in Args::_Boolean last, in Args::UInt16 msgId);
Args::UInt16 writeRsp(in Args::UInt16 commId, in Args::TimeDuration time-out, in Args::OctetArray payload, in Args::_Boolean last);
Args::UInt16 flush(in Args::UInt16 commId);
Args::UInt16 readSize(in Args::UInt16 commId, out Args::UInt32 cacheSize);
Args::UInt16 setPayloadSize(in Args::UInt16 commId, in Args::UInt32 size);
Args::UInt16 abort(in Args::UInt16 commId);
Args::UInt16 commStatus(in Args::UInt16 commId, in Args::UInt16 msgId, out Args::UInt16 statusCode);
Args::UInt16 discoverDestinations( );
Args::UInt16 joinGroup(in Args::UInt16 groupId, in Args::UInt16 destId);
Args::UInt16 leaveGroup(in Args::UInt16 groupId, in Args::UInt16 destId);
Args::UInt16 lookupDestId(in Args::UInt16 commId, out Args::UInt16 destId);
Args::UInt16 setRemoteConfiguration(in Args::UInt16 commId, in Args::TimeDuration time-out, in Args::ArgumentArray params);
Args::UInt16 getRemoteConfiguration(in Args::UInt16 commId, in Args::TimeDuration time-out, out Args::ArgumentArray params);
Args::UInt16 sendRemoteCommand(in Args::UInt16 commId, in Args::TimeDuration time-out, in Args::UInt8 cmdClassId, in Args::UInt8 cmdFunctionId, in Args::ArgumentArray inArgs, out Args::ArgumentArray outArgs);

11.3.1 IEEE1451Dot0::ModuleCommunication::NetComm::open

```
IDL: Args::UInt16 open(
in  Args::UInt16      destId,
in  Args::_Boolean    twoWay,
out Args::UInt16      maxPayloadLen,
out Args::UInt16      commId);
```

This method is called by the IEEE 1451.0 layer on the initiating node to open a communication channel.

Parameters:
The “destId” parameter specifies the destination for the receiving NCAP or TIM.

If the “twoWay” parameter is true, it indicates that the initiator expects a return result from the communication.

The [out] “maxPayloadLen” parameter indicates the maximum payload size that will be accepted in subsequent write and read operations.

The [out] “commId” parameter is returned from this call.

Return result: Error code

Copyright © 2007 IEEE. All rights reserved.

11.3.2 IEEE1451Dot0::ModuleCommunication::NetComm::openQoS

```
IDL: Args::UInt16 openQoS(
in    Args::UInt16        destId,
in    Args::_Boolean      twoWay,
out   Args::UInt16        maxPayloadLen,
out   Args::UInt16        commId,
inout Args::QoSParams     qosParams);
```

This method is called by the IEEE 1451.0 layer on the initiating node to open a communication channel with "quality of service" parameters. If the call fails with QOS_FAILURE, the qosParams will be modified to indicate values that the IEEE 1451.X layer can provide.

Parameters:
The "destId" parameter specifies the destination for the receiving NCAP or TIM.

If the "twoWay" parameter is true, it indicates that the initiator expects a return result from the communication.

The [out] "maxPayloadLen" parameter indicates the maximum payload size that will be accepted in subsequent write and read operations.

The [out] "commId" parameter is returned from this call.

The [inout] "qosParams" parameter provides desired "quality of service" parameters.

Return result: Error code

11.3.3 IEEE1451Dot0::ModuleCommunication::NetComm::close

```
IDL: Args::UInt16 close(Args::UInt16 commId);
```

This method is called by the IEEE 1451.0 layer to close a communication channel. This is called the IEEE 1451.0 layer on the initiating node. No further communications on this "commId" are allowed once the channel is closed.

Parameters:
The "commId" parameter specifies the communication channel.

Return result: Error code

11.3.4 IEEE1451Dot0::ModuleCommunication::NetComm::readMsg

```
IDL: Args::UInt16 readMsg(
in    Args::UInt16         commId,
in    Args::TimeDuration   timeout,
inout Args::UInt32         len,
out   Args::OctetArray     payload,
out   Args::_Boolean       last);
```

This method is called by the IEEE 1451.0 layer to retrieve information on the current communication operation. This method is always called by the IEEE 1451.0 layer on the receiving node for both one-way and two-way communication sequences.

Copyright © 2007 IEEE. All rights reserved.

In cases where the IEEE 1451.0 layer makes multiple readMsg( ) calls for large payloads, the "len" value returns the length of each transfer, not the total length. See 11.3.9, the readSize( ) method, to manage the cached length.

Parameters:
The "commId" parameter specifies the communication channel.

The "timeout" parameter specifies the maximum amount of time the caller is willing to block before a time-out error should be returned. A value of secs == 0, nsecs == –1 means "wait forever."

As an "in" function the "len" parameter indicates the maximum number of octets to be transferred by the IEEE 1451.X layer. In the reply it indicates the number of octets transferred.

The [out] "payload" parameter is the OctetArray provided by the IEEE 1451.0 layer to the IEEE 1451.X layer. The IEEE 1451.X layer will transfer available data into this array. Note the length of the returned OctetArray may be less than the maxLen parameter.

The [out] "last" parameter indicates if additional calls to readMsg( ) should be performed by the IEEE 1451.0 layer. A false value indicates that the IEEE 1451.X layer has more octets to communicate to the IEEE 1451.0 layer and that the IEEE 1451.0 layer must make additional readMsg( ) calls. A true value indicates that the complete payload has been transferred from the IEEE 1451.X layer to the IEEE 1451.0 layer.

Return result: Error code

11.3.5 IEEE1451Dot0::ModuleCommunication::NetComm::readRsp

```
IDL: Args::UInt16 readRsp(
in  Args::UInt16         commId,
in  Args::TimeDuration   timeout,
in  Args::UInt16         msgId,
in  Args::UInt32         maxLen,
out Args::OctetArray     payload,
out Args::_Boolean       last);
```

This method is called by the IEEE 1451.0 layer to retrieve the response information on the current communication operation. This method is always called by the IEEE 1451.0 layer on the initiating node for two-way communication sequences.

In cases where the IEEE 1451.0 layer makes multiple readRsp( ) calls for large payloads, the "len" value returns the length of each transfer, not the total length. See 11.3.9, the readSize( ) method, to manage the cached length.

Parameters:
The "commId" parameter specifies the communication channel.

The "timeout" parameter specifies the maximum amount of time the caller is willing to block before a time-out error should be returned. A value of secs == 0, nsecs == –1 means "wait forever."

The "msgId" parameter shall be the same value that was provided in the writeMsg( ) call that began this transaction. See the notifyRsp( ) callback on how the initiator is notified that the response is ready.

The "maxLen" parameter indicates the maximum number of octets to be transferred by the IEEE 1451.X layer.

The [out] "payload" parameter is the OctetArray provided by the IEEE 1451.0 layer to the IEEE 1451.X layer. The IEEE 1451.X layer will transfer available data into this array. Note the length of the OctetArray may be less than the maxLen parameter.

Copyright © 2007 IEEE. All rights reserved.

The [out] "last" parameter indicates if additional calls to readRsp( ) should be performed by the IEEE 1451.0 layer. A false value indicates that the IEEE 1451.X layer has more octets to communicate to the IEEE 1451.0 layer and that the IEEE 1451.0 layer must make additional readRsp( ) calls. A true value indicates that the complete payload has been transferred from the IEEE 1451.X layer to the IEEE 1451.0 layer.

Return result: Error code

11.3.6 IEEE1451Dot0::ModuleCommunication::NetComm::writeMsg

```
IDL: Args::UInt16 writeMsg(
in Args::UInt16          commId,
in Args::TimeDuration    timeout,
in Args::OctetArray      payload,
in Args::_Boolean        last,
in Args::UInt16          msgId);
```

This method is called by the IEEE 1451.0 layer to begin or continue a communication operation. It is only called on the initiating node to begin communication. For two-way communications, see writeRsp( ) for how the receiving node provides the response.

Parameters:
The "commId" parameter specifies the communication channel.

The "timeout" parameter specifies the maximum amount of time the caller is willing to block before a time-out error should be returned. A value of secs == 0, nsecs == –1 means "wait forever."

The "payload" parameter is the OctetArray to be communicated.

The "last" parameter indicates if additional calls to writeMsg( ) will be made by the IEEE 1451.0 layer to provide additional portions of the payload. A false value indicates that the IEEE 1451.0 layer has more octets to send and will call writeMsg( ) additional times. A true value indicates that the complete payload has been transferred from the IEEE 1451.0 layer to the IEEE 1451.X layer.

The "msgId" parameter defines the "message ID" for this transaction. In two-way transactions, the IEEE 1451.X layer shall pass this same "msgId" back to the initiator in the NetReceive::notifyRsp( ) call. A value of 0 indicates a "one way" call.

Return result: Error code

11.3.7 IEEE1451Dot0::ModuleCommunication::NetComm::writeRsp

```
IDL: Args::UInt16 writeRsp(
in Args::UInt16          commId,
in Args::TimeDuration    timeout,
in Args::OctetArray      payload,
in Args::_Boolean        last);
```

This method is called by the IEEE 1451.0 layer to return a response to a command.

Parameters:
The "commId" parameter specifies the communication channel.

The "timeout" parameter specifies the maximum amount of time the caller is willing to block before a timeout error should be returned. A value of secs == 0, nsecs == –1 means "wait forever."

The "payload" parameter is the OctetArray to be communicated.

Copyright © 2007 IEEE. All rights reserved.

The "last" parameter indicates if additional calls to writeRsp( ) will be made by the IEEE 1451.0 layer to provide additional portions of the payload. A false value indicates that the IEEE 1451.0 layer has more octets to send and will call writeRsp( ) additional times. A true value indicates that the complete payload has been transferred from the IEEE 1451.0 layer to the IEEE 1451.X layer.

Return result: Error code

11.3.8 IEEE1451Dot0::ModuleCommunication::NetComm::flush

```
IDL: Args::UInt16 flush(in Args::UInt16 commId);
```

This method is called by the IEEE 1451.0 layer to flush any cached information to the remote side. In most cases, this call is not used because the IEEE 1451.X layer will always perform the transfer when the writeMsg( ) or writeRsp( ) call is invoked with "last" set to true.

Parameters:
The "commId" parameter specifies the communication channel.

Return result: Error code

11.3.9 IEEE1451Dot0::ModuleCommunication::NetComm::readSize

```
IDL: Args::UInt16 readSize(
in Args::UInt16   commId,
out Args::UInt32  cacheSize);
```

This method is called by the IEEE 1451.0 layer to retrieve the number of octets available to be immediately read. This represents the size of the cached data. This value may be less than the size of the complete payload if the IEEE 1451.X layer has performed segmentation.

Parameters:
The "commId" parameter specifies the communication channel.

The [out] "cacheSize" returns the number of octets available to be immediately read.

Return result: Error code

11.3.10 IEEE1451Dot0::ModuleCommunication::NetComm::setPayloadSize

```
IDL: Args::UInt16 setPayloadSize(
in Args::UInt16   commId,
in Args::UInt32   size);
```

This method is called by the IEEE 1451.0 layer to set the total number of octets available in the full payload. This method should be called when the IEEE 1451.0 layer expects to make multiple calls to writeMsg( ) or writeRsp( ) to pass the payload to the IEEE 1451.X layer in pieces. In cases where a single writeMsg( ) or writeRsp( ) call is made (i.e., last is true), this call is not needed as the IEEE 1451.X layer shall internally call this method to record the payload size.

Copyright © 2007 IEEE. All rights reserved.

Parameters:
The “commId” parameter specifies the communication channel.

The “size” parameter specifies the full size of the payload.

Return result: Error code

11.3.11 IEEE1451Dot0::ModuleCommunication::NetComm::abort

```
IDL: Args::UInt16 abort(in Args::UInt16 commId);
```

This method is called by the IEEE 1451.0 layer on the initiating node to abort or reset a communication channel. If the IEEE 1451.X layer has already initiated communication to the remote node, the IEEE 1451.X layer shall make an effort to abort the remote processing. The “commId” may be used for subsequent communications after this call. This method will not close the communication channel.

Parameters:
The “commId” parameter specifies the communication channel.

Return result: Error code

11.3.12 IEEE1451Dot0::ModuleCommunication::NetComm::commStatus

```
IDL: Args::UInt16 commStatus(
in  Args::UInt16    commId,
in  Args::UInt16    msgId,
out Args::UInt16    statusCode);
```

This method is called by the IEEE 1451.0 layer on the initiating node to retrieve status information on the local and remote state machine. Table 96 provides a list of the available status.

Parameters:
The “commId” parameter specifies the communication channel.

The “msgId” parameter specifies the message transaction ID.

The [out] “statusCode” parameter returns the status information for the local (upper octet) and remote (lower octet) state machines.

Return result: Error code

11.3.13 IEEE1451Dot0::ModuleCommunication::NetComm::discoverDestinations

```
IDL: Args::UInt16 discoverDestinations( );
```

This method is called by the IEEE 1451.0 layer to force a discovery and registration of all TIMs accessible through this IEEE 1451.X instance. In response, the IEEE 1451.X layer will invoke the Register::registerDest( ) method for each destination.

In normal situations, the IEEE 1451.X layer should automatically call registerDest( ) at initialization time and during a hot-swap event.

Copyright © 2007 IEEE. All rights reserved.

Parameters:
None.

Return result: Error code

11.3.14 IEEE1451Dot0::ModuleCommunication::NetComm::joinGroup

```
IDL: Args::UInt16 joinGroup(
In Args::UInt16   groupId,
In Args::UInt16   destId);
```

This method is called by the IEEE 1451.0 layer to add a destination TIM to a multicast group. If the group does not exist, it will be created by this call.

NOTE—This call does not assign TransducerChannels to an AddressGroup. That is accomplished using the AddressGroup definition command described in 7.1.2.3.

Parameters:
The “groupId” parameter specifies the group to join.

The “destId” parameter specifies the node to add to the group.

Return result: Error code

11.3.15 IEEE1451Dot0::ModuleCommunication::NetComm::leaveGroup

```
IDL: Args::UInt16 leaveGroup(
in Args::UInt16   groupId,
in Args::UInt16   destId);
```

This method is called by the IEEE 1451.0 layer to remove a destination from a multicast group. A call to close( ) for a groupId will result in all destinations being removed from the group.

Parameters:
The “groupId” parameter specifies the group to leave.

The “destId” parameter specifies the node to remove from the group.

Return result: Error code

11.3.16 IEEE1451Dot0::ModuleCommunication::NetComm::lookupDestId

```
IDL: Args::UInt16 lookupDestId (
in Args::UInt16   commId,
out Args::UInt16  destId);
```

This method is called by the IEEE 1451.0 layer to convert the commId back into the destId. This may be used on the receiving side to find out the destination ID of the sending node.

Parameters:
The “commId” parameter specifies the communication channel.

The [out] “destId” parameter specifies the destination associated with the commId.

Copyright © 2007 IEEE. All rights reserved.

Return result: Error code

11.3.17 IEEE1451Dot0::ModuleCommunication::NetComm::setRemoteConfiguration

```
IDL: Args::UInt16 setRemoteConfiguration(
in Args::UInt16          commId,
in Args::TimeDuration    timeout,
in Args::ArgumentArray   params);
```

This method is called by the IEEE 1451.0 layer to set the configuration on the remote destination. The contents of the ArgumentArray are IEEE 1451.X specific. In cases where the ArgumentArray needs to be transmitted to the remote node, the standard Encode/Decode mechanism is recommended to convert the ArgumentArray into/ from OctetArray form.

Parameters:
The “commId” parameter specifies the communications channel.

The “timeout” parameter specifies the maximum amount of time the caller is willing to block before a time-out error should be returned. A value of secs == 0, nsecs == –1 means “wait forever.”

The “params” parameter specifies the configuration parameters.

Return result: Error code

11.3.18 IEEE1451Dot0::ModuleCommunication::NetComm::getRemoteConfiguration

```
IDL: Args::UInt16 getRemoteConfiguration(
in  Args::UInt16          commId,
in  Args::TimeDuration    timeout,
out Args::ArgumentArray   params);
```

This method is called by the IEEE 1451.0 layer to get the configuration from the remote destination. The contents of the ArgumentArray are IEEE 1451.X specific. In cases where the ArgumentArray needs to be transmitted to the remote node, the standard Encode/Decode mechanism is recommended to convert the ArgumentArray into/from OctetArray form.

Parameters:
The “commId” parameter specifies the communications channel.

The “timeout” parameter specifies the maximum amount of time the caller is willing to block before a time-out error should be returned. A value of secs == 0, nsecs == –1 means “wait forever.”

The [out] “params” parameter specifies the configuration parameters.

Return result: Error code

Copyright © 2007 IEEE. All rights reserved.

11.3.19 IEEE1451Dot0::ModuleCommunication::NetComm::sendRemoteCommand

```
IDL: Args::UInt16 sendRemoteCommand(
in  Args::UInt16          commId,
in  Args::TimeDuration    timeout,
in  Args::UInt8           cmdClassId,
in  Args::UInt8           cmdFunctionId,
in  Args::ArgumentArray   inArgs,
out Args::ArgumentArray   outArgs);
```

This low-level mechanism sends an arbitrary command to the remote IEEE 1451.X layer. It is recommended that the IEEE 1451.X layer use the standard Encoder/Decoder to convert the ArgumentArrays to/from OctetArrays. This method will perform a blocking operation. The format of input and output arguments is command dependent. The caller shall make sure to use the correct data types for each input argument.

Parameters:
The “commId” specifies the communication channel.

The “cmdClassId” specifies the desired command class code. See Table 15 for details.

The “cmdFunctionId” specifies the desired command function code. See Clause 7 for details.

The “timeout” is the maximum time to wait before a time-out error. A value of secs == 0, nsecs == –1 means “wait forever.”

The “inArgs” are the input arguments in ArgumentArray form.

The [out] “outArgs” are returned output arguments.

Return result: Error code

11.4 IEEE1451Dot0::ModuleCommunication::Registration

```
IDL: interface Registration { }
```

The abstract Registration Interface is a collection of methods provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer. The methods are listed in Table 98. See the interfaces P2PRegistration and NetRegistration for additional methods.

Table 98—Registration interface methods

IEEE1451Dot0::ModuleCommunication::Registration
Args::UInt16 unRegisterModule (in Args::UInt8 moduleId);
Args::UInt16 reportModules (in Args::UInt8 maxLen, in Args::UInt8 offset, out Args::UInt8Array moduleIds);
Args::UInt16 getCommModule(in Args::UInt8 moduleId, out Comm commObject, out Args::UInt8 type, out Args:UInt8 technologyId);

11.4.1 IEEE1451Dot0::ModuleCommunication::Registration::registerModule

Two methods are used to register an IEEE 1451.X interface with the IEEE 1451.0 layer. One method, P2Pregistration (see 11.5.1), is called by devices using the P2PComm methods. The other method, NetRegistration (see 11.6.1), is used with devices using the NetComm methods. The enumerations used to identify the various interfaces are listed in Table 99.

Copyright © 2007 IEEE. All rights reserved.

Table 99—Communication module identifiers

Enumeration	Standard
0	Reserved
1	IEEE Std 1451.2-1997 using 10-wire TII
2	IEEE Std 1451.2-1997 using RS-232
3	IEEE Std 1451.3-2003
4	IEEE Std 1451.5-2007, Clause 6 [B4]
5	IEEE Std 1451.5-2007, Clause 8 [B4]
6	IEEE Std 1451.5-2007, Clause 7 [B4]
7	IEEE Std 1451.5-2007, Clause 9 [B4]
8	IEEE P1451.6 [B3]
9–254	Reserved
255	Not in conformance to any released or in development standard

11.4.2 IEEE1451Dot0::ModuleCommunication::Registration::unRegisterModule

```
IDL: Args::UInt16 unRegisterModule (in Args::UInt8 moduleId);
```

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer to de-register an interface. Its purpose is to inform the IEEE 1451.0 layer that this instance of the IEEE 1451.X interface is no longer available.

Parameters:
The “moduleId” parameter specifies which interface to de-register. The IEEE 1451.0 layer will remove all knowledge of this interface from its cache.

Return result: Error code

11.4.3 IEEE1451Dot0::ModuleCommunication::Registration::reportModules

```
IDL: Args::UInt16 reportModules (
in  Args::UInt16       maxLen,
in  Args::UInt16       offset,
out Args::UInt8Array   moduleIds);
```

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer or other layers to report the known interfaces.

Parameters:
The “maxLen” parameter indicates the maximum number of interfaces to report.

The “offset” parameter indicates the starting position when the number of modules registered is greater than maxLen. Zero is the beginning position.

The [out] “moduleIds” parameter contains the return values. These are the “moduleId” value for each available module interface. The length of this array may be less than maxLen.

Return result: Error code

Copyright © 2007 IEEE. All rights reserved.

11.4.4 IEEE1451Dot0::ModuleCommunication::Registration::getCommModule

```
IDL: Args::UInt16 getCommModule(
in  Args::UInt8      moduleId,
out Comm             commObject,
out Args::UInt8      type,
out Args::UInt8      technologyId);
```

This returns the abstract "Comm" object. By consulting the "type" parameter, a safe downcast to either the "P2PComm" or "NetComm" object may be performed.

Use extreme caution when accessing the underlying "Comm" objects as incorrect usage can compromise the IEEE 1451.0 and IEEE 1451.X layers. This method is provided to allow expansion beyond the IEEE 1451.0 architecture. The enumerations for the argument "type" are listed in Table 94.

Parameters:
The "moduleId" parameter is the desired communications module ID.

The [out] "commObject" parameter is returned to the application and is a reference to the underlying object.

The [out] "type" parameter is returned to the application in order to allow a safe downcast. Valid values are listed in Table 94.

The [out] "technologyId" specifies the underlying IEEE 1451.X technology. See Table 99 for a list of the enumerations used to identify these technologies.

Return result: Error code

11.5 IEEE1451Dot0::ModuleCommunication::P2PRegistration

```
IDL: interface P2PRegistration { }
```

The P2P Registration Interface is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer for the P2P case. The methods that make up this interface are listed in Table 100.

Table 100—P2PRegistration interface methods

IEEE1451Dot0::ModuleCommunication::P2PRegistration
Args::UInt16 registerModule(in P2PComm commInterface, in Args::UInt8 technologyId, in Args::_String name, out Args::UInt8 moduleId);
Args::UInt16 registerDestination(in Args::UInt8 moduleId, in Args::UInt16 maxPayloadSize);
Args::UInt16 unregisterDestination(in Args::UInt8 moduleId);

11.5.1 IEEE1451Dot0::ModuleCommunication::P2PRegistration::registerModule

```
IDL: Args::UInt16 registerModule(
in  P2PComm          commInterface,
in  Args::UInt8      technologyId,
in  Args::_String    name,
out Args::UInt8      moduleId);
```


Copyright © 2007 IEEE. All rights reserved.

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer during configuration for the "P2P" case. Its purpose is to inform the IEEE 1451.0 layer that the IEEE 1451.X layer is available.

Parameters:
The "commInterface" parameter specifies the P2PComm interface. See Table 94 for possible values. the IEEE 1451.0 layer will cache this information and use it to invoke appropriate methods when engaging in communication operations.

The "technologyId" specifies the underlying IEEE 1451.X technology. See Table 99 for a list of the enumerations used to identify these technologies.

The "name" parameter is a human readable name string for display purposes.

The out "moduleId" parameter is returned as a "interface identifier" that can be used in the unRegister( ) method.

Return result: Error code

11.5.2 IEEE1451Dot0::ModuleCommunication::P2PRegistration::registerDestination

```
IDL: Args::UInt16 registerDestination(
in Args::UInt8       moduleId,
in Args::UInt16      maxPayloadSize);
```

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer during configuration for the "P2P" case. Its purpose is to inform the IEEE 1451.0 layer that a valid TIM has been attached to the interface and is available.

Parameters:
The "moduleId" parameter specifies the P2PComm "interface identifier"

The "maxPayloadLen" parameter indicates the maximum payload size that will be accepted in subsequent read( ) and write( ) operations.

Return result: Error code

11.5.3 IEEE1451Dot0::ModuleCommunication::P2PRegistration::unregisterDestination

```
IDL: Args::UInt16 unregisterDestination(in Args::UInt8 moduleId);
```

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer during configuration for the "P2P" case. Its purpose is to inform the IEEE 1451.0 layer that the TIM has been detached from the interface and is no longer available.

Parameters:
The "moduleId" parameter specifies the P2PComm "interface identifier"

Return result: Error code

Copyright © 2007 IEEE. All rights reserved.

11.6 IEEE1451Dot0::ModuleCommunication::NetRegistration

```
IDL: interface NetRegistration { }
```

The NetRegistration Interface is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer for the network case. The methods required by this interface are listed in Table 101.

Table 101—NetRegistration interface methods

IEEE1451Dot0::ModuleCommunication::NetRegistration
Args::UInt16 registerModule (in NetComm commInterface, in Args::UInt8 technologyId, in Args::_String name, out Args::UInt8 moduleId);
Args::UInt16 registerDestination (in Args::UInt8 moduleId, out Args::UInt16 destId);
Args::UInt16 unRegisterDestination (in Args::UInt8 moduleId, in Args::UInt16 destId);
Args::UInt16 reportDestinations (in Args::UInt8 moduleId, in Args::UInt16 maxLen, in Args::UInt16 offset, out Args::UInt16Array destinations);
Args::UInt16 reportGroups (in Args::UInt8 moduleId, in Args::UInt16 maxLen, in Args::UInt16 offset, out Args::UInt16Array groups);
Args::UInt16 reportGroupMembers (in Args::UInt8 moduleId, in Args::UInt16 groupId, in Args::UInt16 maxLen, in Args::UInt16 offset, out Args::UInt16Array groups);

11.6.1 IEEE1451Dot0::ModuleCommunication::NetRegistration::registerModule

```
IDL: Args::UInt16 registerModule(
in  NetComm         commInterface,
in  Args::UInt8     technologyId,
in  Args::_String   name,
out Args::UInt8     moduleId);
```

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer during configuration for the "Net" case. Its purpose is to inform the IEEE 1451.0 layer that the IEEE 1451.X layer is available.

Parameters:
The "commInterface" parameter specifies the NetComm interface. See Table 94 for possible values. The IEEE 1451.0 layer will cache this information and use it to invoke appropriate methods when engaging in communication operations.

The "technologyId" specifies the underlying IEEE 1451.X technology. See Table 99 for a list of the enumerations used to identify these technologies.

The "name" parameter is a human-readable name string for display purposes.

The [out] "moduleId" parameter is returned as a "interface identifier" that can be used in the unRegister( ) method.

Return result: Error code

11.6.2 IEEE1451Dot0::ModuleCommunication::NetRegistration::registerDestination

```
IDL: Args::UInt16 registerDestination (
in  Args::UInt8      moduleId,
out Args::UInt16     destId );
```


Copyright © 2007 IEEE. All rights reserved.

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer to register a new destination. Its purpose is to inform the IEEE 1451.0 layer that the new destination is available. It is the responsibility of the IEEE 1451.X layer to detect whether a new TIM appears in the system and to register it with the IEEE 1451.0 layer.

Parameters:
The "moduleId" parameter specifies instance ID for this IEEE 1451.X interface.

The [out] "destId" parameter is returned. The "destination identifier" is assigned by the IEEE 1451.0 layer. The IEEE 1451.X layer will need to cache appropriate communication endpoint information for this "destId." The "destId" parameter will be passed back to the IEEE 1451.X layer during the open( ) or openQoS( ) call.

Return result: Error code

11.6.3 IEEE1451Dot0::ModuleCommunication::NetRegistration::unRegisterDestination

```
IDL: Args::UInt16 unRegisterDestination (
in Args::UInt8       moduleId,
in Args::UInt16      destId);
```

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer to unregister a destination. Its purpose is to inform the IEEE 1451.0 layer that the destination is no longer available. It is the responsibility of the IEEE 1451.X layer to detect whether a TIM disappears from the system and unRegister that TIM with the IEEE 1451.0 layer.

Parameters:
The "moduleId" parameter specifies the instance ID for this IEEE 1451.X interface.

The "destId" parameter is the destination identifier to unregister.

Return result: Error code

11.6.4 IEEE1451Dot0::ModuleCommunication::NetRegistration::reportDestinations

```
IDL: Args::UInt16 reportDestinations (
in  Args::UInt8          moduleId,
in  Args::UInt16         maxLen,
in  Args::UInt16         offset,
out Args::UInt16Array    destinations);
```

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer or other layers to report the known destinations associated with this IEEE 1451.X interface. Note that group destinations are not returned in this call. The maxLen and offset parameters can be used on memory-constrained systems to retrieve only a portion of the known modules.

Parameters:
The "moduleId" specifies the instance ID for this IEEE 1451.X interface.

The "maxLen" parameter indicates the maximum number of interfaces to report.

The "offset" parameter indicates the desired starting position, with zero being the first position.

The [out] "destinations" parameter provides the return values. These are the "destId" values for each known destination.

Copyright © 2007 IEEE. All rights reserved.

Return result: Error code

11.6.5 IEEE1451Dot0::ModuleCommunication::NetRegistration::reportGroups

```
IDL: Args::UInt16 reportGroups (
in  Args::UInt8         moduleId,
in  Args::UInt16        maxLen,
in  Args::UInt16        offset,
out Args::UInt16Array   groups);
```

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer or other layers to report the known groups associated with this IEEE 1451.X interface.

Parameters:
The “moduleId” specifies the instance ID for this IEEE 1451.X interface.

The “maxLen” parameter indicates the maximum number of interfaces to report.

The “offset” parameter indicates the desired starting position, with zero being the first position.

The [out] “groups” parameter provides an array for the IEEE 1451.0 layer to use to return the values. These are the “destId” values for each known group destination.

Return result: Error code

11.6.6 IEEE1451Dot0::ModuleCommunication::NetRegistration::reportGroupMembers

```
IDL: Args::UInt16 reportGroupMembers (
in  Args::UInt8         moduleId,
in  Args::UInt16        groupId,
in  Args::UInt16        maxLen,
in  Args::UInt16        offset,
out Args::UInt16Array   groups);
```

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer or other layers to report the known members in a specific “groupId” that is associated with this IEEE 1451.X interface.

Parameters:
The “moduleId” specifies the instance ID for this IEEE 1451.X interface.

The “groupId” parameter indicates the desired group ID.

The “maxLen” parameter indicates the maximum number of interfaces to report.

The “offset” parameter indicates the desired starting position, with zero being the first position.

The [out] “groups” parameter provides an array for the IEEE 1451.0 layer to use to return the values. These are the “destId” values for each known group destination.

Return result: Error code

Copyright © 2007 IEEE. All rights reserved.

11.7 IEEE1451Dot0::ModuleCommunication::Receive

IDL: `interface Receive { }`

The abstract Receive Interface does not define any generic methods. It is provided for future expansion.

11.8 IEEE1451Dot0::ModuleCommunication::P2PReceive

IDL: `interface P2PReceive { }`

The P2PReceive interface is a collection of methods provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer. The methods used to implement this interface are listed in Table 102.

Table 102—P2PReceive interface methods

IEEE1451Dot0::ModuleCommunication::P2PReceive
Args::UInt16 abort(in Args::UInt16 status);
Args::UInt16 notifyMsg(in Args::_Boolean twoWay, in Args::UInt32 payloadLen, in Args::UInt32 cacheLen, in Args::UInt16 status);
Args::UInt16 notifyRsp(in Args::UInt32 payloadLen, in Args::UInt32 cacheLen, in Args::UInt16 status);

11.8.1 IEEE1451Dot0::ModuleCommunication::P2PReceive::abort

IDL:
```
Args::UInt16 abort( in Args::UInt16 status );
```

This method is called by the IEEE 1451.X layer to abort the current operation. This will be called by the receiving node's IEEE 1451.X layer when the IEEE 1451.0 command processing has already been initiated and must be terminated.

Parameters:
The "status" parameter provides an error code on why abort( ) was called.

Return result: Error code returned back to the IEEE 1451.X layer. Typically, this return value will be ignored by the IEEE 1451.X layer.

11.8.2 IEEE1451Dot0::ModuleCommunication::P2PReceive::notifyMsg

IDL:
```
Args::UInt16 notifyMsg(
in Args::_Boolean         twoWay,
in Args::UInt32           payloadLen,
in Args::UInt32           cacheLen,
in Args::UInt16           status);
```

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer when an incoming message is available. This is invoked on the receiving node.

Parameters:
The "twoWay" parameter indicates if the command does not generate a reply. A "true" value indicates a reply is expected.

The "payloadLen" parameter indicates the total size of the payload.

Copyright © 2007 IEEE. All rights reserved.

The “cacheLen” parameter indicates the number of octets that can be immediately read.

The “status” parameter provides an error code on why notifyMsg( ) was called.

Return result: Error code returned back to the IEEE 1451.X layer. Typically, this return value will be ignored by the IEEE 1451.X layer.

11.8.3 IEEE1451Dot0::ModuleCommunication::P2PReceive::notifyRsp

```
IDL: Args::UInt16 notifyRsp(
in Args::UInt32      payloadLen,
in Args::UInt32      cacheLen,
in Args::UInt16      status);
```

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer when an incoming response message is available. This is invoked on the initiating node for two-way communications.

Parameters:
The “payloadLen” parameter indicates the total size of the payload.

The “cacheLen” parameter indicates the number of octets that can be immediately read.

The “status” parameter provides an error code on why notifyRsp( ) was called.

Return result: Error code returned back to the IEEE 1451.X layer. Typically, this return value will be ignored by the IEEE 1451.X layer.

11.9 IEEE1451Dot0::ModuleCommunication::NetReceive

```
IDL: interface NetReceive { }
```

The NetReceive interface is a collection of methods provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer. The methods used to implement this interface are listed in Table 103.

Table 103—NetReceive interface methods

IEEE1451Dot0::ModuleCommunication::NetReceive
Args::UInt16 abort(in Args::UInt16 commId, in Args::UInt16 status);
Args::UInt16 notifyMsg(in Args::UInt16 rcvCommId, in Args::_Boolean twoWay, in Args::UInt32 payloadLen, in Args::UInt32 cacheLen, in Args::UInt16 maxPayloadLen, in Args::UInt16 status);
Args::UInt16 notifyRsp(in Args::UInt16 rcvCommId, in Args::UInt16 msgId, in Args::UInt32 payloadLen, in Args::UInt32 cacheLen, in Args::UInt16 maxPayloadLen, in Args::UInt16 status);

11.9.1 IEEE1451Dot0::ModuleCommunication::NetReceive::abort

```
IDL: Args::UInt16 abort(
in Args::UInt16   commId,
in Args::UInt16   status);
```

This method is called by the IEEE 1451.X layer to abort the current operation. This will be called by the receiving node’s IEEE 1451.X layer when the IEEE 1451.0 command processing has already been initiated and must be terminated.

Copyright © 2007 IEEE. All rights reserved.

If the commId argument is zero, this call will abort all active transactions.

Parameters:
The “commId” parameter specifies the active communication channel.

The “status” parameter provides an error code on why notifyRsp( ) was called.

Return result: Error code returned back to the IEEE 1451.X layer. Typically, this return value will be ignored by the IEEE 1451.X layer.

11.9.2 IEEE1451Dot0::ModuleCommunication::NetReceive::notifyMsg

```
IDL: Args::UInt16 notifyMsg(
in Args::UInt16      rcvCommId,
in Args::_Boolean    twoWay,
in Args::UInt32      payloadLen,
in Args::UInt32      cacheLen,
in Args::UInt16      maxPayloadLen,
in Args::UInt16      status);
```

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer when an incoming message is available. This is invoked on the receiving node.

Parameters:
The “rcvCommId” parameter specifies the active communication channel. Note that the receiving node’s IEEE 1451.0 layer does not call open( ) or close( ) on this “rcvCommId.” This “rcvCommId” is managed by the IEEE 1451.X layer.

If the “twoWay” parameter is true, then a response is expected in this transaction.

The “payloadLen” parameter indicates the total size of the payload.

The “cacheLen” parameter indicates the number of octets that can be immediately read.

The “maxPayloadLen” parameter indicates the maximum payload size that will be accepted in subsequent readMsg( ) and writeRsp( ) operations.

The “status” parameter provides an error code on why notifyRsp( ) was called.

Return result: Error code returned back to the IEEE 1451.X layer. Typically, this return value will be ignored by the IEEE 1451.X layer.

11.9.3 IEEE1451Dot0::ModuleCommunication::NetReceive::notifyRsp

```
IDL: Args::UInt16 notifyRsp(
in Args::UInt16   rcvCommId,
in Args::UInt16   msgId,
in Args::UInt32   payloadLen,
in Args::UInt32   cacheLen,
in Args::UInt16   maxPayloadLen,
in Args::UInt16   status);
```

This method is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer when an incoming response message is available. This is invoked on the initiating node for two-way communications.

NOTE—The receiving node’s IEEE 1451.0 layer does not call open( ), so a “rcvCommId” is automatically provided in this call. This “rcvCommId” is managed by the IEEE 1451.X layer.

Copyright © 2007 IEEE. All rights reserved.

For "two way" communications, the receiving IEEE 1451.0 layer will invoke writeRsp( ) and eventually close( ) to end this side of the transaction.

Parameters:
The "rcvCommId" parameter specifies the active communication channel. Note that the receiving node's IEEE 1451.0 layer does not call open( ) or close( ) on this "rcvCommId." This "rcvCommId" is managed by the IEEE 1451.X layer.

The "msgId" is passed to the IEEE 1451.0 layer to associate this reply with the corresponding call to writeMsg( ).

The "payloadLen" parameter indicates the total size of the payload.

The "cacheLen" parameter indicates the number of octets that can be immediately read.

The "maxPayloadLen" parameter indicates the maximum payload size that will be accepted in subsequent readRsp( ) operation.

The "status" parameter provides an error code on why notifyRsp( ) was called.

Return result: Error code returned back to the IEEE 1451.X layer. Typically, this return value will be ignored by the IEEE 1451.X layer.

12. HTTP protocol

The HTTP is a protocol used to transfer or convey information on the World Wide Web. HTTP is a client–server protocol by which two processors can communicate over a TCP/IP connection. An HTTP server is a program that resides in a processor listening to a port for HTTP requests. An HTTP client opens a TCP/IP connection to the server via a socket, transmits a request, and then waits for a reply from the server. For this IEEE standard, the "client-–server" model, which is used to define HTTP, is analogous to the IEEE 1451.0 "NCAP-End User." The NCAP can be compared with the "server" as it serves data to the attached network, and the End-User can be compared with the "client" as the end-user receives sensor data to view from the server and sends commands and data to the server to drive actuators.

> HTTP Client Request: The HTTP client sends a request message formatted according to the rules of the HTTP standard—an HTTP Request. This message specifies the resource that the client wishes to retrieve or includes information to be provided to the server.
>
> HTTP Server Response: The server reads and interprets the request. It takes action relevant to the request and creates an HTTP response message, which it sends back to the client. The response message indicates whether the request was successful and may contain the content of the resource that the client requested, if appropriate.

HTTP defines eight methods indicating the desired action to be performed on the identified resource. These methods include GET, POST, HEAD, PUT, DELETE, TRACE, and OPTIONS. The APIs in this clause use only the GET and POST methods.

GET: Retrieve whatever information is identified by the Request-URI. This is typically how a query form works. The interaction is more like a question, such as a query, read operation, or lookup. GET can be used to retrieve data from the server. In this API, GET can be used to read and write transducer data and transducer TEDS.

POST: Used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. With HTTP this is typically how a complex data submission works. The interaction is more like an order, or the interaction changes the state of the resource in a way that the user would perceive (for example, a subscription to a service) or would

Copyright © 2007 IEEE. All rights reserved.

hold accountable for the results of the interaction. This method can be used to take a command or a commitment from the user. It is often used to pass form data to a server. In this API, POST can be used to read and write transducer data and TEDS.

12.1 IEEE 1451.0 HTTP API

The TSI is an NCAP-only API used by measurement and control applications to access the IEEE 1451.0 layer. This API contains operations to read and write TransducerChannels, read and write TEDS, and send configuration, control, and operation commands to the TIMs. The Transducer Services Interface of the IEEE 1451.0 layer contains five interfaces: TransducerAccess, TransducerManager, TimDiscovery, TEDSManager, and AppCallback. The first four interfaces are implemented by the IEEE 1451.0 layer and are called by the measurement applications. If the application desires advanced optional features, it will need to implement the "AppCallback" interface, which the IEEE 1451.0 layer will invoke.

Discovery: Methods for applications to discover available IEEE 1451.X communications modules, TIMs, and TransducerChannels are organized in this interface.

TransducerAccess: When an application desires to access transducer and actuator TransducerChannels, it will use methods on this interface.

TransducerManager: Applications that need more control over TIM access will use methods on this interface. For example to lock the TIM for exclusive use and to send arbitrary commands to the TIM.

TEDSManager: Applications use methods on this interface to read and write TEDS. This class also manages the NCAP-side TEDS cache information.

AppCallback: Applications that need advanced features need to implement this interface. For example, this allows the application to configure measurement streams and the IEEE 1451.0 layer will invoke appropriate callbacks in the application.

NOTE—There is no AppCallback API specified in this document.

The IEEE 1451.0 HTTP APIs focuses mainly on accessing transducer data and TEDS using the HTTP 1.1 protocol. Figure 23 shows that HTTP 1.1 accesses an IEEE 1451.0 NCAP on which a HTTP server is running. In Figure 23, the "S: in the TIMs means "Sensor." Similairly the "A" means "Actuator." Users can send an HTTP request to the HTTP server on the NCAP and get an HTTP response from the HTTP server. The request–response process can be described as follows:

a) A user or client sends an HTTP request to the HTTP server on the IEEE 1451.0 NCAP.

b) The HTTP server on the NCAP receives an HTTP request, processes it, and then calls the corresponding IEEE 1451.0 API.

c) The IEEE 1451.0 API calls on the IEEE 1451.X API to communicate with the IEEE 1451.X TIM and to get the results from the TIM.

d) The HTTP server on the NCAP gets the results from the IEEE 1451.0 API and then returns the HTTP response to the user.

Copyright © 2007 IEEE. All rights reserved.

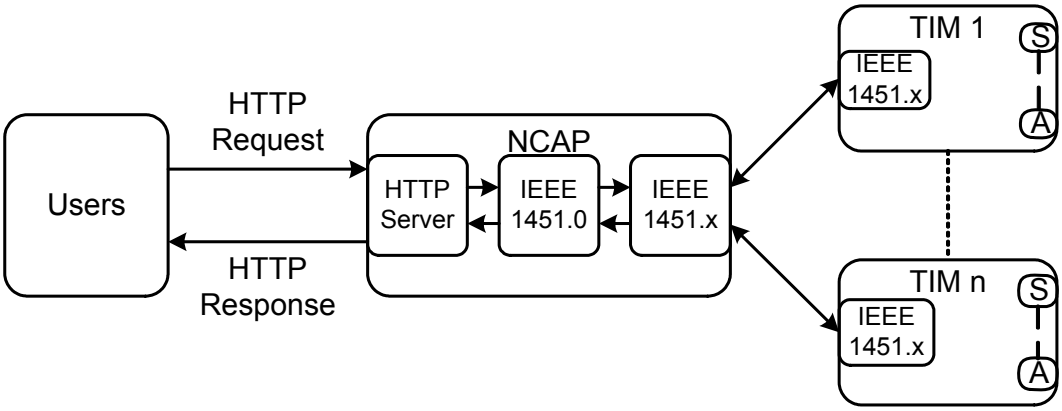


Figure 23—HTTP access to an IEEE 1451.0 NCAP

12.1.1 HTTP message format

In this clause, we describe the use of the "http" protocol to send messages from a remote client to an IEEE 1451.0 NCAP, which, for this model, acts like a server, serving transducer data to the remote client. Table 104 lists the format of the http message, Table 105 gives examples of possible arguments, and Table 106 lists the http APIs. The HTTP message from a remote client unit will be transmitted across a network connecting the remote client with the IEEE 1451.X transducer nodes. The message will adhere to the HTTP URL syntax (RFC 2616) as follows:

`http://<host>:<port>/<path>?<parameters>`

Table 104—HTTP communications message format

Field	Definition	Example
<host>:	The host portion of this statement will include the target IEEE 1451 node domain name.	<host> = "192.168.1.91"
<port>	The port number is optional and will default to port #80 if this is not specified in the request. It may be useful to select an unused port # (not port #80) that will become the main IEEE 1451 port. The downside of using a non-standard port # is that many security routers will only allow non-standard port numbers through the use of an explicit rule change.	<port>="80"
<path>	The path indicates the IEEE 1451 path including the command itself.	<path>="1451/TransducerAccess/ReadData"
<parameters>	The parameters associated with the command.	<parameters>="timId=1&channelId=2&timeout=14&samplingMode=continuous&format=text"

Table 105—Example parameters

Field	Definition	Example
timId	The identifier of selected TIM. See 10.1.2.	1
channelId	The identifier of the selected TransducerChannel of the TIM. See 10.2.1.	2
timeout	This argument specifies how long to wait to perform the reading before generating a time-out error.	14
samplingMode	This argument specifies the sampling mechanism. See 5.10.1 and 7.1.2.4.	Continuous
format	"format" is the desired return format. The options are "text," "HTML," or "xml." The contents of the string are case insensitive.	text

Copyright © 2007 IEEE. All rights reserved.

Table 106—HTTP API

API type	Name	Path
Discovery API	TIMDiscovery	1451/Discovery/TIMDiscovery
	TransducerDiscovery	1451/Discovery/TransducerDiscovery
Transducer Access API	ReadData	1451/TransducerAccess/ReadData
	StartReadData	1451/TransducerAccess/StartReadData
	MeasurementUpdate	1451/TransducerAccess/MeasurementUpdate
	WriteData	1451/TransducerAccess/WriteData
	StartWriteData	1451/TransducerAccess/StartWriteData
TEDS Manager API	ReadTeds	1451/TEDSManager/ReadTeds
	ReadRawTeds	1451/TEDSManager/ReadRawTeds
	WriteTeds	1451/TEDSManager/WriteTeds
	WriteRawTeds	1451/TEDSManager/WriteRawTeds
	UpdateTedsCache	1451/TEDSManager/UpdateTedsCache
Transducer Manager API	SendCommand	1451/TransducerManager/SendCommand
	StartCommand	1451/TransducerManager/StartCommand
	CommandComplete	1451/TransducerManager/CommandComplete
	Trigger	1451/TransducerManager/Trigger
	StartTrigger	1451/TransducerManager/StartTrigger

12.1.2 HTTP response format

The output specifications or output arguments, for example, ArgumentArray, can be included in the http request or can be included in the response to the http request. The output format can be either XML, HTML, or TEXT format.

The response format is specified in the format argument of the http message. Responses shall be formatted as specified in this section

12.1.2.1 XML response format

If XML format is specified, the response shall be formatted as specified in the XML schema associated with that command. The formatting specified in the commands relies on a type library based on the data types defined in Clause 4. A full XML schema is available at

http://grouper.ieee.org/groups/1451/0/1451HTTPAPI/.

The file name is SmartTransducerHTTPResponse.xsd.

12.1.2.2 HTML response format

If HTML response format is specified, the response shall be formatted as a valid web page in HTTP 1.1 format. The web page formatting and layout is not specified. However, all parameters that are specified responses shall use the tagged data format, with the tag matching the return response identifier specified in the return format of the command.

12.1.2.3 Text response

Text response shall be in the format where individual parameters are returned and shall be terminated with a carriage return/line feed sequence (CR/LF) (ASCII 13,10). Values shall be returned in the order specified by the semantics of the command.

Copyright © 2007 IEEE. All rights reserved.

If an array is returned as a response, each value shall be separated by a comma and the entire array shall be terminated with a carriage return/line feed sequence. A multi-dimensional array shall be returned with the right-most ordinal being indexed first. A CR/LF sequence shall be returned after each complete set of right-most indexes has been returned. This is commonly referred to as csv format.

Integers shall be returned in the format `<Sign><Value>` where `<Sign>="+" or "-"` and `<Value>` is defined as a string containing one or more of the characters between the value `"0" and "9"`.

Floating point numbers shall be returned in scientific notation in the format:
`<Sign><Leading digit>.<Mantissa>E<Sign><Exponent>`

Where `<Sign>` is as defined above, `<Leading Digit>` is a single character as defined in `<Value>` above. And `<Mantissa>` and `<Exponent>` are as defined for `<Value>` above.

Any value returned that represents an enumeration shall return its ordinal value as an integer.

Any value returned that represents a string shall return that string enclosed in quotation marks (`"<string>"`). An embedded quote mark shall be returned as a doubled quotation mark (`""`).

12.2 Discovery API

The Discovery API focuses on discovering all of the available TIMs and TransducerChannels (transducers).

12.2.1 TIMDiscovery

This API supports reporting the timIds of all TIMs connected to a given NCAP (host). This API corresponds to `Args::UInt16 reportTims()` as described in 10.1.2.

Path: 1451/Discovery/TIMDiscovery

GET: Retrieves the timIds of all TIMs connected to the NCAP (host), and returns the timIds in an XML document or ASCII format.

12.2.1.1 Input parameters

The following parameters shall be supplied with this API call:

`_String responseFormat:` Specifies the response format as defined in 12.1.2.

12.2.1.2 TIMDiscoveryHTTPResponse

The response to this API call shall contain the following parameters:

UInt16 errorCode: error information as defined in 9.3.1.2.
UInt16Array timeIds: timIds of all TIMs available to this NCAP.

Copyright © 2007 IEEE. All rights reserved.

12.2.1.3 TIMDiscovery XML response schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="TIMDiscoveryHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timIds" type="stml:UInt16Array"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.2.1.4 TransducerDiscovery

This API supports reporting channelIds of all transducer available in the specified TIM of the specified NCAP (host). This retrieves the transducer list and names for this TIM. This information is retrieved from the cached TEDS in the NCAP. This API corresponds to the interface `Args::UInt16 reportChannels()` as described in 10.1.3.

Path: 1451/Discovery/TransducerDiscovery

GET: Retrieves the channelIds of all transducers available in specified TIM of the NCAP (host), and reports the channelIds in a specified format.

12.2.1.5 Input parameters

The following parameters shall be supplied with this API call:

UInt16 timId: the timId of the specified TIM.
_String responseFormat: Specifies the response format as defined in 12.1.2.

12.2.1.6 TransducerDiscoveryHTTPResponse

The response to this API call shall contain the following parameters:

UInt16 errorCode: error information.
UInt16 timId: the timId of specified TIM.
UInt16Array channelIds: a list of the channelIds of all transducers (sensors and actuators) available in the specified TIM.
StringArray transducerNames: a list of the transducerNames of all transducer (sensors and actuators) available in the specified TIM.

12.2.1.7 TransducerDiscovery XML response schema

If the response format is "XML", the following schema shall be used for the response.

Copyright © 2007 IEEE. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType="TransducerDiscoveryHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId" type="stml:UInt16"/>
      <xs:element name="channelIds" type="stml:UInt16Array"/>
      <xs:element name="transducerNames" type="stml:StringArray"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.3 Transducer access API

The Transducer Access API focuses on reading and writing transducers (sensors and actuators) or TransducerChannels.

12.3.1 Transducer ReadData API

The Transducer read API is used to read transducer data (value).

12.3.1.1 ReadData

This API supports retrieving transducer data from a TransducerChannel with a specified channelId on the TIM specified by a timId on a specified NCAP (host).

This API is corresponding to the `Args::UInt16 readData()` as described in 10.2.6.

Path: 1451/TransducerAccess/ReadData

GET: Retrieves data from the specified TransducerChannel in the specified TIM of the NCAP (host), and returns the transducer's data in a specified format.

12.3.1.1.1 Input parameters

The following parameters shall be supplied with this API call:

UInt16 timId: the timId of the TIM containing the TransducerChannel from which the data will be read.

UInt16 channelId: the channelId of the TransducerChannel that is being read.

TimeDuration timeout: this argument specifies how long to wait to perform the reading without generating a time-out error if no response is received. A value of secs == 0, nsecs == –1 implies wait forever. Using a value of "wait forever" is extremely dangerous as it can effectively lock out a resource.

UInt8 SamplingMode: this argument specifies the triggering mechanism. See 5.10.1 and 7.1.2.4 for details.

_String responseFormat: specifies the response format as defined in 12.1.2.

Copyright © 2007 IEEE. All rights reserved.

12.3.1.1.2 ReadDataHTTPResponse

The response to this API call shall contain the following parameters:

UInt16 errorCode: error information as defined in 9.3.1.2.

UInt16 timId: the timId of the specified TIM

UInt16 channelId: the channelId of the specified TransducerChannel

ArgumentArray transducerData: this array contains the data read from the specified TransducerChannel of the selected TIM.

12.3.1.1.3 ReadData XML rsponse schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="ReadDataHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId" type="stml:UInt16"/>
      <xs:element name="channelId" type="stml:UInt16"/>
      <xs:element name="transducerData" type="stml:ArgumentArrayType"/>
      </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.3.1.2 StartReadData

This API supports starting to retrieve transducer data of specified transducer of specified TIM of specified NCAP (host). This method begins a non-blocking read of the specified TransducerChannels. This API is corresponding to the `Args::UInt16 startReadData()` as described in 10.2.8. The transducer data to be transferred are completed by calling the MeasurementUpdate API.

Path: 1451/TransducerAccess/StartReadData

GET: Starting to retrieve transducer data of transducer available in specified TIM of the NCAP (host), and returns the error code in a specified format.

12.3.1.2.1 Input parameters

The following parameters shall be supplied with this API call.

UInt16 timId: the timId of the TIM containing the TransducerChannel to be read.

UInt16 channelId: the channelId of the specified TransducerChannelChannel.

TimeInstance triggerTime: this argument specifies when to begin the read operation.

Copyright © 2007 IEEE. All rights reserved.

TimeDuration timeout: this argument specifies how long to wait to perform the reading without generating a time-out error if no response is received.. A value of secs == 0, nsecs == –1 implies wait forever. Using a value of "wait forever" is extremely dangerous as it can effectively lock out a resource.

UInt8 SamplingMode: this argument specifies the triggering mechanism. See 5.10.1 and 7.1.2.4 for details.

_String responseFormat: specifies the response format as defined in 12.1.2.

12.3.1.2.2 StartReadData HTTP response

The response to this API call shall contain the following parameters:

UInt16 errorCode: error information as defined in 9.3.1.2.

UInt16 timId: the timId of the specified TIM.

UInt16 channelId: the channelId of the specified TransducerChannel.

12.3.1.2.3 StartReadData XML response schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="StartReadDataHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId" type="stml:UInt16"/>
      <xs:element name="channelId" type="stml:UInt16"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.3.1.3 MeasurementUpdate

In this API, the non-blocking read initiated by a StartReadData call is completed, retrieving transducer data of specified transducer of specified TIM on the specified NCAP (host). This API is corresponding to the `Args::UInt16 measurementUpdate()` as described in 10.6.1.

Path: 1451/TransducerAccess/MeasurementUpdate

GET: Retrieves transducer data of transducer available in specified TIM of the NCAP (host), and returns the transducer data in a specified format.

12.3.1.3.1 Input parameters

The following parameters shall be supplied with this API call.

`UInt16 timId:` the timId of the TIM containing the TransducerChannel to be read.

`UInt16 channelId:` the channelId of the specified TransducerChannelChannel

Copyright © 2007 IEEE. All rights reserved.

`_String responseFormat:` specifies the response format.

12.3.1.3.2 Measurement Update HTTP response

The response to this API call shall contain the following parameters:

UInt16 errorCode: error information as defined in 9.3.1.2.

UInt16 timId: the timId of the specified TIM.

UInt16 channelId: the channelId of the specified TransducerChannel.

ArgumentArray transducerData: this array contains the data read from the specified TransducerChannel of the selected TIM.

12.3.1.3.3 Measurement Update XML response schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="MeasurementUpdateHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId" type="stml:UInt16"/>
      <xs:element name="transducerId" type="stml:UInt16"/>
      <xs:element name="transducerData" type="stml:ArgumentArrayType"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.3.2 Transducer WriteData API

The Transducer write API is used to write TransducerChannel data (value).

12.3.2.1 WriteData

In this API, writing transducer data of specified transducer of specified TIM of specified NCAP (host). This method performs a blocking write of the specified TransducerChannels or transducers. This API is corresponding to the `Args::UInt16 writeData()` as described in 10.2.7.

Path: 1451/TransducerAccess/WriteData

POST: Writes transducer data to the designated TransducerChannel in the specified TIM connected to the NCAP (host).

Copyright © 2007 IEEE. All rights reserved.

12.3.2.1.1 Input parameters

The following parameters shall be supplied with this API call:

`UInt16 timId:` the timId of the TIM containing the TransducerChannel to be written.

`UInt16 channelId:` the channelId of specified TransducerChannel.

`TimeDuration timeout:` this argument specifies how long to wait after writing the data before generating a time-out error if the return is not received. A value of secs == 0, nsecs == –1 implies wait forever. Using a value of "wait forever" is extremely dangerous as it can effectively lock out a resource.

`UInt8 SamplingMode:` this argument specifies the sampling mechanism. See 5.10.1 and 7.1.2.4 for details.

`ArgumentArray transducerData:` this array contains the data to be written to the specified TransducerChannel of the selected TIM.

`_String responseFormat:` specifies the response format as defined in 12.1.2.

12.3.2.1.2 WriteData HTTP response

The response to this API call shall contain the following parameters.

`UInt16 errorCode`: error information as defined in 9.3.1.2.

`UInt16 timId`: the timId of the specified TIM.

`UInt16 channelId`: the channelId of the specified TransducerChannel.

12.3.2.1.3 WriteData XML response schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="WriteDataHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId" type="stml:UInt16Array"/>
      <xs:element name="channelId" type="stml:UInt16"/>  </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.3.2.2 StartWriteData

In this API, writing transducer data for the specified transducer of the specified TIM on the specified NCAP (host). This method performs a non-blocking write of the specified TransducerChannel. The user is responsible for determining when the command completes by sending a SendCommand call (12.5.1**)** with the ReadStatusEventRegister command specified **(7.1.1.8)**, and checking for the DataProcessed bit

Copyright © 2007 IEEE. All rights reserved.

(5.13.10) to be asserted. This API corresponds to the `Args::UInt16 startWriteData()` as described in 10.2.9**.**

Path: 1451/TransducerAccess/StartWriteData

POST: Writes transducer data to the designated TransducerChannel in the specified TIM connected to the NCAP (host), and returns result in a specified format.

12.3.2.2.1 Input parameters

The following parameters shall be supplied with this API call.

`UInt16 timId:` the timId of the TIM containing the TransducerChannel to be read.

`UInt16 channelId:` the channelId of specified transducer.

`TimeInstance triggerTime:` This argument specifies when to begin the write operation.

`TimeDuration timeout:` this argument specifies how long to wait after writing the data before generating a time-out error if no response is received.. A value of secs == 0, nsecs == –1 implies wait forever. Using a value of "wait forever" is extremely dangerous as it can effectively lock out a resource.

`UInt8 SamplingMode:` this argument specifies the triggering mechanism. See 5.10.1 and 7.1.2.4 for details.

`ArgumentArray transducerData:` this array contains the data read from the specified TransducerChannel of the selected TIM.

`_String responseFormat:` specifies the response format as defined in 12.1.2.

12.3.2.2.2 StartWriteData HTTP response

The response to this API call shall contain the following parameters:

`UInt16 errorCode:` error information as defined in 9.3.1.2.

`UInt16 timId:` the timId of the specified TIM.

`UInt16 channelId:` the channelId of the specified TransducerChannel.

12.3.2.2.3 StartWriteData XML response schema

If the response format is "XML", the following schema shall be used for the response.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="StartWriteDataHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId" type="stml:UInt16Array"/>
      <xs:element name="channelId" type="stml:UInt16"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```


Copyright © 2007 IEEE. All rights reserved.

12.4 TEDS Manager API

The TEDS Access API includes ReadTEDS for reading TEDS and WriteTEDS for writing TEDS.

12.4.1 ReadTeds

This API supports retrieving TEDS data associated with a specified TransducerChannel or TIM from a specified NCAP (host). This method will read the desired TEDS block from the TEDS Cache. If the TEDS is not available from the cache, it will read the TEDS from the TIM. This API is corresponding to the `Args::UInt16 readTeds()` as described in 10.4.1.

Path: 1451/TEDSManager/ReadTeds

GET: Retrieves a TEDS from the specified TransducerChannel or TIM on the NCAP (host), and returns the result in a specified format.

12.4.1.1 Input parameters

The following parameters shall be supplied with this API call.

`UInt16 timId`: the timId of the TIM containing the TransducerChannel to be read.

`UInt16 channelId`: the channelId of specified TransducerChannel. This argument will be zero if a TEDS associated with the entire TIM is being accessed.

`TimeDuration` timeout: this argument specifies how long to wait to perform the reading without generating a time-out error if no response is received. A value of secs == 0, nsecs == –1 implies wait forever. Using a value of "wait forever" is extremely dangerous as it can effectively lock out a resource.

`UInt8 TEDSType`: this argument specifies the TEDSType as listed in Table 17 where it is called the TEDS Access Code.

`_String responseFormat`: specifies the response format as defined in 12.1.2.

12.4.1.2 ReadTEDS HTTP response

The response to this API call shall contain the following parameters:

`UInt16 errorCode`: error information as defined in 9.3.1.2.

`UInt16 timId`: the timId of the specified TIM.

`UInt16 channelId`: the channelId of the specified TransducerChannel.

`UInt8 tedsType`: this argument specifies the tedsType as listed in Table 17 where it is called the TEDS Access Code.

`ArgumentArray teds`: this array contains the data read from the specified TEDS.

Copyright © 2007 IEEE. All rights reserved.

12.4.1.3 ReadTEDS XML response schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="ReadTEDSHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId"     type="stml:UInt16Array"/>
      <xs:element name="channelId" type="stml:UInt16"/>
      <xs:element name="tedsType"  type="stml:UInt8"/>
      <xs:element name="teds"      type="stml:ArgumentArrayType"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.4.2 ReadRawTeds

This API supports retrieving rawTEDS data from a selected TransducerChannel or from a TIM on a specified NCAP (host). This method will read the desired TEDS block from the TIM bypassing the TEDS Cache in the NCAP. The TEDS cache is not updated. This API is corresponding to the `Args::UInt16 readRawTeds()` as described in 10.4.3:

For the purposes of this API, all TEDS are binary structures. In order to encode these structures in a specified format, it is necessary to encode them as text. To accomplish this, all TEDS contents shall be encoded using the Base64 encoding described in 6.8 of RFC 2045 [B8].

Path: 1451/TEDSManager/ReadRawTeds

GET: Retrieves rawTEDS of TransducerChannel available in specified TIM of the NCAP (host), and displays the result in a specified format.

12.4.2.1 Input parameters

The following parameters shall be supplied with this API call:

`UInt16 timId`: the timId of the TIM containing the TransducerChannel to be read.

`UInt16 channelId`: the channelId of specified TransducerChannel. This argument will be zero if a TEDS associated with the entire TIM is being accessed.

`TimeDuration timeout`: this argument specifies how long to wait to perform the reading without generating a time-out error if no response is received. A value of secs == 0, nsecs == –1 implies wait forever. Using a value of "wait forever" is extremely dangerous as it can effectively lock out a resource.

`UInt8 TEDSType`: This argument specifies the TEDSType as listed in Table 17 where it is called TEDS Access Code.

`_String responseFormat`: specifies the response format.

Copyright © 2007 IEEE. All rights reserved.

12.4.2.2 ReadRawTeds HTTP response

The response to this API call shall contain the following parameters.

`UInt16 errorCode`: error information as defined in 9.3.1.2.

`UInt16 timId`: the timId of the specified TIM.

`UInt16 channelId`: the channelId of the specified TransducerChannel.

`UInt8 tedsType`: this argument specifies the tedsType as listed in Table 17 where it is called TEDS Access Code.

`OctetArray TEDS`: Raw TEDS data of specified TransducerChannel or TIM encoded as Base64.

12.4.2.3 ReadRawTEDS XML response schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="ReadRawTEDSHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId"     type="stml:UInt16Array"/>
      <xs:element name="channelId" type="stml:UInt16"/>
      <xs:element name="tedsType"  type="stml:UInt8"/>
      <xs:element name="teds"      type="stml:ArgumentArrayType"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.4.3 WriteTeds

This API is used when writing TEDS to a specified TransducerChannel or TIM of the specified NCAP (host). This method will write the desired TEDS block to the TIM. The TEDS cache is also updated if the write succeeds. The provided TEDS information is encoded in an ArgumentArray. This API corresponds to the `Args::UInt16 writeTeds()` as described in 10.4.2:

Path: 1451/TEDSManager/WriteTeds

POST: Writes the TEDS data to the specified TransducerChannel or TIM of the specified NCAP (host), and displays the result in the specified format.

12.4.3.1 Input parameters

The following parameters shall be supplied with this API call:

`UInt16 timId`: the timId of the TIM containing the TEDS to be written.

`UInt16 channelId`: the channelId of specified TransducerChannel. This argument will be zero if a TEDS associated with the entire TIM is being accessed.

Copyright © 2007 IEEE. All rights reserved.

`TimeDuration timeout`: this argument specifies how long to wait after writing the TEDS before generating a time-out error if no response is received.. A value of secs == 0, nsecs == –1 implies wait forever. Using a value of "wait forever" is extremely dangerous as it can effectively lock out a resource.

`UInt8 TEDSType`: this argument specifies the TEDSType as listed in Table 17 where it is called TEDS Access Code.

`ArgumentArray TEDS`: TEDS data of specified TransducerChannel or TIM.

`_String responseFormat`: specifies the response format as defined in 12.1.2.

12.4.3.2 WriteTEDS HTTP response

The response to this API call shall contain the following parameters:

`UInt16 errorCode`: error information as defined in 9.3.1.2.

`UInt16 timId`: the timId of the TIM containing the TEDS to be written.

`UInt16 channelId`: the channelId of specified TransducerChannel. This argument will be zero if a TEDS associated with the entire TIM is being accessed.

`UInt8 tedsType`: this argument specifies the tedsType as listed in Table 17 where it is called the TEDS Access Code.

12.4.3.3 WriteTEDS XML response schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="WriteTEDSHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId"     type="stml:UInt16Array"/>
      <xs:element name="tedsId"    type="stml:UInt16"/>
      <xs:element name="tedsType"  type="stml:UInt8"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.4.4 WriteRawTeds

In this API, writing rawTEDS data of specified transducer of specified TIM of specified NCAP (host). This method will write the desired TEDS block to the TIM bypassing the TEDS cache. The provided TEDS information is encoded in "tuple" form in an OctetArray. This API is corresponding to the `Args::UInt16 writeRawTeds()` as described in 10.4.4:

For the purposes of this API, all TEDS are binary structures. In order to encode these structures in a specified format, it is necessary to encode them as text. To accomplish this, all TEDS contents shall be encoded using the Base64 encoding described in 6.8 of RFC 2045.

Path: 1451/TEDSManager/WriteRawTeds

Copyright © 2007 IEEE. All rights reserved.

POST: Writes transducer rawTEDS data of transducer available in specified TIM of the NCAP (host), and displays the result in a specified format.

12.4.4.1 Input parameters

The following parameters shall be supplied with this API call:

`UInt16 timId:` the timId of the specified TIM

`UInt16 channelId`: the channelId of the specified TransducerChannel or "0" if the TEDS applies to the entire TIM.

`TimeDuration timeout`: specifies how long to wait after writing the TEDS before generating a time-out error if no response is received. A value of secs == 0, nsecs == –1 implies wait forever. Using a value of "wait forever" is extremely dangerous as it can effectively lock out a resource.

`UInt8 tedsType`: specifies the tedsType as listed in Table 17 where it is called the TEDS Access Code.

`OctetArray rawTEDS`: Raw TEDS data of specified TransducerChannel or of specified TIM.

`_String responseFormat`: Specifies the response format as defined in 12.1.2.

12.4.4.2 WriteRawTEDS HTTP response

The response to this API call shall contain the following parameters:

`UInt16 errorCode`: error information as defined in 9.3.1.2.

`UInt16 timId`: the timId of the specified TIM.

`UInt16 channelId`: the channelId of the specified TransducerChannel.

`UInt8 tedsType`: specifies the tedsType as listed in Table 17 where it is called the TEDS Access Code.

12.4.4.3 WriteRawTEDS XML response schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="WriteRawTEDSHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId"     type="stml:UInt16Array"/>
      <xs:element name="channelId" type="stml:UInt16"/>
      <xs:element name="tedsType"  type="stml:UInt8"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```


Copyright © 2007 IEEE. All rights reserved.

12.4.5 UpdateTedsCache

In this API, updating the TEDS cache of the specified TransducerChannel or TIM of specified NCAP (host). This API will update the TEDS cache. The TEDS checksum will be read from the TIM and compared with the cached TEDS checksum. If the checksums differ, the TEDS will be read from the TIM and stored in the cache. This API corresponds to the Args::UInt16 updateTedsCache as described in 10.4.5:

Path: 1451/TedsManager/UpdateTedsCache

GET: Updates the TEDS cache of specified transducer of specified TIM of specified NCAP (host), and displays the results in a specified format.

12.4.5.1 Input parameters

The following parameters shall be supplied with this API call:

`UInt16 timId`: a timId of specified TIM.

`UInt16 channelId`: a channelId of specified transducer or "0" means reading TIM TEDS.

`TimeDuration timeout`: specifies how long to wait to perform the reading w/o generating a time-out error.

`UInt8 tedsType`: specifies the tedsType as listed in Table 17 where it is called the TEDS Access Code.

`_String responseFormat`: specifies the response format as defined in 12.1.2.

12.4.5.2 UpdateTedsCache HTTP response

The response to this API call shall contain the following parameters:

`UInt16 errorCode`: error information.

`UInt16 timId`: a timId of the specified TIM.

`UInt16 channelId`: the channelId of the specified TransducerChannel.

`UInt8 tedsType`: specifies the tedsType as listed in Table 17 where it is called the TEDS Access Code.

12.4.5.3 UpdateTEDS XML response schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="UpdateTEDSHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId"     type="stml:UInt16Array"/>
      <xs:element name="channelId" type="stml:UInt16"/>
```


Copyright © 2007 IEEE. All rights reserved.

```
      <xs:element name="tedsType"  type="stml:UInt8"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.5 Transducer Manager API

This API contains four APIs. The SendCommand and StartCommand APIs allow the system to send commands directly to a TransducerChannel or TIM. Trigger and StartTrigger APIs are used to send triggers to a TransducerChannel or group of TransducerChannels on the same NCAP.

12.5.1 SendCommand

This method will perform a blocking operation. The format of input and output arguments are command dependent. The caller must make sure to use the correct data types for each input argument. This API is corresponding to the `Args::UInt16 sendCommand()` as described in 10.3.5:

Path: 1451/TransducerManager/SendCommand

POST: Sends a command to a TransducerChannel or TIM of the NCAP (host), and returns the result in a specified format.

12.5.1.1 Input parameters

The following parameters shall be supplied with this API call:

UInt16 timId: the timId of the specified TIM.

UInt16 channelId: the channelId of the specified TransducerChannel.

TimeDuration timeout: this argument specifies how long to wait after writing the TEDS before generating a time-out error if no response is received.. A value of secs == 0, nsecs == –1 implies wait forever. Using a value of "wait forever" is extremely dangerous as it can effectively lock out a resource.

UInt8 cmdClassId: specifies the desired command class code as described in 7.1 and listed in Table 15.

UInt8 cmdFunctionId: specifies the desired command function code. cmdFunctionIds are described with the description of each command in Clause 7 and will be listed in the Commands TEDS for non-standard commands.

ArgumentArray inArgs: the input arguments in ArgumentArray form.

_String responseFormat: specifies the response format as defined in 12.1.2.

12.5.1.2 SendCommand HTTP response

The response to this API call shall contain the following parameters:

UInt16 errorCode: error information as defined in 9.3.1.2.

UInt16 timId: the timId of the specified TIM.

UInt16 channelId: the channelId of the specified TransducerChannel.

ArgumentArray outArgs: returned output arguments.

Copyright © 2007 IEEE. All rights reserved.

12.5.1.3 SendCommand XML response schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="SendCommandHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId"     type="stml:UInt16Array"/>
      <xs:element name="channelId" type="stml:UInt16"/>
      <xs:element name="outArgs"   type="stml:ArgumentArrayType"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.5.2 StartCommand

This method starts a non-blocking operation. The format of input arguments are command dependent. The caller must make sure to use the correct data types for each input argument. This API is corresponding to the `Args::UInt16 startCommand()` as described in 10.3.7. The returned ArgumentArray is completed by calling CompleteCommand API.

Path: 1451/TransducerManager/StartCommand

GET: Sends a command to a TransducerChannel or TIM of the NCAP (host), and returns the result in a specified format.

12.5.2.1 Input parameters

The following parameters shall be supplied with this API call.

UInt16 timId: the timId of the specified TIM

UInt16 channelId: the channelId of the specified TransducerChannel or "0" addressed to the TIM.

TimeInstance triggerTime: specifies when to begin the operation.

TimeDuration Timeout: the maximum time to wait before a time-out error.

UInt8 cmdClassId: specifies the desired command class code as described in 7.1 and listed in Table 15.

UInt8 cmdFunctionId: specifies the desired command function code. cmdFunctionIds are described with the description of each command in clause 7 and will be listed in the Commands TEDS for non-standard commands.

ArgumentArray inArgs: the input arguments in ArgumentArray form. These are command dependent.

_String responseFormat: 12.1.2 specifies the response format.

12.5.2.2 StartCommand HTTP response

The response to this API call shall contain the following parameters:

Copyright © 2007 IEEE. All rights reserved.

UInt16 errorCode: error information as defined in 9.3.1.2 from the non-blocking command complete operation.

UInt16 timId: the timId of the specified TIM.

UInt16 channelId: the channelId of the specified TransducerChannel.

12.5.2.3 StartCommand XML response schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="StartCommandHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId"     type="stml:UInt16Array"/>
      <xs:element name="channelId" type="stml:UInt16"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.5.3 CommandComplete

This method completes a non-blocking StartCommand operation. The format of input arguments is command dependent. The caller must make sure to use the correct data types for each input argument. This API is corresponding to dArgs::UInt16 commandComplete()as described in 11.6.4:

Path: 1451/TransducerManager/CommandComplete

GET: Retrieves result of transducer available in specified TIM of the NCAP (host), and returns the result from the StartCommand specified in 12.5.2 in the specified format.

12.5.3.1 Input parameters

The following parameters shall be supplied with this API call:

UInt16 timId: the timId of the specified TIM.

UInt16 channelId: the channelId of the specified TransducerChannel or "0" addressed to the TIM.

_String responseFormat: specifies the response format.

12.5.3.2 CommandComplete HTTP response

The response to this API call shall contain the following parameters:

UInt16 errorCode: error information as defined in 9.3.1.2 from the non-blocking command complete operation.

UInt16 timId: the timId of the specified TIM.

Copyright © 2007 IEEE. All rights reserved.

UInt16 channelId: the channelId of the specified TransducerChannel.

ArgumentArray outArgs: the returned ArgumentArray. This information is specific to each command.

12.5.3.3 CommandComplete XML response schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="CommandCompleteHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode"    type="stml:UInt16"/>
      <xs:element name="timId"        type="stml:UInt16Array"/>
      <xs:element name="transducerId" type="stml:UInt16"/>
      <xs:element name="outArgs"      type="stml:ArgumentArrayType"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.5.4 Trigger

This method performs a blocking trigger on the specified TransducerChannel or group of TransducerChannels. This API is corresponding to the Args::UInt16 trigger()as described in 10.3.9:

Path: 1451/TransducerManager/Trigger

POST: trigger the TransducerChannel or TransducerChannels attached to a particular NCAP (host).

12.5.4.1 Input parameters

The following parameters shall be supplied with this API call:

`Int16 timId:` the timId of the specified TIM.

`UInt16 channelId:` the channelId of the specified TransducerChannel or "0" if read TIM TEDS.

`TimeInstance triggerTime:` specifies when to begin the operation.

`TimeDuration Timeout:` the maximum time to wait before a time-out error.

`UInt16 SamplingMode:` specifies the sampling mode for the TransducerChannel(s). See 5.11 for details.

`_String responseFormat:` specifies the response format as defined in 12.1.2.

12.5.4.2 Trigger HTTP response

The response to this API call shall contain the following parameters:

`UInt16 errorCode:` error information as defined in 9.3.1.2.

`UInt16 timId:` the timId of the specified TIM.

Copyright © 2007 IEEE. All rights reserved.

`UInt16 channelId:` the channelId of the specified TransducerChannel or "0" if read TIM TEDS.

12.5.4.3 Trigger XML response schema

If the response format is "XML", the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="TriggerHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId"     type="stml:UInt16Array"/>
      <xs:element name="channelId" type="stml:UInt16"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

12.5.5 StartTrigger

This method performs a non-blocking trigger on the specified TransducerChannel or TransducerChannels. The user is responsible for determining when the command completes by sending a SendCommand call (12.5.1**)** with the ReadStatusEventRegister command specified (7.1.1.8**)**, and checking for the DataProcessed bit (5.13.10) to be asserted. This API is corresponding to the `Args::UInt16 startTrigger()` as described in 10.3.10.

Path: 1451/TransducerManager/StartTrigger

POST: Starts a trigger to the transducer available in the specified TIM of the NCAP (host), and displays the result in a specified format.

12.5.5.1 Input parameters

The following parameters shall be supplied with this API call:

`UInt16 timId:` the timId of the specified TIM.

`UInt16 channelId:` the channelId of the specified TransducerChannel or "0" if read TIM TEDS.

`TimeInstance triggerTime:` specifies when to begin the operation.

`TimeDuration Timeout:` is the maximum time to wait before a time-out error.

`UInt16 SamplingMode:` specifies the trigger mode. See 5.11 for details.

`_String responseFormat:` specifies the response format as defined in 12.1.2.

12.5.5.2 StartTriggerResponse

The response to this API call shall contain the following parameters:

`UInt16 errorCode:` error information as defined in 9.3.1.2.

Copyright © 2007 IEEE. All rights reserved.

`Int16 timId:` the timId of the specified TIM.

`UInt16 channelId:` the channelId of the specified TransducerChannel or “0” if read TIM TEDS.

12.5.5.3 StartTrigger XML response schema

If the response format is “XML”, the following schema shall be used for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
<xs:complexType name="StartTriggerHTTPResponse">
   <xs:sequence>
      <xs:element name="errorCode" type="stml:UInt16"/>
      <xs:element name="timId"     type="stml:UInt16Array"/>
      <xs:element name="channelId" type="stml:UInt16"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```


Copyright © 2007 IEEE. All rights reserved.

Annex A

(informative)

Bibliography

[B1] Hamilton, B., "A compact representation of physical units," Hewlett-Packard Company, Palo Alto, CA, Hewlett-Packard Laboratories Technical Report HPL-96-61, 1995.[11]

[B2] IEEE 100, *The Authoritative Dictionary of IEEE Standards Terms*, Seventh Edition, New York, Institute of Electrical and Electronics Engineers, Inc.

[B3] IEEE P1451.6, Draft Standard for a Smart Transducer Interface for Sensors and Actuators—A High-Speed CANopen-Based Transducer Network Interface for Intrinsically Safe and Non-Intrinsically Safe Applications.[12]

[B4] IEEE Std 1451.5-2007, IEEE Standard for a Smart Transducer Interface for Sensors and Actuators—Wireless Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats.[13,14]

[B5] ISO 8601, Data Elements and Interchange Formats—Information Interchange—Representation of Dates and Times.[15]

[B6] ISO/IEC 7498, Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model.[16]

[11] Available from Aligent Technical Publications Department, 1501 Page Mill Road, Mail Stop 2L, Palo Alto, CA 94304, USA.
[12] This IEEE standards project was not approved by the IEEE-SA Standards Board at the time this publication went to press. For information about obtaining a draft, contact the IEEE.
[13] IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).
[14] The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.
[15] ISO publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/ Suisse (http://www.iso.ch/). ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).
[16] ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (http://www.iso.ch/). ISO/IEC publications are also available in the United States from Global Engineering Documents, [15] Inverness Way East, Englewood, CO 80112, USA (http://global.ihs.com/). Electronic copies are available in the United States from the American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).

Copyright © 2007 IEEE. All rights reserved.

[B7] OGC™ Recommendation Paper, Version: 1.0, "URNs of definitions in ogc namespace," OGC document 05-010, 2005.

[B8] RFC 2045, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.[17]

[B9] Taylor, B. N., Ed., *The International System of Units (SI)*, National Institute of Standards and Technology, Special Publication 330. Washington, D.C.: U.S. Government Printing Office, August 1991.

[B10] Taylor, B. N. and Kuyatt, C. E., "Guidelines for evaluating and expressing the uncertainty of NIST measurement results," NIST Technical Note 1297, National Institute of Standards Technology, Gaithersburg, MD, 1994 edition.

[17] RFC publications are available from Global Engineering Documents, 15 Inverness Way East, Englewood, CO 80112, USA (http://global.ihs.com/).

Copyright © 2007 IEEE. All rights reserved.

Annex B

(informative)

Guidance to Transducer Services Interface

Measurement and control applications interact with the IEEE 1451.0 layer by the Transducer Services Interface. This annex gives examples of the use of this interface.

The simplest and most common measurement application will be reading a TransducerChannel's value. If we assume that the TransducerChannel is being operated in the immediate operation sampling mode, the TransducerChannel will automatically trigger the measurement and return the result. Hence, this is a polled measurement.

The sequence diagram in Figure B.1 illustrates the flow. The application is on the left, and time runs down the diagram. IEEE 1451.0 processing details below the public API are not illustrated:

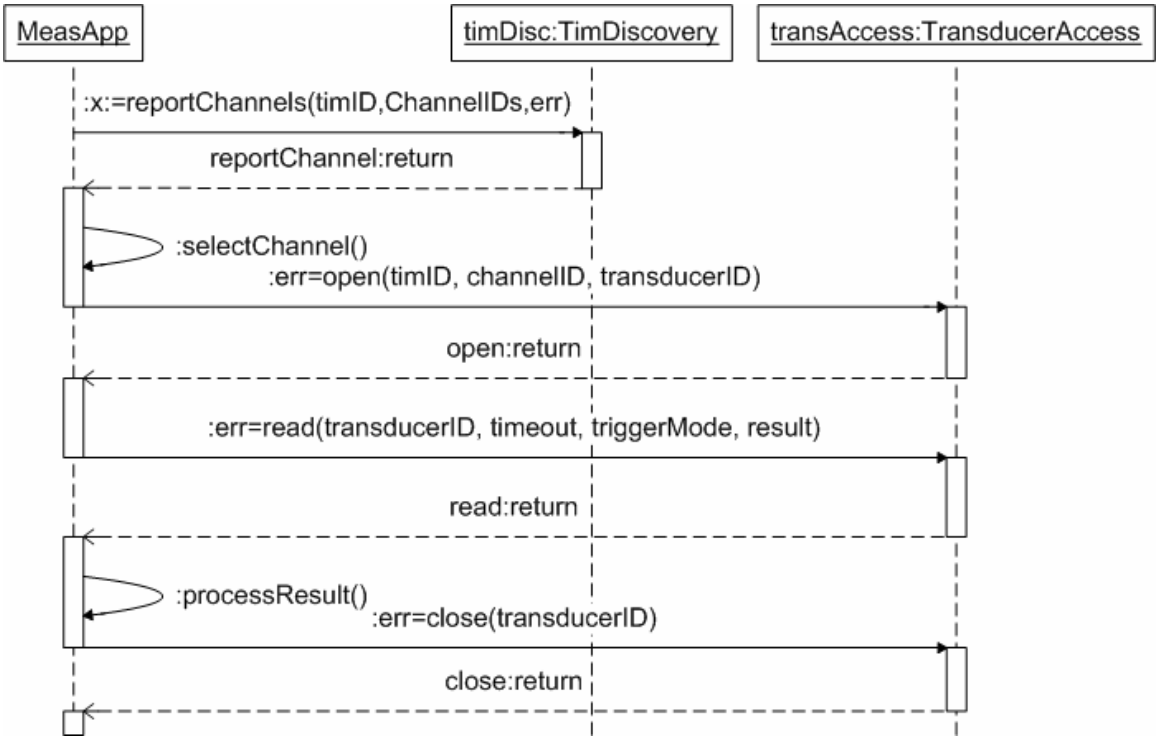


Figure B.1—Simple polled measurement example

The application (labeled MeasApp object) accesses methods on the TimDiscovery object to discover available TIMs and TransducerChannels.

Once channel selection has been made, the open( ) call is invoked on the TransducerAccess object. This will return a valid "transducerID" that will be used in subsequent calls.

Copyright © 2007 IEEE. All rights reserved.

When the application is ready to make a measurement, it invokes the read( ) call. The new measurement value will be returned in the "result" output parameter. The result is an ArgumentArray object, and the application will use the get( ) method to retrieve measurement attributes of interest. For example, the "value" attribute has the measurement's value, whereas the "timestamp" attribute contains the time when the measurement was made.

The application can control what attributes are returned in the ArgumentArray by using the configureAttributes( ) call on the TransducerManager object (not shown). For example, the "units," "accuracy," "name," and "id" attributes may be enabled. This will force the read( ) call to return an ArgumentArray with these attributes for more complete "self describing" data.

The application can invoke read( ) as often as necessary to acquire multiple readings. When the application is done, the close( ) call is invoked to free resources.

The above example illustrates the simplest case where the following conditions hold:

A single TransducerChannel is being accessed.

The application blocks until the result is returned. The maximum amount of time the application is willing to wait is specified by the "timeout" parameter.

The application specifies the desired sampling mode. In this example, the immediate operation sampling mode is assumed that will cause the TransducerChannel to make a measurement.

Methods are available to perform more elaborate measurement sequences.

B.1 Non-blocking Read example

In many measurement and control applications, measurements are requested or scheduled by one call. Later, when the measurement process has completed, the results are delivered back to the application via a callback. This non-blocking example is illustrated in Figure B.2.

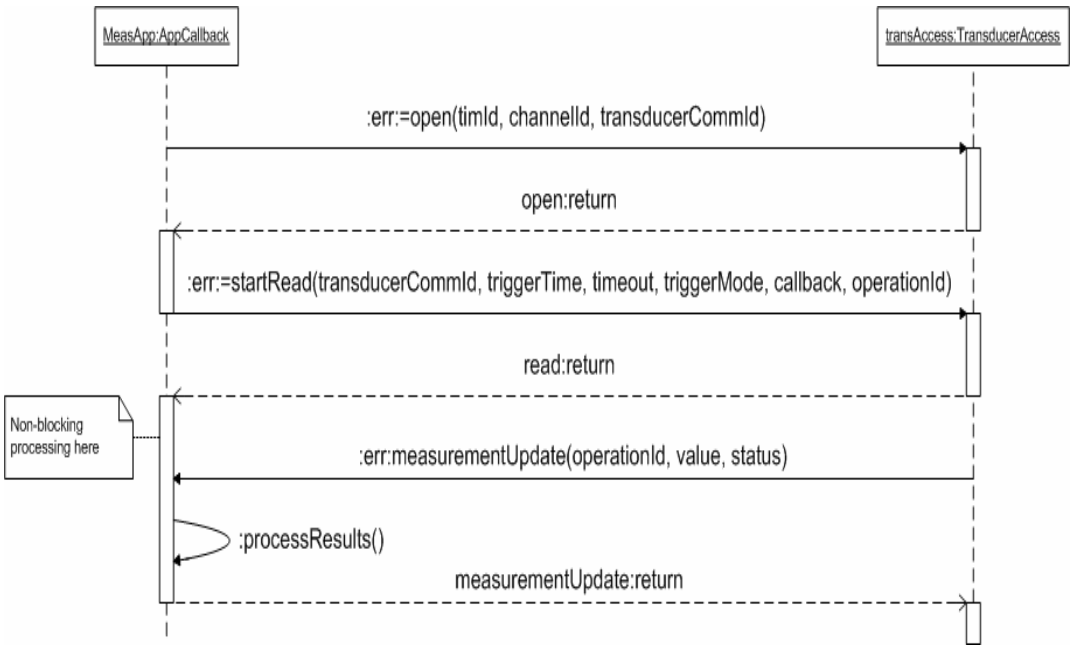


Figure B.2—Non-blocking Read example

Copyright © 2007 IEEE. All rights reserved.

In the non-blocking example, the application implements the AppCallback interface. During the startRead( ) call, this interface is passed in as the "callback" parameter. The desired trigger time and time-out are also provided.

When the measurement completes, the measurementUpdate( ) callback method is invoked to pass results back to the applications.

B.2 Generic SendCommand( ) mechanism

As discussed in Clause 7, many commands are defined in the IEEE 1451.0 layer. In many cases, specific "convenience" methods are provided via the APIs to make it easy for the application to perform useful work, for example, reading a TransducerChannel or TEDS. Behind these "convenience" methods, a generic sendCommand( ) mechanism is used. To access features that are not exposed via "convenience" methods, the sendCommand( ) method is available and used as follows:

An open( ) call is made to get a "transducerId" for the desired TIM and TransducerChannel.
A "commandId" is selected from all possible commands as enumerated in Clause 7. This UInt16 number contains the command "class" as the upper 8 bits and the command "function" code as the lower 8 bits.
All command specific input arguments are packaged into the "inArgs" ArgumentArray. The position and types of arguments in the ArgumentArray shall match what is required for the command in question.
The sendCommand( ) call is invoked.
Output parameters are returned to the caller via the "outArgs" ArgumentArray. Again, the position and types of arguments in this ArgumentArray will match what is returned by the command in question.

Figure B.3 illustrates the use of the sendCommand( ) mechanism.

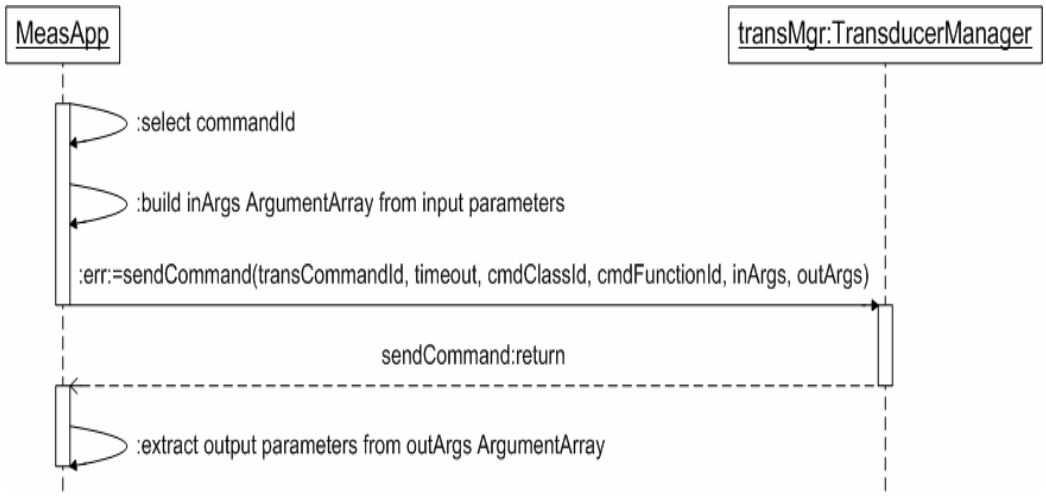


Figure B.3—SendCommand sequence diagram

A non-blocking version of this command is also available. See startCommand( ) and the commandComplete( ) callback.

A non-blocking version of this command is also available. See startCommand( ) and the commandComplete( ) callback.

Copyright © 2007 IEEE. All rights reserved.

B.3 IEEE 1451.0 processing details

To illustrate the type of processing that occurs behind the read( ) call when making a measurement, the sequence diagram in Figure B.4 is presented. Because the internal details of the IEEE 1451.0 layer are not part of the this standard, this diagram is only an aid to explain one way that the IEEE 1451.0 processing could be structured.

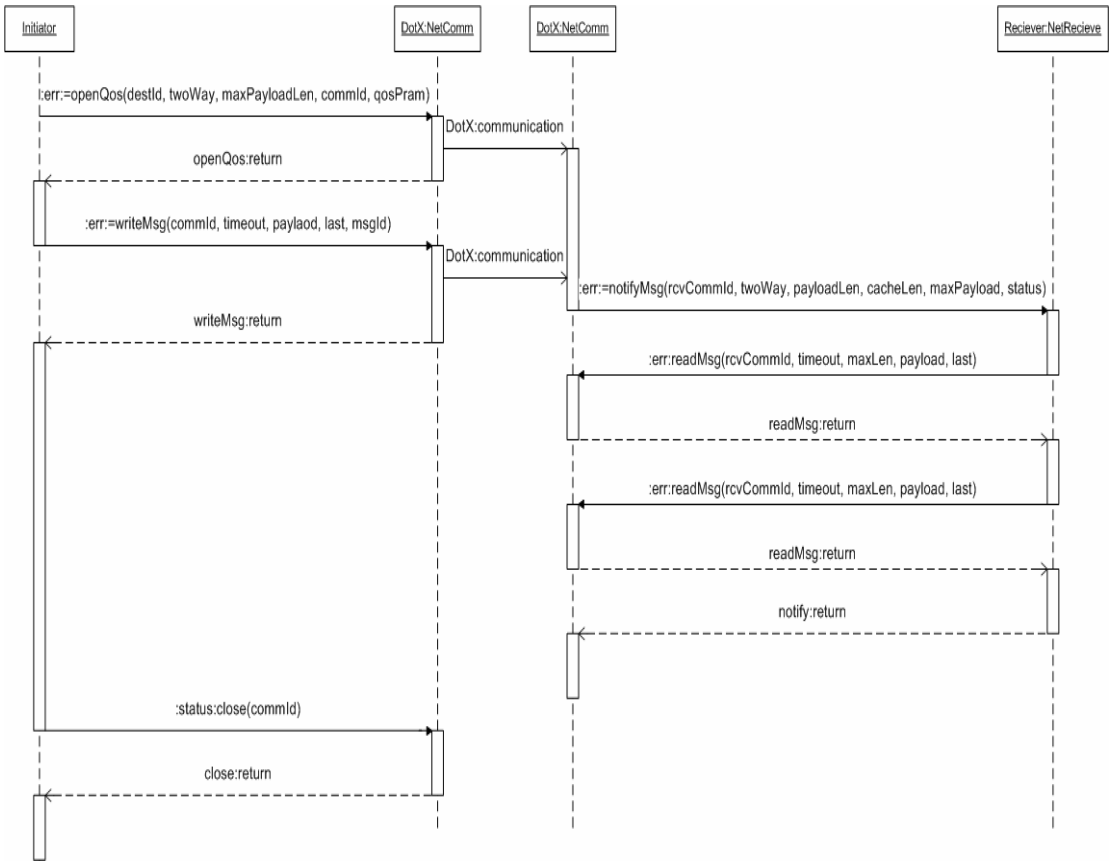


Figure B.4—Details of the read operation sequence diagram

Copyright © 2007 IEEE. All rights reserved.

The application is on the left, and the IEEE 1451.X layers are in the center. Note the open( ) call and TIM-side processing above the IEEE 1451.0 layer are not illustrated in this diagram. Here, we assume the application is performing an immediate read of TransducerChannel 1 from the desired TIM. Once the application invokes the read() method, the following processing occurs:

a) An appropriate command is formatted to instruct the TIM to read the desired TransducerChannel with the specified samplingmode (see 7.1.2.4) (immediate mode operation in this case). This packages all input arguments into the "inArg" ArgumentArray.

b) The Codec's encode( ) method is used to build the command "payload" OctetArray. As this is a read command, the "payload" contains the following octets. Note that the outgoing "payload" is 10 octets long.

   1) TransducerChannel identifier of 0x0001 as a UInt16. This is the TransducerChannel number to be read.

   2) Command Class of 0x02 as a UInt8 that indicates an "operational" command class.

   3) Command Function code as a UInt8. The command code is "Read TransducerChannel data-set segment" (value 1) in this case.

   4) Length of variable portion of 0x0004 as a UInt16.

   5) The DataSetOffset is the only argument of the command, and it contains 0x00000000 as a UInt32 since the data set contains a single value.

   6) The write( ) command is invoked on the IEEE 1451.X layer to initiate transfer of the OctetArray to the TIM. This thread will now block as it waits for the TIM to make the measurement. The maximum amount of time to wait is specified by the application.

   7) When the TIM-side processing has completed, the IEEE 1451.X layer will transfer a result OctetArray back to the NCAP. This will result in the notify( ) method being invoked.

   8) The Notify processing wakes up the blocked application thread.

   9) The read( ) method is invoked to retrieve the response OctetArray.

   10) The Codec's decode( ) method is used to convert the OctetArray back to an ArgumentArray.

   11) The CorrectionEngine's convert( ) method is used to generate corrected values (i.e., conversion from TIM-side to NCAP-side units).

   12) The ArgumentArray is further formatted to return the desired measurement attributes. This may involve extracting select information from cached TEDS (for example, units).

   13) The ArgumentArray is returned as an "out" parameter back to the application.

Copyright © 2007 IEEE. All rights reserved.

Annex C

(informative)

Guidance to Module Communication Interface

C.1 Module communication API discussion

The logical communication between the NCAP and TIMs or between TIMs is handled by this "Module Communication" API. The four aspects of this API are as follows:

a) The communication messaging may be either "one-way" from the initiator to the receiver or the communication may be "two-way" where the initiator sends a command to the receiver who then replies.

b) The communication messaging may be either "one-to-one" where only two devices are involved or the communication may be "one-to-many" where the initiator is communicating with a group of devices. The "one-to-many" is always limited to "one-way" communications.

c) The communication may be either "default" or "special" Quality of Service. Default is a best-effort delivery without any special communication mechanisms to provide better quality.

d) The physical interface may be a simple "point-to-point" link or may be a "network" where multiple devices share the communication medium.

The "ModuleCommunication" API is subdivided into three main interfaces: "Comm," "Registration," and "Receive." The "Comm" interface is implemented by the IEEE 1451.X layer and is called by the IEEE 1451.0 layer to control communications between the NCAP and TIMs. The "Registration" interface is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer to register itself and to register known destinations. The "Receive" interface is provided by the IEEE 1451.0 layer and is called by the IEEE 1451.X layer when incoming communication messages are received by the IEEE 1451.X layer across the link.

Each of these three interfaces is further subdivided into two sub-interfaces: the "point-to-point" case where an NCAP and TIM are directly communication with each other, and the "network" case where a NCAP and multiple TIMs are communicating together. The prefix "P2P" and "Net" are used to differentiate these cases. Figure C.1 illustrates these relationships for the Comm interfaces. Figure C.2 illustrates these relationships for the Registration interfaces. Figure C.3 illustrates these relationships for the Receive interfaces.

Note that "point-to-point" is from a communication point of view. It does not imply that the underlying physical link is point to point.

Copyright © 2007 IEEE. All rights reserved.

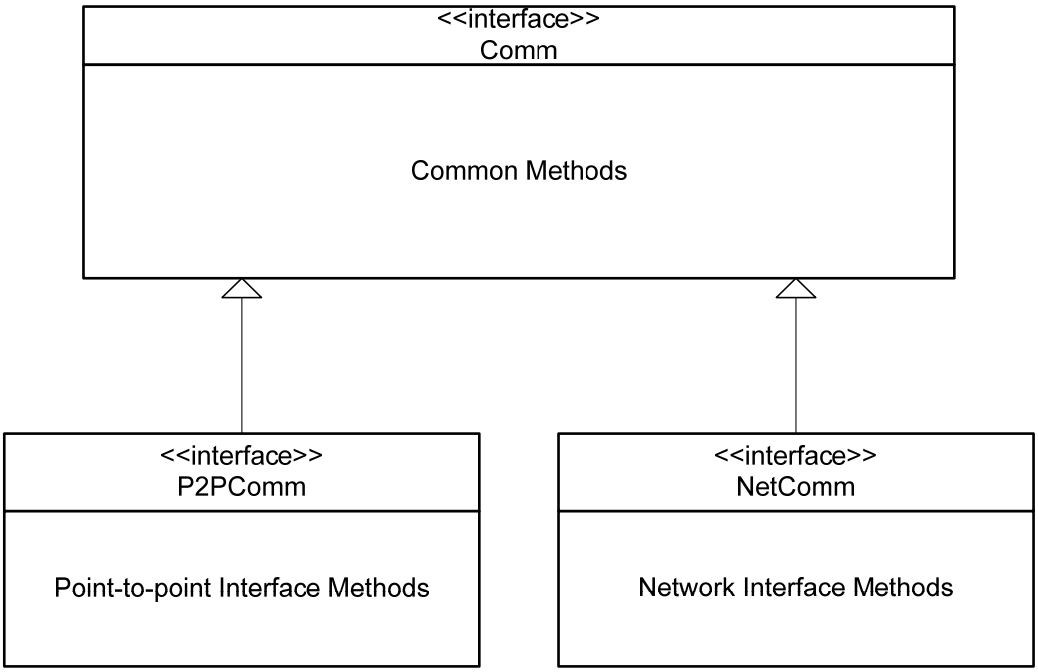


Figure C.1— Communications methods

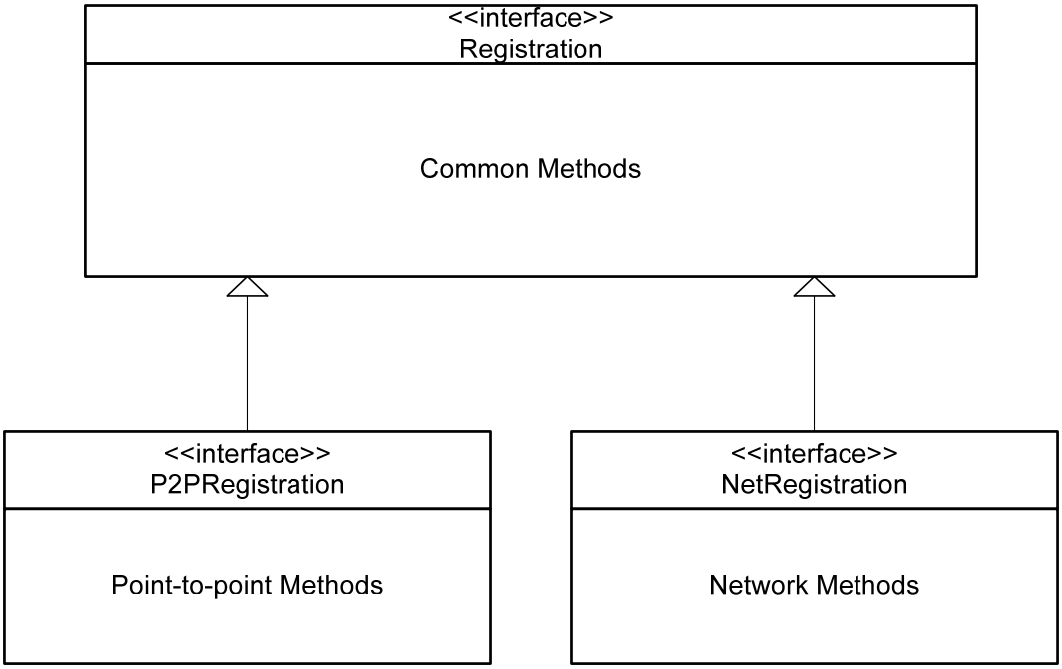


Figure C.2—Registration interfaces

Copyright © 2007 IEEE. All rights reserved.

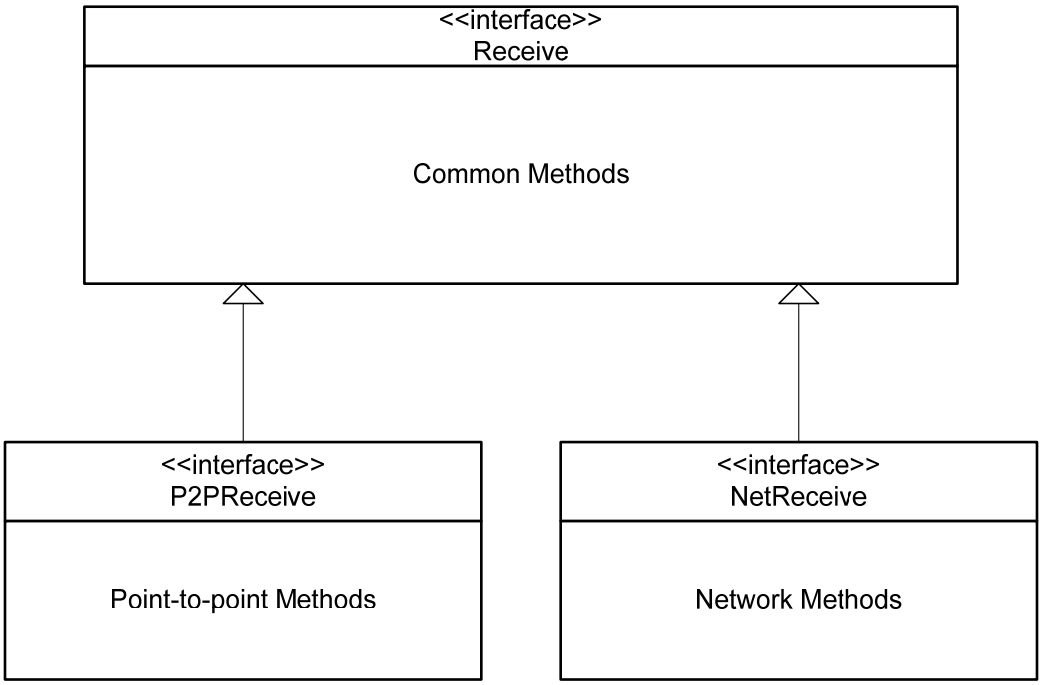


Figure C.3—Receive interfaces

C.2 Symmetric APIs

These APIs are symmetric and support NCAP-to-TIM, TIM-to-NCAP, and optionally TIM-to-TIM communications. Each communication operation begins with the initiating node sending a payload to one or more receiving nodes. Optionally, the receiving node may issue a reply to the initiating node.

The following communication patterns are supported:

Two-way/one-to-one. Example: NCAP issues a read TEDS command to a specific TIM. TIM replies with the TEDS contents.

One-way/one-to-one. Example: TIM generates a periodic measurement that is sent back to the NCAP.

One-way/one-to-many. Example: NCAP issues a group trigger command that flows to multiple TIMs participating in the measurement.

C.3 Implementation choices

The NCAP and TIMs may make different choices on what interfaces are implemented and still be able to communicate with each other. The key issue is whether the device needs to initiate communication to many devices or whether it only communicates to a single destination. The former case would need to implement the NetComm interface. The later case would implement the P2PComm interface. A device that only receives and replies (i.e., it never initiates) would also implement the P2PComm interface.

The second issue to consider is how many different IEEE 1451.X communication paths are supported on the device and whether they are static or dynamic in nature. In most cases, only a single IEEE 1451.X communication path is available. Consequently, the software build process (for example, linker) can be used to attach the IEEE 1451.X implementation to the IEEE 1451.0 implementation. This negates the need to use some methods of the Registration interface.

Copyright © 2007 IEEE. All rights reserved.

A slightly more complex device may provide a fixed set of IEEE 1451.X communication paths. Again, because these paths are known at build time, the SW build process (for example, linker) can be used to manage registration. The IEEE 1451.0 layer will be responsible for using the correct IEEE 1451.X instance when trying to communicate through that path.

A high-end device may need to be able to dynamically manage what IEEE 1451.X communication paths are available. For this situation, a dynamic registration mechanism is provided to allow the IEEE 1451.X implementation to notify the IEEE 1451.0 layer that it is available. This is appropriate for both P2PComm and NetComm situations.

A device that is connected to a multi-drop IEEE 1451.X network and needs to initiate communications to more than one destination would implement the NetComm interface. This interface provides additional methods and parameters to handle simultaneous overlapping communication transactions to multiple destinations.

C.4 Implementation examples

The simplest situation is a basic TIM that only replies to requests from the NCAP. This could be over a "point-to-point" link (for example, IEEE 1451.2-RS232) or over a "network" physical link (for example, Clause 7 of IEEE Std 1451.5-2007 [B4]). However from the point of view of this TIM, it only replies to requests. It never initiates a communication operation. Because the communication requirements are so simple, the TIM's IEEE 1451.X implementation would provide the P2PComm interface. Because there is only a single IEEE 1451.X Comm implementation, the registration mechanism would be through static linkage established during the software build process.

The next simplest case is a simple NCAP with a single point-to-point physical connection to a single TIM (for example, IEEE 1451.2-RS232). The NCAP's IEEE 1451.X implementation would also provide the P2PComm interface. The IEEE 1451.X-to-IEEE 1451.0 linkage would be through the software build process.

Growing in complexity is a NCAP with multiple point-to-point physical connections (for example, multiple IEEE 1451.2-RS232 ports). Multiple IEEE 1451.X instances are needed, one for each physical connection. The P2PComm interface would be provided in each case. The IEEE 1451.0 implementation on the NCAP is responsible for using the appropriate IEEE 1451.X interface when it communicates with each TIM.

The most complex NCAP needs to support multiple physical interfaces of various types, and the configuration is dynamic in nature. Each physical interface would need either a P2PComm or NetComm instance. These instances also dynamically register themselves so that the IEEE 1451.0 layer knows they are available.

C.5 Node communication parameters

In the NetComm configurations, each node on the network is uniquely identified by IEEE 1451.X-specific "node" communication parameters. For example with IEEE 1451.5-802.11, each node has a unique IP address and network ports are allocated for IEEE 1451.0 communications. Each IEEE 1451.X technology could have a different set of required parameters.

Although the nature of these parameters is specific to each IEEE 1451.X technology, the IEEE 1451.0 layer needs the ability to query and pass them around the system in a generic form. The getNodeParams( ) call is used to retrieve these parameters in an ArgumentArray. This call returns the parameters from the local IEEE 1451.X layer.

Copyright © 2007 IEEE. All rights reserved.

C.6 The destination identifier “destId” parameter

In the IEEE 1451.0 layer, the single “owning” NCAP and associated TIMs form a logical communication group. Each node (either the NCAP or a TIM) is assigned a unique UInt16 ID by the IEEE 1451.0 layer on the NCAP-side. Value 0x0000 is reserved as the broadcast address and a value of 0x0001 is reserved as the “owning” NCAP address. This “IEEE 1451.0 Destination ID” will be called the destId in the remainder of this document.

On the NCAP-side, the IEEE 1451.X layer is responsible for discovering all TIMs in the logical communication group and registering them with the NCAP-side IEEE 1451.0 layer by calling the registerDest( ) method. The IEEE 1451.0 layer will assign the unique “destId”, and the IEEE 1451.X layer shall cache appropriate network “node” information and associate it with this “destId.” For example, a IEEE 1451.5-802.11 implementation would need to associate the “destId” with the remote node’s IP address and port number.

On the TIM-side, the IEEE 1451.X layer must know how to communicate with the “owning” NCAP using “destId” 0x0001. No registration with the TIM-side IEEE 1451.0 layer is required.

As an optional feature, in cases where a TIM is initiating communicates with a different destination (for example, a TIM-generated trigger command to another TIM or a group), an IEEE 1451.0 command is provided on the TIM-side to handle the necessary configuration. The ArgumentArray retrieved via the getNodeParams( ) on the destination TIM (or group) will be passed to the initiating TIM’s IEEE 1451.0 layer. This information will be passed down to the initiating TIM’s IEEE 1451.X layer via the addDestination( ) call where it can cache the necessary private network destination information. In this case, the IEEE 1451.X layer on the initiating TIM would not call the IEEE 1451.0’s registerDest( ) method.

The “destId” 0x0000 is reserved as the broadcast address for the logical communication group. It may only be used to send one-way messages to all nodes within the group.

C.7 The communication session “commId” parameter

When initiating communications, the IEEE 1451.0 layer will invoke the open( ) or openQoS( ) calls and specify a “destId” and a unique “commId.” If these calls succeed, IEEE p1451.0 will refer to that session by the “commId.” When the IEEE 1451.0 layer is finished, it will invoke close( ) to notify the IEEE 1451.X layer that the session has completed and IEEE 1451.X resources can be safely reclaimed.

Optionally, the IEEE 1451.X layer can support multiple open( ) calls to the same or different destinations. When IEEE 1451.X has reached network or memory resource limits, it should generate appropriate fail codes. The IEEE 1451.0 layer should attempt to call close( ) to free up IEEE 1451.X resources before subsequent calls to open( ).

By supporting multiple open( ) calls, IEEE 1451.X is allowing overlapped communications which usually result in improved efficiencies. This is an optional feature.

In cases where the node is the receiving role, IEEE 1451.X will invoke the notifyMsg( ) method on the IEEE 1451.0 layer and provide a unique “commId.” In this case, open( ) is not called on the receiving side. The IEEE 1451.0 layer will use that “commId” to read the incoming network data via the readMsg( ) call. Optionally, the IEEE 1451.0 layer will use that “commId” to send a response back to the initiating node. The IEEE 1451.0 layer will not call close( ) on this commId because this is handled internally by the IEEE 1451.X layer.

Copyright © 2007 IEEE. All rights reserved.

C.8 The message transaction identifier “msgId” parameter

When initiating communications after successfully calling the open( ) or openQoS( ) calls, the IEEE 1451.0 layer will begin communications by calling the writeMsg( ) method. The IEEE 1451.0 layer will provide the appropriate “commId” so that the IEEE 1451.X layer will know what communication session to use. The IEEE 1451.0 layer will also specify a unique “msgId” so that the IEEE 1451.X layer will know how to associate a response to the outgoing message. The IEEE 1451.X layer is responsible for caching both these IDs for use in the notifyRsp( ) callback to the IEEE 1451.0 layer when the response has been received.

Optionally, the IEEE 1451.X layer can support multiple calls to writeMsg( ) on the same “commId” but with different “msgIds.” This allows for overlapped communications over the same communication session and can result in significant performance improvements. When the IEEE 1451.X layer reaches memory or network limits, it should fails on subsequent writeMsg( ) calls. The IEEE 1451.0 layer would back off until pending responses have been received before further calls to writeMsg( ).

C.9 Memory constrained implementations

The design of these APIs is intended to work on devices that have severe RAM memory constraints like an 8-bit PIC microprocessor. When the IEEE 1451.0 layer invokes the methods that read or write the OctetArray from/to the IEEE 1451.X layer (for example, readMsg( ), readRsp( ), writeMsg( ), and writeRsp( )), the IEEE 1451.0 layer will always specify the maximum number of octets that it is providing or can accept. Multiple calls to these methods may be required to transfer large OctetArrays. A “last” flag parameter is used to signal when the complete OctetArray has been transferred.

In these memory-constrained situations, it is extremely important that the IEEE 1451.X layer provide a mechanism where the flow of octets across the network can be regulated. A common example is a memory constrained TIM that is performing a write TEDS operation. If the TEDS block is larger than the available TIM-side memory, the IEEE 1451.0 layer on the TIM will need to read the OctetArray in pieces via the readRsp( ) call. With each piece, it will write the data to appropriate persistence storage (for example, flash). Note that this may take a very long time if the flash must be erased. In any case, it is a IEEE 1451.X responsibility to wait for the IEEE 1451.0 layer to make the subsequent readMsg( ) call without overrunning any local network buffers. In most cases, this will require IEEE 1451.X flow control messages between the NCAP and TIM.

C.10 IEEE 1451.X communication state machines

Figure C.4 and Figure C.5 illustrate the state transitions for the “initiating” and “receiving” nodes. In the NetComm case when simultaneous transactions are supported, each <“commId”, “msgId”> pair results in new invocation of these state machines. Figure C.4 diagrams the state machine for the initiating node and Figure C.5 gives the state diagram for the receiving node.

Copyright © 2007 IEEE. All rights reserved.

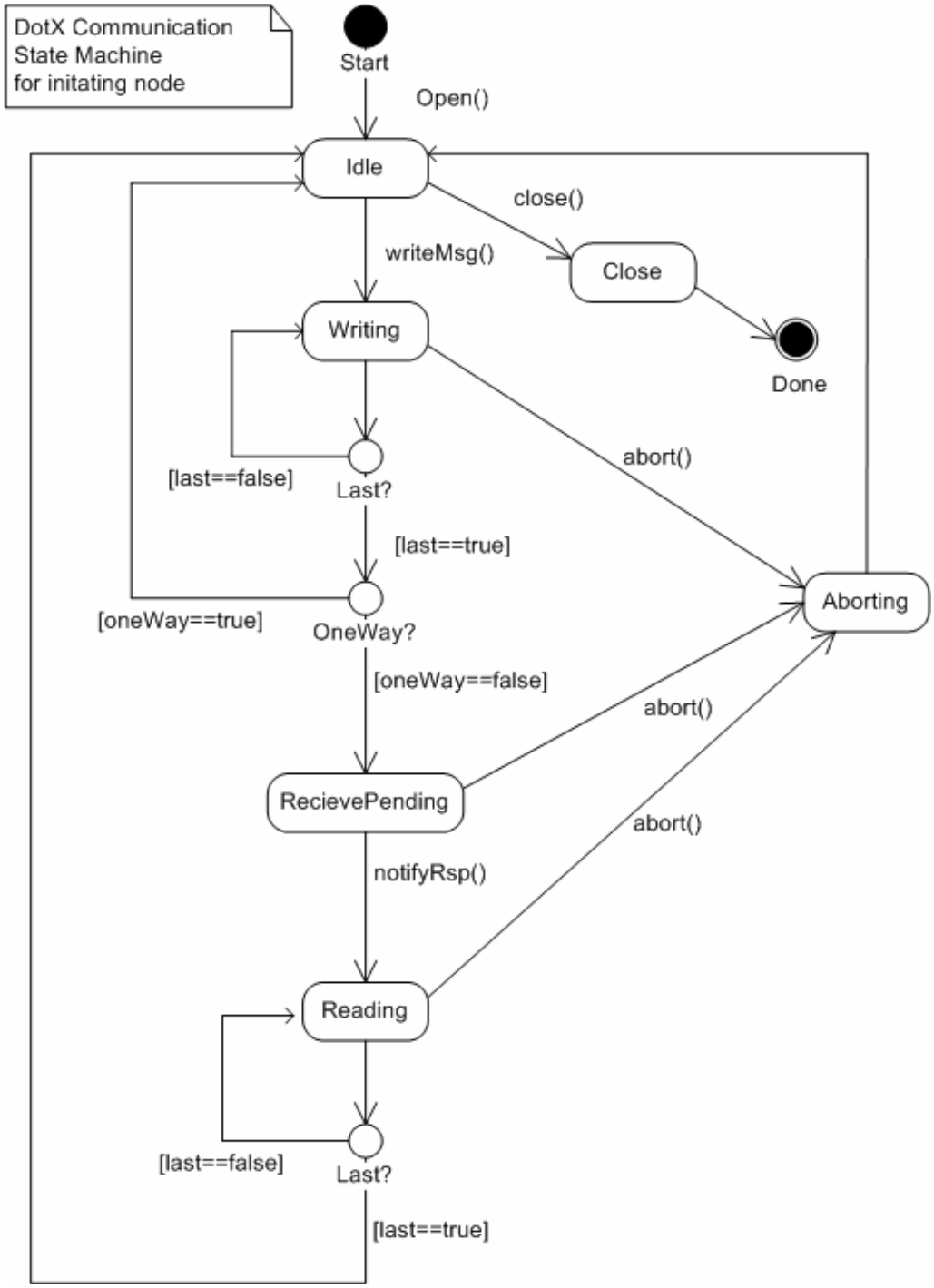


Figure C.4—Communications state machine for the initiating node

Copyright © 2007 IEEE. All rights reserved.

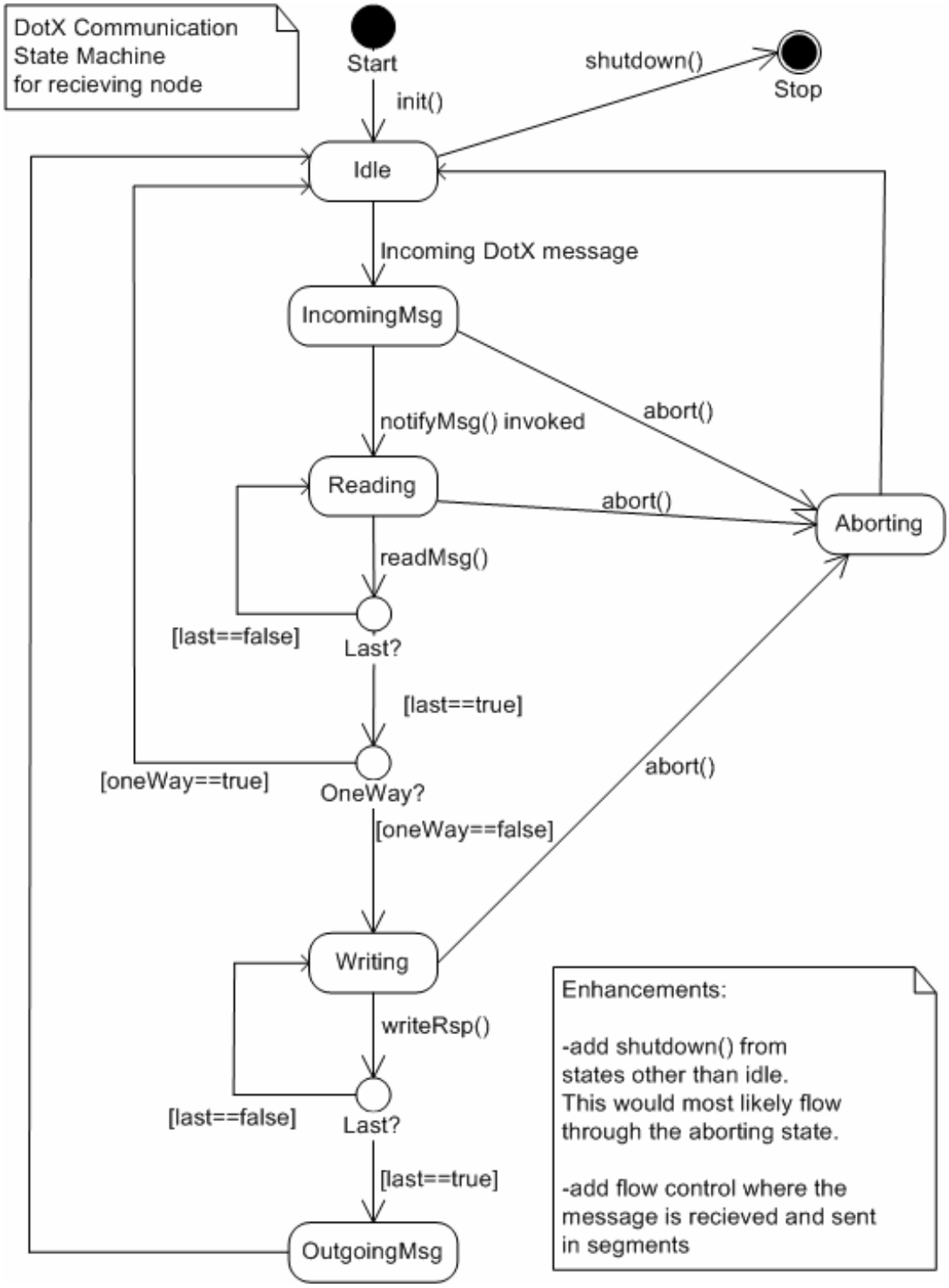


Figure C.5—Communications state machine for the receiving node

Copyright © 2007 IEEE. All rights reserved.

C.11 Communication sequence

The sequence diagram[18] in Figure C.6 illustrates the typical call sequence between the initiating and receiving node for a "one-way" operation. Information flows from the IEEE 1451.0 layer on the initiating node to the IEEE 1451.0 layer on the receiving node.

This example assumes that the receive side has significant memory limitations that require multiple readMsg( ) calls. Also, using the IEEE 1451.X thread for receive-side processing is illustrated. The notifyMsg( ) call only returns at the completion of the receive-side processing. Also, the openQoS( ) call was used in order to provide "quality of service" requirements to the IEEE 1451.X layer.

On the initiating node, the IEEE 1451.0 layer invokes openQoS(), writeMsg(), and close(). On the receiving node, the IEEE 1451.X layer invokes the receiveMsg() method to inform the IEEE 1451.0 layer that a new communication operation has started. The IEEE 1451.0 layer calls back into the IEEE 1451.X layer with two readMsg() calls to retrieve the information in stages. Note that the open(), openQoS(), and close() calls are only invoked on the initiating node.

Although only a single asynchronous message is illustrated between the two IEEE 1451.X layers on the initiating and receiving nodes, the actual number of network messages is outside the scope of this standard. The IEEE 1451.X layer may need to send numerous network messages in both directions to complete the communication operation. Issues like flow control, segmentation, reassembly, retries, and congestion avoidance may need to be handled.

The sequence diagram in Figure C.7 illustrates a "two-way" flow of information from the IEEE 1451.0 layer on the initiating node to the IEEE 1451.0 layer on the receiving node for the "Network" case. The receiving node generates a response that is communicated back to the initiating node. For illustration purposes, the open( ) call is used because default "quality of service" is acceptable to the initiator.

In cases where improved performance is required, it is advised to use a multi-threaded architecture to accommodate overlapping communication processing. Note that the notifyMsg( ) and notifyRsp( ) calls do not block for the duration of the processing. The implementation shall provide some mechanism to wake up or launch a processing thread that completes the necessary processing. Although illustrated as running in the same object, this is an implementation choice as well.

[18] In a sequence diagram, time flows down this figure. A standard blocking function or method call is indicated by a solid arrow with a solid head. The return action is represented as a dashed open arrow. For asynchronous method calls, a solid arrow with an open head is used.

Copyright © 2007 IEEE. All rights reserved.

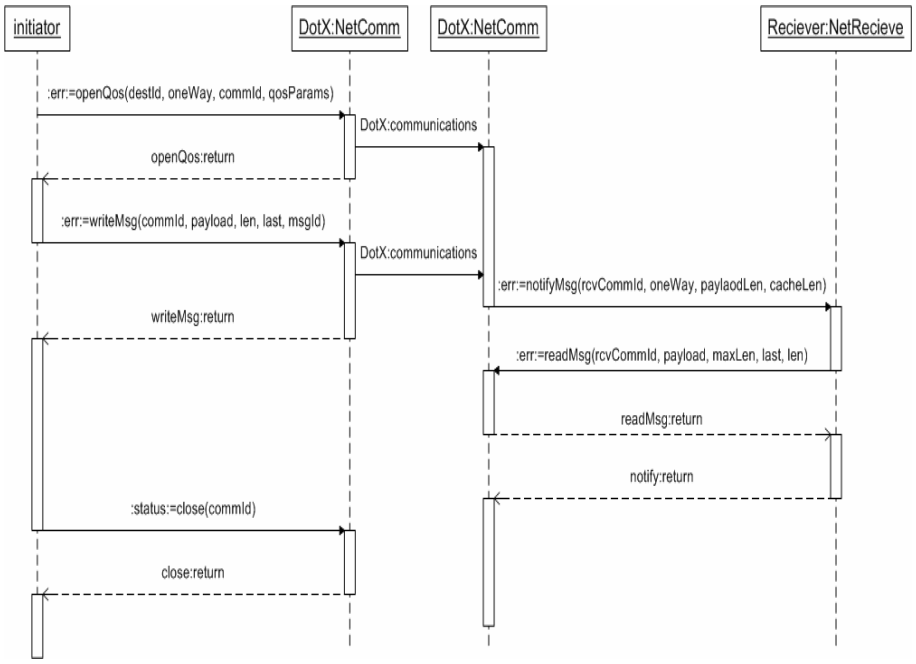


Figure C.6—Simple communications sequence

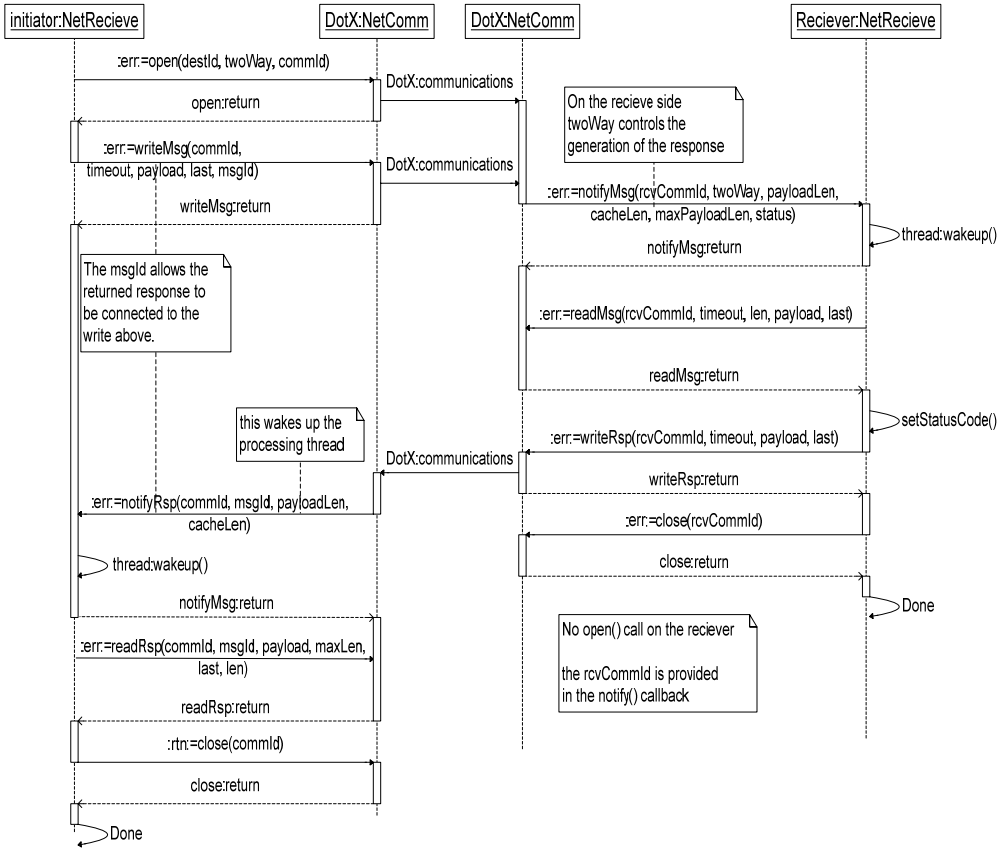


Figure C.7—Two way communications sequence

Copyright © 2007 IEEE. All rights reserved.

Annex D

(informative)

XML Schema for Text-based TEDS

D.1 Introduction to text-based TEDS

Text-based TEDS provide a mechanism for the manufacturer to embed textual information into a smart transducer. The text-based TEDS defined within this standard are as follows:

Meta-Identification TEDS (schema is informative).

TransducerChannel Identification TEDS (schema is informative).

Calibration Identification TEDS (schema is informative).

Commands TEDS (schema is normative).

Location and Title TEDS (schema is informative).

Geographic Location TEDS (schema is normative and provided by another body).

Any text-based TEDS has a structure that encapsulates the textual information as a "data block" of arbitrary length. The TEDS structure accommodates multiple blocks to support presentation of the content in multiple languages. The only constraint imposed by this standard is that each data block is an XML "instance document."

XML is a flexible meta-markup language. In simple terms, it is a language in which the markup tags can be defined as needed within the context of the information domain in which the tags are used. These tags shall be defined and organized according to a set of rules. There are two vehicles available to declare tags: the Document Type Definition (DTD) or the XML Schema. Throughout the remainder of this annex, we will refer to these declarations as "schema."

The schema may be declared inline (entirely contained within the instance document), it may be external, or it may be external with internal additions or extensions. This annex provides schemas that may serve as the external schema for all text-based TEDS defined in this standard. This schema has two objectives:

— Use of an external schema minimizes the size of the instance document.

— Use of a common schema promotes standardization of the domain semantics.

The instance document within the text-based TEDS is not **required** to reference this schema. The manufacturer may assume that any processor of the instance document can obtain the schema without an overt reference from the instance document.

The schemas are provided in English. The user is free to translate them into any language that they need.

D.2 Schema

Schema for the Meta-Identification TEDS, TransducerChannel TEDS, and Calibration Identification TEDS are provided and are based on the information in the corresponding TEDS defined in IEEE Std 1451.2-1997. The IEEE 1451.0 layer does not require that these schemas be used. A suggested schema is also provided for the Location and Title TEDS. The schema for the Commands TEDS is required if a

Copyright © 2007 IEEE. All rights reserved.

Commands TEDS is provided. The schema for the Geographic Location TEDS that is provided in the data block shall be in the Geography Markup Language (GML) specified in ISO/DIS 19136. Geographic information – Geography Markup Language, International Organization for Standardization, 2005 or later. The schema for the Units Extension TEDS is required if a Units Extension TEDS is provided. See 5.5.2.7 for additional details. Electronic copies of these schemas can be found at the following URL:

http://grouper.ieee.org/groups/1451/0/1451HTTPAPI/

D.3 Include file "SmartTransducerDataModel.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding"
xmlns:corba="http://www.omg.org/IDL-WSDL/1.0/"
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <!--This is IEEE 1451.0 data types-->
   <xs:simpleType name="Int8">
      <xs:restriction base="xs:byte"/>
   </xs:simpleType>
   <xs:simpleType name="Int16">
      <xs:restriction base="xs:short"/>
   </xs:simpleType>
   <xs:simpleType name="Int32">
      <xs:restriction base="xs:int"/>
   </xs:simpleType>
   <xs:simpleType name="UInt8">
      <xs:restriction base="xs:unsignedByte"/>
   </xs:simpleType>
   <xs:simpleType name="UInt16">
      <xs:restriction base="xs:unsignedShort"/>
   </xs:simpleType>
   <xs:simpleType name="UInt32">
      <xs:restriction base="xs:unsignedInt"/>
   </xs:simpleType>
   <xs:simpleType name="Float32">
      <xs:restriction base="xs:float"/>
   </xs:simpleType>
   <xs:simpleType name="Float64">
      <xs:restriction base="xs:double"/>
   </xs:simpleType>
   <xs:simpleType name="_Boolean">
      <xs:annotation>
         <xs:documentation source="boolean "/>
      </xs:annotation>
      <xs:restriction base="xs:boolean"/>
   </xs:simpleType>
   <xs:simpleType name="Octet">
      <xs:restriction base="xs:unsignedByte"/>
   </xs:simpleType>
   <xs:simpleType name="_String">
      <xs:annotation>
         <xs:documentation source="string"/>
      </xs:annotation>
      <xs:restriction base="xs:string"/>
   </xs:simpleType>
   <xs:simpleType name="Int8Array">
      <xs:annotation>
         <xs:documentation source="Array of Int8"/>
      </xs:annotation>
```


Copyright © 2007 IEEE. All rights reserved.

```
   <xs:list itemType="stml:Int8"/>
</xs:simpleType>
<xs:simpleType name="Int16Array">
   <xs:annotation>
      <xs:documentation source="Array of Int16"/>
   </xs:annotation>
   <xs:list itemType="stml:Int16"/>
</xs:simpleType>
<xs:simpleType name="Int32Array">
   <xs:annotation>
      <xs:documentation source="Array of  Int32"/>
   </xs:annotation>
   <xs:list itemType="stml:Int32"/>
</xs:simpleType>
<xs:simpleType name="UInt8Array">
   <xs:annotation>
      <xs:documentation source="Array of UInt8"/>
   </xs:annotation>
   <xs:list itemType="stml:UInt8"/>
</xs:simpleType>
<xs:simpleType name="UInt16Array">
   <xs:annotation>
      <xs:documentation source="Array of UInt16"/>
   </xs:annotation>
   <xs:list itemType="stml:UInt16"/>
</xs:simpleType>
<xs:simpleType name="UInt32Array">
   <xs:annotation>
      <xs:documentation source="Array of UInt32"/>
   </xs:annotation>
   <xs:list itemType="stml:UInt32"/>
</xs:simpleType>
<xs:simpleType name="Float32Array">
   <xs:annotation>
      <xs:documentation source="Array of Float32"/>
   </xs:annotation>
   <xs:list itemType="stml:Float32"/>
</xs:simpleType>
<xs:simpleType name="Float64Array">
   <xs:annotation>
      <xs:documentation source="Array of Float64"/>
   </xs:annotation>
   <xs:list itemType="stml:Float64"/>
</xs:simpleType>
<xs:simpleType name="OctetArray">
   <xs:annotation>
      <xs:documentation source="Array of Octet"/>
   </xs:annotation>
   <xs:list itemType="stml:Octet"/>
</xs:simpleType>
<xs:simpleType name="BooleanArray">
   <xs:annotation>
      <xs:documentation source="Array of _Boolean"/>
   </xs:annotation>
   <xs:list itemType="stml:_Boolean"/>
</xs:simpleType>
<xs:simpleType name="StringArray">
   <xs:annotation>
      <xs:documentation source="Array of _String"/>
   </xs:annotation>
   <xs:list itemType="stml:_String"/>
</xs:simpleType>
<xs:simpleType name="ErrorCode" final="restriction">
   <xs:restriction base="xs:string">
```


Copyright © 2007 IEEE. All rights reserved.

```
            <xs:enumeration value="NO_ERROR"/>
            <xs:enumeration value="INVALID_COMMID"/>
            <xs:enumeration value="UNKNOWN_DESTID"/>
            <xs:enumeration value="TIMEOUT"/>
            <xs:enumeration value="NETWORK_FAILURE"/>
            <xs:enumeration value="NETWORK_CORRUPTION"/>
            <xs:enumeration value="MEMORY"/>
            <xs:enumeration value="QOS_FAILURE"/>
            <xs:enumeration value="MCAST_NOT_SUPPORTED"/>
            <xs:enumeration value="UNKNOWN_GROUPID"/>
            <xs:enumeration value="UNKNOWN_MODULEID"/>
            <xs:enumeration value="UNKNOWN_MSGID"/>
            <xs:enumeration value="NOT_GROUP_MEMBER"/>
            <xs:enumeration value="ILLEGAL_MODE"/>
            <xs:enumeration value="LOCKED_RESOURCE"/>
            <xs:enumeration value="FATAL_TEDS_ERROR"/>
            <xs:enumeration value="NON_FATAL_TEDS_ERROR"/>
            <xs:enumeration value="CLOSE_ON_LOCKED_RESOURCE"/>
            <xs:enumeration value="LOCK_BROKEN"/>
            <xs:enumeration value="NETWORK_RESOURCE_EXCEEDED"/>
            <xs:enumeration value="MEMORY_RESOURCE_EXCEEDED"/>
         </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="TypeCode">
         <xs:annotation>
            <xs:documentation>Each valid type of 1451.0 Argument has a
unique typeCode.</xs:documentation>
         </xs:annotation>
         <xs:restriction base="xs:string">
            <xs:enumeration value="UNKNOWN_TC"/>
            <xs:enumeration value="INT8_TC"/>
            <xs:enumeration value="INT16_TC"/>
            <xs:enumeration value="INT32_TC"/>
            <xs:enumeration value="UINT8_TC"/>
            <xs:enumeration value="UINT16_TC"/>
            <xs:enumeration value="UINT32_TC"/>
            <xs:enumeration value="FLOAT32_TC"/>
            <xs:enumeration value="FLOAT64_TC"/>
            <xs:enumeration value="STRING_TC"/>
            <xs:enumeration value="OCTET_TC"/>
            <xs:enumeration value="BOOLEAN_TC"/>
            <xs:enumeration value="TIME_INSTANCE_TC"/>
            <xs:enumeration value="TIME_DURATION_TC"/>
            <xs:enumeration value="QOS_PARAMS_TC"/>
            <xs:enumeration value="INT8_ARRAY_TC"/>
            <xs:enumeration value="INT16_ARRAY_TC"/>
            <xs:enumeration value="INT32_ARRAY_TC"/>
            <xs:enumeration value="UINT8_ARRAY_TC"/>
            <xs:enumeration value="UINT16_ARRAY_TC"/>
            <xs:enumeration value="UINT32_ARRAY_TC"/>
            <xs:enumeration value="FLOAT16_ARRAY_TC"/>
            <xs:enumeration value="FLOAT32_ARRAY_TC"/>
            <xs:enumeration value="STRING_ARRAY_TC"/>
            <xs:enumeration value="OCTET_ARRAY_TC"/>
            <xs:enumeration value="BOOLEAN_ARRAY_TC"/>
            <xs:enumeration value="TIME_INSTANCE_ARRAY_TC"/>
            <xs:enumeration value="TIME_DURATION_ARRAY_TC"/>
         </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="UUID">
         <xs:list itemType="xs:short"/>
      </xs:simpleType>
      <xs:simpleType name="ErrorCodeSource">
         <xs:restriction base="xs:string">
```


Copyright © 2007 IEEE. All rights reserved.

```
            <xs:enumeration value="LOCAL_1451_0_LAYER"/>
            <xs:enumeration value="LOCAL_1451_X_LAYER"/>
            <xs:enumeration value="REMOTE_1451_0_LAYER"/>
            <xs:enumeration value="REMOTE_1451_X_LAYER"/>
            <xs:enumeration value="REMOTE_APPLICATION_LAYER"/>
         </xs:restriction>
      </xs:simpleType>
      <xs:element name="Argument" type="stml:ArgumentType"/>
      <xs:element name="ArgumentArray" type="stml:ArgumentArrayType"/>
      <xs:element name="QoSParam" type="stml:QoSParamType"/>
      <xs:element name="TimeDurationArray"
type="stml:TimeInstanceArrayType"/>
      <xs:element name="TimeDuration" type="stml:TimeDurationType"/>
      <xs:element name="TimeInstanceArray"
type="stml:TimeInstanceArrayType"/>
      <xs:element name="TimeInstance" type="stml:TimeInstanceType"/>
      <xs:complexType name="TimeDurationType">
         <xs:sequence>
            <xs:element name="secs" type="stml:UInt32"/>
            <xs:element name="nsecs" type="stml:UInt32"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="QoSParamType">
         <xs:annotation>
            <xs:documentation source="QoSParams structure S"/>
         </xs:annotation>
         <xs:sequence>
            <xs:element name="service" type="stml:_Boolean"/>
            <xs:element name="period" type="stml:TimeDurationType"/>
            <xs:element name="transmitSize" type="stml:UInt32"/>
            <xs:element name="accesLatency" type="stml:TimeDurationType"/>
            <xs:element name="transmitLatency"
type="stml:TimeDurationType"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="TimeInstanceArrayType">
         <xs:sequence>
            <xs:element name="TimeInstance" type="stml:TimeDurationType"
minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="size" type="xs:short"/>
      </xs:complexType>
      <xs:complexType name="TimeInstanceType">
         <xs:sequence>
            <xs:element name="secs" type="stml:UInt32"/>
            <xs:element name="nsecs" type="stml:UInt32"/>
            <xs:element name="epoch" type="stml:UInt8"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="TimeDurationArrayType">
         <xs:sequence>
            <xs:element name="timeDuration" type="stml:TimeDurationType"
minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="size" type="xs:short"/>
      </xs:complexType>
      <xs:complexType name="ArgumentType">
         <xs:annotation>
            <xs:documentation source="A generic data container"/>
         </xs:annotation>
         <xs:sequence>
            <xs:element name="discriminator" type="stml:TypeCode"/>
            <xs:choice>
```


Copyright © 2007 IEEE. All rights reserved.

```
            <xs:element name="ValueError" type="stml:_Boolean"
minOccurs="0"/>
            <xs:element name="valueInt8" type="stml:UInt8"
minOccurs="0"/>
            <xs:element name="valueInt16" type="stml:Int16"
minOccurs="0"/>
            <xs:element name="valueInt32" type="stml:Int32"
minOccurs="0"/>
            <xs:element name="valueUInt8" type="stml:UInt8"
minOccurs="0"/>
            <xs:element name="valueUInt16" type="stml:UInt16"
minOccurs="0"/>
            <xs:element name="valueUIn32" type="stml:UInt32"
minOccurs="0"/>
            <xs:element name="valueFloat32" type="stml:Float32"
minOccurs="0"/>
            <xs:element name="valueFloat64" type="stml:Float64"
minOccurs="0"/>
            <xs:element name="valueString" type="stml:_String"
minOccurs="0"/>
            <xs:element name="valueOctet" type="stml:Octet"
minOccurs="0"/>
            <xs:element name="valueBoolean" type="stml:_Boolean"
minOccurs="0"/>
            <xs:element name="valueTimeInstance"
type="stml:TimeDurationType" minOccurs="0"/>
            <xs:element name="valueTimeDuration"
type="stml:TimeDurationType" minOccurs="0"/>
            <xs:element name="valueQosParams" type="stml:QoSParamType"
minOccurs="0"/>
            <xs:element name="valueUInt8Array" type="stml:UInt8Array"
minOccurs="0"/>
            <xs:element name="valueUInt16Array" type="stml:UInt16Array"
minOccurs="0"/>
            <xs:element name="valueUInt32Array" type="stml:UInt32Array"
minOccurs="0"/>
            <xs:element name="valueFloat32Array"
type="stml:Float32Array" minOccurs="0"/>
            <xs:element name="valueFloat64Array"
type="stml:Float64Array" minOccurs="0"/>
            <xs:element name="valueStringArray" type="stml:StringArray"
minOccurs="0"/>
            <xs:element name="valueOctetArray" type="stml:OctetArray"
minOccurs="0"/>
            <xs:element name="valueBooleanArray"
type="stml:BooleanArray" minOccurs="0"/>
            <xs:element name="valueTimeInstanceArray"
type="stml:TimeInstanceArrayType" minOccurs="0"/>
            <xs:element name="valueTimeDurationArray"
type="stml:TimeDurationArrayType" minOccurs="0"/>
         </xs:choice>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="Units">
      <xs:annotation>
         <xs:documentation source="Definitions of the SI base units are
given in The International System of Units (SI), "/>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="interpret" type="stml:UInt8"/>
         <xs:element name="radians" type="stml:UInt8"/>
         <xs:element name="steradians" type="stml:UInt8"/>
         <xs:element name="meters" type="stml:UInt8"/>
         <xs:element name="kilograms" type="stml:UInt8"/>
```


Copyright © 2007 IEEE. All rights reserved.

```
         <xs:element name="seconds" type="stml:UInt8"/>
         <xs:element name="amperes" type="stml:UInt8"/>
         <xs:element name="kelvins" type="stml:UInt8"/>
         <xs:element name="moles" type="stml:UInt8"/>
         <xs:element name="candelas" type="stml:UInt8"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ArgumentArrayType">
      <xs:sequence>
         <xs:element name="argument" type="stml:ArgumentType"
minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="size" type="xs:short"/>
   </xs:complexType>
</xs:schema>
```

D.4 Include file “TextTEDS.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include schemaLocation="SmartTransducerDataModel.xsd"/>
   <xs:include schemaLocation="MetaIdentificationTEDS.xsd"/>
   <xs:include
schemaLocation="TransducerChannelIdentificationTEDS.xsd"/>
   <xs:include schemaLocation="CalibrationIdentificationTEDS.xsd"/>
   <xs:include schemaLocation="CommandsTEDS.xsd"/>
   <xs:include schemaLocation="LocationAndTitleTEDS.xsd"/>
   <xs:include schemaLocation="GeographicLocationTEDS.xsd"/>
   <xs:element name="TextTEDS" type="stml:TextTEDSType"
abstract="true"/>
   <xs:complexType name="TextTEDSDataBlockType">
      <xs:annotation>
         <xs:documentation>Structure of a Text-based TEDS data
block</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="TEDSID">
            <xs:annotation>
               <xs:documentation>TEDS Identification
Header</xs:documentation>
            </xs:annotation>
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Type" type="xs:short" default="3"/>
                  <xs:element name="Length" type="xs:short"
default="4"/>
                  <xs:element name="Value" type="stml:UInt8Array"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="NumLang">
            <xs:annotation>
               <xs:documentation>The number N of different language
blocks in this TEDS</xs:documentation>
            </xs:annotation>
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Type" type="xs:short" default="10"/>
```


Copyright © 2007 IEEE. All rights reserved.

```
                  <xs:element name="Length" type="xs:short"
default="1"/>
                  <xs:element name="Value" type="stml:UInt8"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="DirBlock">
            <xs:annotation>
               <xs:documentation>Language block description This block
is repeated N times</xs:documentation>
            </xs:annotation>
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Type" type="xs:short" default="11"/>
                  <xs:element name="Length" type="xs:short"/>
                  <xs:sequence>
                     <xs:element name="LangCode">
                        <xs:annotation>
                           <xs:documentation>Language code from ISO 639
(two letters in USASCII)</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="Type" type="xs:short"
default="20"/>
                              <xs:element name="Length" type="xs:short"
default="2"/>
                              <xs:element name="Value"
type="stml:UInt8"/>
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                     <xs:element name="LangOffset">
                        <xs:annotation>
                           <xs:documentation>Language
offset</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="Type" type="xs:short"
default="21"/>
                              <xs:element name="Length" type="xs:short"
default="4"/>
                              <xs:element name="Value"
type="stml:UInt32"/>
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                     <xs:element name="LangLength">
                        <xs:annotation>
                           <xs:documentation>Language length =
LL</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="Type" type="xs:short"
default="22"/>
                              <xs:element name="Length" type="xs:short"
default="4"/>
                              <xs:element name="Value"
type="stml:UInt32"/>
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
```


Copyright © 2007 IEEE. All rights reserved.

```
                     <xs:element name="Compress">
                        <xs:annotation>
                           <xs:documentation>Enumeration identifying the
compression technique used.</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="Type" type="xs:short"
default="23"/>
                              <xs:element name="Length" type="xs:short"
default="1"/>
                              <xs:element name="Value"
type="stml:UInt8"/>
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                  </xs:sequence>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="SubSum">
            <xs:annotation>
               <xs:documentation>Non-displayable data
checksum</xs:documentation>
            </xs:annotation>
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Type" type="xs:short" default="12"/>
                  <xs:element name="Length" type="xs:short"
default="2"/>
                  <xs:element name="Value" type="stml:UInt16"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:simpleType name="LanguageCodes">
      <xs:restriction base="xs:string">
         <xs:enumeration value="aa"/>
         <xs:enumeration value="da"/>
         <xs:enumeration value="de"/>
         <xs:enumeration value="en"/>
         <xs:enumeration value="es"/>
         <xs:enumeration value="eu"/>
         <xs:enumeration value="fi"/>
         <xs:enumeration value="fr"/>
         <xs:enumeration value="ga"/>
         <xs:enumeration value="it"/>
         <xs:enumeration value="nl"/>
         <xs:enumeration value="no"/>
         <xs:enumeration value="pl"/>
         <xs:enumeration value="pt"/>
         <xs:enumeration value="ru"/>
         <xs:enumeration value="sv"/>
         <xs:enumeration value="vi"/>
         <xs:enumeration value="zu"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:complexType name="TextTEDSType">
      <xs:annotation>
         <xs:documentation>This is a class of optional TEDS. The
function of these TEDS is to provide information for display to an
operator. There are six TEDS listed in Table 17 that fall into this
category. They are the Meta-Identification TEDS, TransducerChannel
```


Copyright © 2007 IEEE. All rights reserved.

```
Identification TEDS, Calibration-Identification TEDS, Commands TEDS and
the Location and Title TEDS and the Geographic Location TEDS.
</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="TEDSLength" type="stml:UInt32"/>
         <xs:element name="TextTEDSDataBlock"
type="stml:TextTEDSDataBlockType"/>
         <xs:element name="Checksum" type="stml:UInt16"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

D.5 MetaIdentificationTEDS.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include schemaLocation="SmartTransducerDataModel.xsd"/>
   <xs:include schemaLocation="TextTEDS.xsd"/>
   <xs:element name="Meta-IdentificationTEDS" type="stml:Meta-
IdentificationTEDSType" substitutionGroup="stml:TextTEDS">
      <xs:annotation>
         <xs:documentation>1451 Smart Sensor Meta-Identification TEDS
schema</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:complexType name="Meta-IdentificationTEDSDataBlockType">
      <xs:sequence>
         <xs:element name="ManufacturerId" type="stml:_String"
minOccurs="0" maxOccurs="255"/>
         <xs:element name="ModelNo" type="stml:_String" minOccurs="0"
maxOccurs="255"/>
         <xs:element name="VersionCode" type="stml:_String"
minOccurs="0" maxOccurs="255"/>
         <xs:element name="SerialNo" type="stml:_String" minOccurs="0"
maxOccurs="255"/>
         <xs:element name="DateCode" type="stml:_String" minOccurs="0"
maxOccurs="255"/>
         <xs:element name="NumOfChannelGrouping" type="stml:UInt8"/>
         <xs:element name="GroupName" type="stml:_String" minOccurs="0"
maxOccurs="255"/>
         <xs:element name="ProductDescription" type="stml:_String"
minOccurs="0" maxOccurs="65535"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="Meta-IdentificationTEDSType">
      <xs:complexContent>
         <xs:extension base="stml:TextTEDSType">
            <xs:sequence>
               <xs:element name="XMLText" type="stml:Meta-
IdentificationTEDSDataBlockType"/>
               <xs:element name="XMLSum" type="stml:UInt16"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:schema>
```


Copyright © 2007 IEEE. All rights reserved.

D.6 TransducerChannelIdentificationTEDS.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include schemaLocation="SmartTransducerDataModel.xsd"/>
   <xs:include schemaLocation="TextTEDS.xsd"/>
   <xs:element name="TransducerChannelIdentificationTEDS"
type="stml:TransducerChannelIdentificationTEDSType"
substitutionGroup="stml:TextTEDS">
      <xs:annotation>
         <xs:documentation>1451 Smart Sensor
TransducerChannelIdentification TEDS schema</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:complexType
name="TransducerChannelIdentificationTEDSDataBlockType">
      <xs:sequence>
         <xs:element name="ManufactruereID" type="stml:_String"/>
         <xs:element name="ModelNo" type="stml:_String" minOccurs="0"
maxOccurs="255"/>
         <xs:element name="VersionCode" type="stml:_String"
minOccurs="0" maxOccurs="255"/>
         <xs:element name="SerialNo" type="stml:_String" minOccurs="0"
maxOccurs="255"/>
         <xs:element name="ChannelDescription" type="stml:_String"
minOccurs="0" maxOccurs="65535"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="TransducerChannelIdentificationTEDSType">
      <xs:complexContent>
         <xs:extension base="stml:TextTEDSType">
            <xs:sequence>
               <xs:element name="XMLText"
type="stml:TransducerChannelIdentificationTEDSDataBlockType"/>
               <xs:element name="XMLSum" type="stml:UInt16"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:schema>
```

D.7 CalibrationIdentificationTEDS.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include schemaLocation="SmartTransducerDataModel.xsd"/>
   <xs:include schemaLocation="SmartTransducerTEDS.xsd"/>
   <xs:include schemaLocation="TextTEDS.xsd"/>
   <xs:element name="CalibrationIdentificationTEDS"
type="stml:CalibrationIdentificationTEDSType"
substitutionGroup="stml:TextTEDS">
      <xs:annotation>
         <xs:documentation>1451 Smart Sensor
CalibrationIdentificationTEDS schema</xs:documentation>
      </xs:annotation>
   </xs:element>
```


Copyright © 2007 IEEE. All rights reserved.

```
   <xs:complexType name="CalibrationIdentificationTEDSDataBlockType">
      <xs:sequence>
         <xs:element name="CalLabNo" type="stml:_String"/>
         <xs:element name="CalTech" type="stml:_String" minOccurs="0"
maxOccurs="255"/>
         <xs:element name="CalibrationDescrption" type="stml:_String"
minOccurs="0" maxOccurs="65535"/>
         <xs:element name="CalDate" type="stml:_String"/>
         <xs:element name="CalProc" type="stml:_String" minOccurs="0"
maxOccurs="255"/>
         <xs:element name="CalStd" type="stml:_String" minOccurs="0"
maxOccurs="255"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="CalibrationIdentificationTEDSType">
      <xs:complexContent>
         <xs:extension base="stml:TextTEDSType">
            <xs:sequence>
               <xs:element name="XMLText">
                  <xs:complexType>
                     <xs:complexContent>
                        <xs:extension
base="stml:CalibrationIdentificationTEDSDataBlockType"/>
                     </xs:complexContent>
                  </xs:complexType>
               </xs:element>
               <xs:element name="XMLSum" type="stml:UInt16"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:schema>
```

D.8 CommandsTEDS.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include schemaLocation="SmartTransducerDataModel.xsd"/>
   <xs:include schemaLocation="TextTEDS.xsd"/>
   <xs:element name="CommandTEDS" type="stml:CommandTEDSType"
substitutionGroup="stml:TextTEDS">
      <xs:annotation>
         <xs:documentation>1451 Smart Sensor CommandTEDS
schema</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:complexType name="CommandTEDSDataBlockType">
      <xs:sequence>
         <xs:element name="Name" type="stml:_String"/>
         <xs:element name="CmdClass" type="stml:UInt8"/>
         <xs:element name="CmdFunction" type="stml:UInt8"/>
         <xs:element name="Scope" type="stml:_String"/>
         <xs:element name="arg" minOccurs="0" maxOccurs="16535">
            <xs:annotation>
               <xs:documentation>Each arg element is one element in the
argumentArray that will be sent as part of the
command.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
```


Copyright © 2007 IEEE. All rights reserved.

```
               <xs:choice>
                  <xs:element name="TagValuePair" minOccurs="0"
maxOccurs="unbounded">
                     <xs:complexType mixed="true">
                        <xs:choice>
                           <xs:element name="Int8Value" type="stml:Int8"
minOccurs="0"/>
                           <xs:element name="UInt8Value"
type="stml:UInt8" minOccurs="0"/>
                           <xs:element name="Int16Value"
type="stml:Int16" minOccurs="0"/>
                           <xs:element name="UInt16Value"
type="stml:UInt16" minOccurs="0"/>
                           <xs:element name="Int32Value"
type="stml:Int32" minOccurs="0"/>
                           <xs:element name="UInt32Value"
type="stml:UInt32" minOccurs="0"/>
                           <xs:element name="floatValue"
type="stml:Float32" minOccurs="0"/>
                           <xs:element name="doubleValue"
type="stml:Float64" minOccurs="0"/>
                           <xs:element name="timeValue" minOccurs="0">
                              <xs:complexType>
                                 <xs:simpleContent>
                                    <xs:extension base="xs:dateTime"/>
                                 </xs:simpleContent>
                              </xs:complexType>
                           </xs:element>
                           <xs:element name="booleanValue"
type="stml:_Boolean" minOccurs="0"/>
                           <xs:element name="StringArg"
type="stml:_String" minOccurs="0"/>
                        </xs:choice>
                        <xs:attribute name="Tag" type="xs:string"
use="required"/>
                     </xs:complexType>
                  </xs:element>
                  <xs:element name="listOfValues" minOccurs="0"
maxOccurs="unbounded">
                     <xs:complexType>
                        <xs:choice>
                           <xs:element name="int8ArrayValue"
type="stml:Int8Array" minOccurs="0"/>
                           <xs:element name="uInt8ArrayValue"
type="stml:UInt8Array" minOccurs="0"/>
                           <xs:element name="int16ArrayValue"
type="stml:Int16Array" minOccurs="0"/>
                           <xs:element name="uInt16ArrayValue"
type="stml:UInt16Array" minOccurs="0"/>
                           <xs:element name="int32ArrayValue"
type="stml:Int32Array" minOccurs="0"/>
                           <xs:element name="uInt32ArrayValue"
type="stml:UInt32Array" minOccurs="0"/>
                           <xs:element name="float32ArrayValue"
type="stml:Float32Array" minOccurs="0"/>
                           <xs:element name="float64ArrayValue"
type="stml:Float64Array" minOccurs="0"/>
                           <xs:element name="stringArrayValue"
type="stml:StringArray" minOccurs="0"/>
                           <xs:element name="octetArrayValue"
type="stml:OctetArray" minOccurs="0"/>
                           <xs:element name="booleanArrayValue"
type="stml:BooleanArray" minOccurs="0"/>
```


Copyright © 2007 IEEE. All rights reserved.

```
                              <xs:element name="timeInstanceArrayValue"
type="stml:TimeInstanceArrayType" minOccurs="0"/>
                              <xs:element name="timeDurationArrayValue"
type="stml:TimeDurationArrayType" minOccurs="0"/>
                           </xs:choice>
                           <xs:attribute name="Tag" type="xs:string"
use="required"/>
                        </xs:complexType>
                     </xs:element>
                     <xs:element name="rangeOfValues" minOccurs="0">
                        <xs:complexType mixed="true">
                           <xs:choice>
                              <xs:element name="Int8Step" type="stml:Int8"
minOccurs="0"/>
                              <xs:element name="UInt8Step"
type="stml:UInt8" minOccurs="0"/>
                              <xs:element name="Int16Step"
type="stml:Int16" minOccurs="0"/>
                              <xs:element name="UInt16Step"
type="stml:UInt16" minOccurs="0"/>
                              <xs:element name="Int32Step"
type="stml:Int32" minOccurs="0"/>
                              <xs:element name="UInt32Step"
type="stml:UInt32" minOccurs="0"/>
                              <xs:element name="floatStep"
type="stml:Float32" minOccurs="0"/>
                              <xs:element name="doubleStep"
type="stml:Float64" minOccurs="0"/>
                              <xs:element name="timeStep"
type="xs:dateTime" minOccurs="0"/>
                              <xs:element name="logical"
type="stml:_Boolean" minOccurs="0"/>
                           </xs:choice>
                           <xs:attribute name="Tag" type="xs:string"
use="required"/>
                        </xs:complexType>
                     </xs:element>
                  </xs:choice>
                  <xs:attribute name="argName" type="xs:string"
use="required"/>
                  <xs:attribute name="argNumber" type="xs:unsignedByte"
use="required"/>
                  <xs:attribute name="dataModel" type="stml:DataModelTypes"
use="required"/>
                  <xs:attribute name="desc" type="xs:string"/>
               </xs:complexType>
            </xs:element>
            <xs:element name="reply" minOccurs="0" maxOccurs="65535">
               <xs:annotation>
                  <xs:documentation>Each reply element is one element in
the argumentArray that will be returned as part of the reply to the
command. The attribute replyArgOffset gives the number of octets from
the beginning of the reply message to the first octet in this
argument.</xs:documentation>
               </xs:annotation>
               <xs:complexType>
                  <xs:annotation>
                     <xs:documentation>This field provides an offset in
octets from the beginning of the reply message to the first octet of
this value.</xs:documentation>
                  </xs:annotation>
                  <xs:choice>
                     <xs:annotation>
```


Copyright © 2007 IEEE. All rights reserved.

```
                     <xs:documentation>This field provides an array of
octets. The exact structure and meaning of the elements of this array is
defined in the desc attribute or by reference to another
document.</xs:documentation>
                  </xs:annotation>
                  <xs:element name="timeValue" type="xs:dateTime"/>
                  <xs:element name="BitMappedOctet">
                     <xs:annotation>
                        <xs:documentation>The mask allows a bit or bits
to be selected from the reply argument. This can be repeated as many
times as required by specifying the same replyArgOffset repeatedly. A
one in a bit position in the mask selects the bit and a zero rejects
that bit.</xs:documentation>
                     </xs:annotation>
                     <xs:complexType>
                        <xs:simpleContent>
                           <xs:extension base="xs:unsignedByte">
                              <xs:attribute name="Mask"
type="stml:UInt8" use="required"/>
                           </xs:extension>
                        </xs:simpleContent>
                     </xs:complexType>
                  </xs:element>
                  <xs:element name="Int8Value" type="stml:Int8"/>
                  <xs:element name="UInt8Value" type="stml:UInt8"/>
                  <xs:element name="Int16Value" type="stml:Int16"/>
                  <xs:element name="UInt16Value" type="stml:UInt16"/>
                  <xs:element name="Int32Value" type="stml:Int32"/>
                  <xs:element name="UInt32Value" type="stml:UInt32"/>
                  <xs:element name="floatValue" type="stml:Float32"/>
                  <xs:element name="doubleValue" type="stml:Float64"/>
                  <xs:element name="textValue" type="stml:_String"/>
                  <xs:element name="octetArrayValue"
type="stml:OctetArray"/>
                  <xs:element name="logicalValue" type="stml:_Boolean"/>
               </xs:choice>
               <xs:attribute name="replyArgName" type="xs:string"
use="required"/>
               <xs:attribute name="replyArgOffset" type="xs:unsignedInt"
use="required"/>
               <xs:attribute name="desc" type="xs:string"/>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ArgType">
      <xs:sequence>
         <xs:element name="DataModel">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Type" type="xs:short"/>
                  <xs:element name="Length" type="xs:short"
default="1"/>
                  <xs:element name="Value" type="stml:UInt8"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="description">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Type" type="xs:short"/>
                  <xs:element name="Length" type="xs:short"/>
                  <xs:element name="Value" type="stml:_String"/>
               </xs:sequence>
```


Copyright © 2007 IEEE. All rights reserved.

```
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="CommandTEDSType">
      <xs:complexContent>
         <xs:extension base="stml:TextTEDSType">
            <xs:sequence>
               <xs:element name="XMLText"
type="stml:CommandTEDSDataBlockType"/>
               <xs:element name="XMLSum" type="stml:UInt16"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:simpleType name="DataModelType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="Int8"/>
         <xs:enumeration value="UInt8"/>
         <xs:enumeration value="Int8Array"/>
         <xs:enumeration value="UInt8Array"/>
         <xs:enumeration value="Int16"/>
         <xs:enumeration value="UInt16"/>
         <xs:enumeration value="Int16Array"/>
         <xs:enumeration value="UInt16Array"/>
         <xs:enumeration value="Int32"/>
         <xs:enumeration value="UInt32"/>
         <xs:enumeration value="Int32Array"/>
         <xs:enumeration value="UInt32Array"/>
         <xs:enumeration value="Float32"/>
         <xs:enumeration value="Float32Array"/>
         <xs:enumeration value="Float64"/>
         <xs:enumeration value="Float64Array"/>
         <xs:enumeration value="_String"/>
         <xs:enumeration value="StringArray"/>
         <xs:enumeration value="_Octet"/>
         <xs:enumeration value="OctetArray"/>
         <xs:enumeration value="_Boolean"/>
         <xs:enumeration value="BooleanArray"/>
         <xs:enumeration value="TimeInstance"/>
         <xs:enumeration value="TimeInstanceArray"/>
         <xs:enumeration value="TimeDuration"/>
         <xs:enumeration value="TimeDurationArray"/>
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```

D.9 LocationAndTitleTEDS.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include schemaLocation="SmartTransducerDataModel.xsd"/>
   <xs:include schemaLocation="TextTEDS.xsd"/>
   <xs:element name="LocationAndTitleTEDS"
type="stml:LocationAndTitleTEDSType" substitutionGroup="stml:TextTEDS">
      <xs:annotation>
         <xs:documentation>1451 Smart Sensor LocationAndTitle TEDS
schema</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:complexType name="LocationAndTitleDataBlockType">
```


Copyright © 2007 IEEE. All rights reserved.

```
      <xs:sequence>
         <xs:element name="LocationAndTitle" maxOccurs="255">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="URL4TEDS" type="stml:_String"/>
                  <xs:element name="TEDSTitle" type="stml:_String"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="LocationAndTitleTEDSType">
      <xs:complexContent>
         <xs:extension base="stml:TextTEDSType">
            <xs:sequence>
               <xs:element name="XMLText"
type="stml:LocationAndTitleDataBlockType"/>
               <xs:element name="XMLSum" type="stml:UInt16"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:schema>
```

D.10 UnitsExtensionTEDS.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include schemaLocation="SmartTransducerDataModel.xsd"/>
   <xs:element name="UnitsExtensionDataBlock">
      <xs:annotation>
         <xs:documentation>This is the schema for the contents of the
Text-based Units Extension TEDS</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="UnitsExtensionText" type="stml:_String"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```


Copyright © 2007 IEEE. All rights reserved.

Annex E

(informative)

Example Meta-Identification TEDS

E.1 Introduction

Text-based TEDS provide a mechanism for the manufacturer to embed textual information into a smart transducer. Any text-based TEDS has a structure that encapsulates the textual information as a "block" of arbitrary length. The TEDS structure accommodates multiple blocks to support presentation of the content in multiple languages. The only constraint imposed by this standard is that each block is an XML "instance document."

This annex provides a simple example of an instance document, implemented in English, suitable for inclusion in a Meta-Identification TEDS. Annex D provides the schema for the text-based TEDS in this example. The example provided below does not attempt to extend the schema.

The information content of any text-based TEDS is left to the discretion of the manufacturer. The example provided herein is a suggestive model for the type of information that a manufacturer might want to consider including in the Meta-Identification TEDS.

E.2 Example schema for the data block

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include schemaLocation="SmartTransducerDataModel.xsd"/>
   <xs:element name="MetaIdTEDSDataBlock">
      <xs:annotation>
         <xs:documentation>This is the schema for the contents of the
data block for the Meta Identification TEDS</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="manufacturerId" type="stml:_String"
minOccurs="0"/>
            <xs:element name="ModelNo" type="stml:_String"
minOccurs="0"/>
            <xs:element name="ProductDescription" type="stml:_String"
minOccurs="0"/>
            <xs:element name="serialNo" type="stml:_String"
minOccurs="0"/>
            <xs:element name="dateCode" type="stml:_String"
minOccurs="0"/>
            <xs:element name="versionCode" type="stml:_String"
minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```


Copyright © 2007 IEEE. All rights reserved.

E.3 Example instance document

```
<?xml version="1.0" encoding="UTF-8"?>
< MetaIdentificationTEDSDataBlock
xmlns="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI/Me
taIdentificationTEDSDataBlock.xsd">
   <manufacturerId>Acme Corp.</manufacturerId>
   <ModelNo>1036Z</ModelNo>
   <ProductDescription>4 Channnel Temperature TIM</ProductDescription>
   <serialNo>SNAB64</serialNo>
   <dateCode>9/05/2006</dateCode>
   <versionCode>V1.30</versionCode>
</MetaIdentificationTEDSDataBlock>
```

All elements in this instance document are optional. There is no schema constraint on the order in which these elements are specified. The description (line 4) may be terse or verbose.

Copyright © 2007 IEEE. All rights reserved.

Annex F

(informative)

Example TransducerChannel Identification TEDS

F.1 Introduction

Text-based TEDS provide a mechanism for the manufacturer to embed textual information into a smart transducer. Any text-based TEDS has a structure that encapsulates the textual information as a "block" of arbitrary length. The TEDS structure accommodates multiple blocks to support presentation of the content in multiple languages. The only constraint imposed by this standard is that each block is an XML "instance document."

This annex provides a simple example of an instance document, implemented in English, suitable for inclusion in a TransducerChannel Identification TEDS. Annex B provides the schema for the text-based TEDS in this example. The example provided below does not attempt to extend the schema.

The information content of any text-based TEDS is left to the discretion of the manufacturer. The example provided herein is a suggestive model for the type of information that a manufacturer might want to consider including in the TransducerChannel Identification TEDS.

F.2 Example schema for the data block

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include schemaLocation="SmartTransducerDataModel.xsd"/>
   <xs:element name="TransducerChannelIdDataBlock">
      <xs:annotation>
         <xs:documentation>This is the schema for the contents of the
data block for the TransducerChannel Identification
TEDS</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="manufacturerId" type="stml:_String"
minOccurs="0"/>
            <xs:element name="ModelNo" type="stml:_String"
minOccurs="0"/>
            <xs:element name="ChannelDescription" type="stml:_String"
minOccurs="0"/>
            <xs:element name="serialNo" type="stml:_String"
minOccurs="0"/>
            <xs:element name="versionCode" type="stml:_String"
minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```


Copyright © 2007 IEEE. All rights reserved.

F.3 Example instance document

```
<?xml version="1.0" encoding="UTF-8"?>
<TransducerChannelIdDataBlock
xmlns="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI/Tr
ansducerChannelIdDataBlock.xsd">
   <manufacturerId>Jones Pressure Products</manufacturerId>
   <ModelNo>1500AB</ModelNo>
   <ChannelDescription>Pressure channel</ChannelDescription>
   <serialNo>78529</serialNo>
   <versionCode>V7.00</versionCode>
</TransducerChannelIdDataBlock>
```

All elements in this instance document are optional. There is no schema constraint on the order in which these elements are specified. The description (line 2) may be terse or verbose.

Copyright © 2007 IEEE. All rights reserved.

Annex G

(informative)

Example Calibration Identification TEDS

G.1 Introduction

Text-based TEDS provide a mechanism for the manufacturer to embed textual information into a smart transducer. Any text-based TEDS has a structure that encapsulates the textual information as a "block" of arbitrary length. The TEDS structure accommodates multiple blocks to support presentation of the content in multiple languages. The only constraint imposed by this standard is that each block is an XML "instance document."

This annex provides a simple example of an instance document, implemented in English, suitable for inclusion in a Calibration Identification TEDS. Annex D provides the schema for the text-based TEDS in this example. The example provided below does not attempt to extend the schema.

Unlike other text-based TEDS, the information content of the Calibration Identification TEDS is determined by the calibration laboratory personnel, who may be external service providers. It is still within the discretion of the manufacturer to determine whether to include the resources needed to support this TEDS.

The example provided herein is a suggestive model for the type of information that might be considered for inclusion in the Calibration Identification TEDS.

G.2 Example schema for the data block

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include schemaLocation="SmartTransducerDataModel.xsd"/>
   <xs:element name="CalibrationIdentificationTEDSDataBlock">
      <xs:annotation>
         <xs:documentation>This is the schema for the contents of the
data block for the Calibration Identification TEDS</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="CalLabNo" type="stml:_String"
minOccurs="0"/>
            <xs:element name="CalTech" type="stml:_String"
minOccurs="0"/>
            <xs:element name="CalibrationlDescription"
type="stml: String" minOccurs="0"/>
            <xs:element name="CalDate" type="stml:_String"
minOccurs="0"/>
            <xs:element name="CalProc" type="stml:_String"
minOccurs="0"/>
            <xs:element name="CalStd" type="stml:_String"
minOccurs="0"/>
         </xs:sequence>
```


Copyright © 2007 IEEE. All rights reserved.

```
      </xs:complexType>
   </xs:element>
</xs:schema>
```

G.3 Example instance document

```
<?xml version="1.0" encoding="UTF-8"?>
<CalibrationIdentificationTEDSDataBlock
xmlns="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI/Ca
librationIdentificationTEDSDataBlock.xsd">
   <CalLabNo>Acurate Calibrations Inc</CalLabNo>
   <CalTech>C.P. Lee</CalTech>
   <CalibrationlDescription>I had difficulty setting the gain on IA 3.
Watch for excessive drift on next cal cycle</CalibrationlDescription>
   <CalDate>08/01/2002</CalDate>
   <CalProc>Calibrated per the procedures in D6-46289</CalProc>
   <CalStd>Ektron 8200, SN 24698</CalStd>
</CalibrationIdentificationTEDSDataBlock>
```


Copyright © 2007 IEEE. All rights reserved.

Annex H

(informative)

Example Commands TEDS

H.1 Introduction

Text-based TEDS provide a mechanism for the manufacturer to embed textual information into a smart transducer. Any text-based TEDS has a structure that encapsulates the textual information as a "block" of arbitrary length. The TEDS structure accommodates multiple blocks to support presentation of the content in multiple languages. The only constraint imposed by this standard is that each block is an XML "instance document."

This annex provides a simple example of an instance document, implemented in English, suitable for inclusion in a Commands TEDS. Annex D provides the schema for the text-based TEDS in this example. The example provided below does not attempt to extend the schema.

The information content of any text-based TEDS is left to the discretion of the manufacturer. The example provided herein is a suggestive model for the type of information that a manufacturer might want to consider including in the Commands TEDS. Conformance with this recommendation will posture the manufacturer to take advantage of the generic interactive tools that are expected to evolve from this work. An example of such a tool is suggested herein. Failure to comply with the recommendations herein will force the manufacturer to develop a custom tool set to invoke manufacturer-defined commands

H.2 Example situation

Assume that a TIM manufacturer has a proprietary implementation of a signal conditioner that distinguishes this product from all competitors. This front end has multiple stages of amplification with programmable gains and offsets. To support calibration and troubleshooting, the vendor wants to define the following commands:

SetGain	Set gain, PreAmp
AlternateSetGain	Set gain, PreAmp
SetOffset1	Set offset, PreAmp

This contrived example has an intentionally abbreviated list of commands, but it could be easily extended to cover the "multiple stages of amplification" suggested above.

The scenario described was carefully chosen to convey the fact that manufacturer-defined commands are not intended to be used in an operational sense. Rather, they are unique support for troubleshooting, calibration, or commissioning purposes.

H.3 Schema for the data block

```
<?xml version="1.0" encoding="UTF-8"?>
<mdCmd xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<xs:schema
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
```


Copyright © 2007 IEEE. All rights reserved.

```
xsi:noNamespaceSchemaLocation="http://grouper.ieee.org/groups/1451/0/145
1HTTPAPI/IEEE1451CommandsTEDSSchema.xsd">
   <name desc="This command is a manufacturer unique command and is used
set the gain of an amplifier.">setGain</name>
   <CmdClass>128</CmdClass>
   <CmdFunction>2</CmdFunction>
   <arg argName="GainValue" argNumber="0" dataModel="Int8" desc="This
amplifier has allowable gains of 1, 2, 4, 8, 16 or 32. Select the value
for the gain that you want to use and enter it into the argument array
for the command.">
      <TagValuePair Tag="gain=1">
         <Int8Value>0</Int8Value>
      </TagValuePair>
      <TagValuePair Tag="gain=2">
         <Int8Value>1</Int8Value>
      </TagValuePair>
      <TagValuePair Tag="gain=4">
         <Int8Value>2</Int8Value>
      </TagValuePair>
      <TagValuePair Tag="gain=8">
         <Int8Value>3</Int8Value>
      </TagValuePair>
      <TagValuePair Tag="gain=16">
         <Int8Value>4</Int8Value>
      </TagValuePair>
      <TagValuePair Tag="gain=32">
         <Int8Value>5</Int8Value>
      </TagValuePair>
   </arg>
   <reply replyArgName="Success/Fail" desc="The reply to this command
should contain the Success Fail flag and a length of one"
replyArgOffset="0">
      <Int8Value>0</Int8Value>
   </reply>
</mdCmd>
```

H.4 Example instance documents

```
 <?xml version="1.0" encoding="UTF-8"?>
<mdCmd xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<xs:schema
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
xsi:noNamespaceSchemaLocation="http://grouper.ieee.org/groups/1451/0/145
1HTTPAPI/IEEE1451CommandsTEDSSchema.xsd">
   <name desc="This is a manufacturer unique command is used to set gain
from a list of allowable gains.">SetGain</name>
   <CmdClass>128</CmdClass>
   <CmdFunction>2</CmdFunction>
   <arg argName="Select Gain" argNumber="0" dataModel="Int8" desc="This
command allows setting gains from a list of allowable gains. Decide what
gain you want and enter a value from the list in the argument for the
command">
      <listOfValues Tag="List of Gain Values">
         <UInt8Array>1,2,4,8,16,32</UInt8Array>
      </listOfValues>
   </arg>
   <reply replyArgName="Success/Fail" desc="The reply to this command
should contain the Success Fail flag and a length of one"
replyArgOffset="0">
      <Int8Value>0</Int8Value>
   </reply>
```


Copyright © 2007 IEEE. All rights reserved.

```
</mdCmd>

 <?xml version="1.0" encoding="UTF-8"?>
<mdCmd xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<xs:schema xmlns:stml=http://grouper.ieee.org/groups/1451/0/1451HTTPAPI
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
xsi:noNamespaceSchemaLocation="http://grouper.ieee.org/groups/1451/0/145
1HTTPAPI/IEEE1451CommandsTEDSSchema.xsd">
   <name desc="This is a manufacturer unique command is used to set an
offset in a signal conditioner.">SetOffset</name>
   <CmdClass>128</CmdClass>
   <CmdFunction>3</CmdFunction>
   <arg argName="Tag" argNumber="0" dataModel="Int8" desc="This command
allows setting an offset between -5v and + 5 V in 250 steps of 0.04
volts each. Decide what offset you want. Divide it by 0.04 and enter
that for the value in the argument for the command">
      <rangeOfValues Tag="Range of Offset Values">
         <Int8Step hiRange="125" loRange="-125"
stepSize="1">0</Int8Step>
      </rangeOfValues>
   </arg>
   <reply replyArgName="Success/Fail" desc="The reply to this command
should contain the Success Fail flag and a length of one"
replyArgOffset="0">
      <Int8Value>0</Int8Value>
   </reply>
</mdCmd>
```

It should be apparent that a generic software tool could be developed to display the set of commands implemented and to prompt the operator for a selection. Based on the command selected, the tool could prompt the operator for each argument. With the accumulated information, the tool could construct a properly structured command string, issue the command to the device, and read and properly display the result(s).

Copyright © 2007 IEEE. All rights reserved.

Annex I

(informative)

Example Location and Title TEDS

I.1 Introduction

Text-based TEDS provide a mechanism for the manufacturer to embed textual information into a smart transducer. Any text-based TEDS has a structure that encapsulates the textual information as a "block" of arbitrary length. The TEDS structure accommodates multiple blocks to support presentation of the content in multiple languages. The only constraint imposed by this standard is that each block is an XML "instance document."

This annex provides a simple example of an instance document, implemented in English, suitable for inclusion in a Location and Title TEDS. Annex D provides the schema for the text-based TEDS in this example..

The information content of any text-based TEDS is left to the discretion of the manufacturer. The example provided herein is a suggestive model for the type of information that a manufacturer might want to consider including in the Location and Title TEDS.

I.2 Example situation

Assume the Acme Smart Transducer Company has produced a TIM, Model M2260, with a signal conditioning suitable for acquisition at a fixed rate of 5 KHz. Also assume that this unit has a decimating digital filter (DDF) to provide effective acquisition rates of 1 KHz, 100 Hz, 10 Hz, and 1 Hz. This manufacturer realizes that to tailor his TransducerChannel for a customer who desires alternative acquisition rates, say 10 Hz rather than 100 Hz, involves a simple substitution of the digital filter coefficients. Thus, Acme implemented four manufacturer-defined TEDS as the mechanism to provide substitute coefficients.

I.3 Required schema for the data block

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include schemaLocation="SmartTransducerDataModel.xsd"/>
   <xs:element name="LocationAndTitleDataBlock">
      <xs:annotation>
         <xs:documentation>This is the schema for the contents of the
data block for the LocationAndTitleTEDS</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="URL4TEDS" type="stml:_String"
minOccurs="0"/>
            <xs:element name="TEDSTitle" maxOccurs="255">
               <xs:complexType>
                  <xs:simpleContent>
                     <xs:extension base="stml:UInt8">
```


Copyright © 2007 IEEE. All rights reserved.

```
                        <xs:attribute name="TEDSAccessCode"/>
                    </xs:extension>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

I.4 Example instance document

```
<?xml version="1.0" encoding="UTF-8"?>
<LocationAndTitleDataBlock
xmlns="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI/Lo
cationAndTitleTEDS.xsd">
   <URL4TEDS>http://www.AcmeCorp.com/1036Z/TEDS</URL4TEDS>
   <TEDSTitle TEDSAccessCode="129">5kHzDigitalFilterTEDS</TEDSTitle>
   <TEDSTitle TEDSAccessCode="130">1kHzDigitalFilterTEDS</TEDSTitle>
   <TEDSTitle TEDSAccessCode="131">100HzDigitalFilterTEDS</TEDSTitle>
   <TEDSTitle TEDSAccessCode="132">10HzDigitalFilterTEDS</TEDSTitle>
   <TEDSTitle TEDSAccessCode="133">1HzDigitalFilterTEDS</TEDSTitle>
</LocationAndTitleDataBlock>
```

Element URL4TEDS declares a network location, in the form of a URL, where the user can find electronic copies of all TEDS that are implemented as virtual TEDS, if any.

Element TEDSTitle provides a descriptive title for the TEDS. It has a single argument that identifies the TEDS Access Code for this TEDS. This element is repeated five times. Once for each TEDS.

Copyright © 2007 IEEE. All rights reserved.

Annex J

(informative)

Example Units Extension TEDS

J.1 Introduction

As described in 5.5.2.8 there are some conditions where it is necessary to add additional information to a set of Physical Units to understand what is being measured. This TEDS is provided to have a place to supply that information.

J.2 Example situation

A transducer has been produced that measures Moles of Chlorine. The Physical Units adequately identify the units as Moles but does not identify Moles of what. This example TEDS supplies text to identify the measurement as Moles of Chlorine.

J.3 Example schema for the data block

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
targetNamespace="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include schemaLocation="SmartTransducerDataModel.xsd"/>
   <xs:element name="UnitsExtensionDataBlock">
      <xs:annotation>
         <xs:documentation>This is the schema for the contents of the
Text-based Units Extension TEDS</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="UnitsExtensionText" type="stml:_String"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

J.4 Example instance document

```
<?xml version="1.0" encoding="UTF-8"?>
<UnitsExtensionDataBlock
xmlns="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://grouper.ieee.org/groups/1451/0/1451HTTPAPIUni
tsExtensionDataBlock.xsd">
   <UnitsExtensionText>of Chlorine</UnitsExtensionText>
</UnitsExtensionDataBlock>
```


Copyright © 2007 IEEE. All rights reserved.

Annex K

(informative)

Examples of Physical Units

The IEEE 1451.2 Working Group has developed a simple and easily stored method for identifying Physical Units when working with smart transducers, allowing them to provide output in terms any user can understand. This standard has taken that and expanded on it. This becomes even more necessary if the transducer must plug into any system and be useable without writing special software. The approach chosen is not without compromises, but it fits in almost any application.

There are two uses for the Physical Units in the transducer module. One purpose is to define what a sensor is measuring or what the output of an actuator represents. By reading the TransducerChannel TEDS, it can be determined that the TIM is designed to measure pressure, acceleration, etc. The other purpose is to identify the scaling that is associated with the TIM input or output. This allows the manufacturer to specify the calibration constants in the Calibration TEDS and to have the scaling known. However, just allowing the manufacturer to specify the calibration constants in the device does not allow the transducer to work in any system without writing special software. For example, what units should be used when calibrating a pressure transducer? It could be pascals or kilopascals or maybe megapascals. If you don't want some form of pascals, then how about using pounds per square inch (psi) or inches of mercury? Maybe a user would want inches of water or even inches of alcohol. All of these units are regularly used for pressure. There is no end to the possible units that different users want. To allow a manufacturer to build a transducer with built-in calibration constants, the Physical Units must be specified. In IEEE Std 1451.2-1997, these two functions were combined into the units fields in the Channel TEDS. Thsi standard also only uses a single units field in the TransducerChannel TEDS, but it has added fields to the Calibration TEDS to allow the calibration constants to represent a different set of units. See the paragraph entitled "System Considerations" near the end of this annex for a description of the use of these fields.

There are two basic approaches to specifying the units that the working group could have used. One choice would have been to try to provide a table of all possible units and to use an enumeration to select which one is being used in the transducer. The difficulty with this choice is that the total number of possible units is almost limitless. The table would be huge and still would not include all possible units. The other choice is to define a small set of basic units that can be combined to provide the required units. The standard then needs to find a way of specifying how to combine these basic units to provide the units for the particular transducer. Not all units that a user might desire can be provided in this fashion, but a set that is adequate for most transducers can be defined. The conversion between the units specified in the standard and what the user wants to see is usually a simple multiplication operation the system needing to display the data can handle.

The working group needed to specify a consistent set of units and to find a practical way to incorporate them into the transducer. Other standards bodies have addressed the problem of finding a consistent set of units. The IEEE 1451.2 Working Group selected an appropriate set and referred to it in the standard. The set selected is the International System of Units established in 1960 by the 11th General Conference on Weights and Measures. The abbreviation SI from the French Le System International d'Units is normally used to identify this set of units. It is based on the metric system of measurement used around the world.

Table K.1 gives the seven base quantities, assumed to be mutually independent, on which the SI is founded and the names and symbols of their respective units, called "SI base units." Definitions of the SI base units are given in *The International System of Units (SI)* [B7].

Copyright © 2007 IEEE. All rights reserved.

Table K.1—SI base units

Base quantity	Name	Symbol
Length	Meter	m
Mass	Kilogram	kg
Time	Second	s
Electric current	Ampere	A
Thermodynamic temperature	Kelvin	K
Amount of substance	Mole	mol
Luminous intensity	Candela	cd

Derived units are expressed algebraically in terms of base units. In the column labeled "Expression in terms of other SI units" of Table K.2, some of the units are in terms of other derived units. However, before they can be represented in accordance with the standard, they must be converted into the base units as shown in the column labeled "Expression in terms of SI Base units." The symbols for derived units are obtained by means of the mathematical operations of multiplication and division. For example, the derived unit for molar mass (mass divided by amount of substance) is the kilogram per mole that has the symbol kg/mol. Table K.2 contains examples of derived units expressed in terms of SI base units.

The Working Group decided to include the SI base units plus the derived units radian and steradian as base units for the standard. This gave a set of nine units from which the Physical Units required for the transducer are derived by the mathematical operations of multiplication and division. What remained was to devise a way to include the information in the transducer.

From an examination of Table K.2, it becomes apparent that any particular unit can be represented as the product of a set of the base units with each unit raised to the appropriate power. Since zero is a legitimate power, it can be seen that a particular unit can be represented by the product of all of the base units with each of the base units raised to the appropriate power. For example, for a sensor that measured distance, a user could write the units for this device in terms of the seven SI base units by writing the following:
$m^1\ kg^0\ s^0\ A^0\ K^0\ mol^0\ cd^0$.

However, units are seldom written with an exponent of "0," and if the exponent is "1," it is understood and not written. The units for the distance measuring device are written simply as "m." Computers need more explicit instructions such as a table that contains all of the exponents in a certain order. The example above could be written 1,0,0,0,0,0,0 and mean $m^1\ kg^0\ s^0\ A^0\ K^0\ mol^0\ cd^0$.

This is the basis of the method of representing Physical Units that the standard uses. The working group decided to add the two derived units radians and steradians to the seven SI base units. This gives only nine basic units for the standard. However, some devices measure the ratio of two quantities with the same units. For example, strain is in terms of meters per meter that is expressed as m/m, m^1m^{-1}, or m^0. This results in what is called a "dimensionless" quantity. It is still desirable to know that the device is measuring something in terms of a physical unit. This situation requires special handling in any system of units. Also, quantities are measured in terms of the logarithm of a quantity or in terms of the logarithm of a "dimensionless" ratio. All of these require some method of identifying them so that we can interpret them. Two categories of units that are not expressed in terms of the base units are the arbitrary units such as hardness and "digital data." The system must be able to identify these quantities. An identifier that defines classes of units can accomplish this.

Copyright © 2007 IEEE. All rights reserved.

Table K.2—Units derived from the SI base units

Derived quantity	Special name	Special symbol	Expression in terms of other SI units	Expression in terms of SI base units
Plane angle	Radian	rad		$m\ m^{-1} = 1$
Solid angle	Steradian	sr		$m^2\ m^{-2} = 1$
Frequency	Hertz	Hz		s^{-1}
Area (square meter)				m^2
Volume (cubic meter)				m^3
Acceleration (meter per second squared)				m/s^2
Wave number (reciprocal meter)				m^{-1}
Mass density(density) (kilogram per cubic meter)				kg/m^3
Specific volume (cubic meter per kilogram)				m^3/kg
Current density (ampere per square meter)				A/m^2
Magnetic field strength (ampere per meter)				A/m
Amount-of-substance concentration (mole per cubic meter)				Mol/m^3
Luminance (candela per square meter)				cd/m^2
Force	Newton	N		$m\ kg\ s^{-2}$
Pressure, stress	Pascal	Pa	N/m^2	$m^{-1}kg\ s^{-2}$
Energy, work, quantity of heat	Joule	J	N m	$m^2\ kg\ s^{-2}$
Power, radiant flux	Watt	W	J/s	$m^2\ kg\ s^{-3}$
Electric charge, quantity of electricity	Coulomb	C		s A
Electric potential, potential difference, electromotive force	Volt	V	W/A	$m^2kg\ s^{-3}A^{-1}$
Capacitance	Farad	F	C/V	$m^{-2}\ kg^{-1}\ S^4\ A^2$
Electric resistance	Ohm	Ω	V/A	$m^2\ kg\ s^{-3}\ A^{-2}$
Electric conductance	Siemens	S	A/V	$m^{-2}\ kg^{-1}\ s^3\ A^2$
Magnetic flux	Weber	Wb	V s	$m^2\ kg\ s^{-2}\ A^{-1}$
Magnetic flux density	Tesla	T	Wb/m^2	$kg\ s^{-2}\ A^{-1}$
Inductance	Henry	H	Wb/A	$m^2\ kg\ s^{-2}\ A^{-2}$
Celsius temperature	Degree Celsius	°C		K
Luminous flux	Lumen	lm		cd sr
Illuminance	Lux	lx	lm/m2	m^{-2} cd sr

The exponents can in theory be anything and should logically be represented as floating point numbers, but as a practical matter exponents almost always lie between ±4 and a resolution of 1/2 is adequate. Two related matters with nothing to do with units now come into play. The first is that the smallest unit in the IEEE Std 1451.2-1997 is the octet. The other is that there was no way defined in IEEE Std 1451.2-1997 to express a signed integer. Since no other area of that standard requires signed integers, the exponents are encoded using an unsigned octet integer. To give a resolution of ½, the exponent is multiplied by two. To represent the exponent as a signed quantity, 128 is added to the exponent after it is multiplied by two. This means that exponents between –64 and +63 can be expressed. In order to remain compatible with IEEE Std 1451.2-1997 and IEEE Std 1451.3-2003, this standard has not changed this representation.

K.1 Examples

Table K.3 gives the fields for the units for distance that is expressed in meters. The use of enumeration 0 signifies that the units are the product of the base units. Since the meter is a base unit only, the exponent for meters is nonzero.

Copyright © 2007 IEEE. All rights reserved.

Table K.3—Distance (m)

	Enum	rad	sr	m	kg	s	A	K	mol	cd
Exponent	0	0	0	1	0	0	0	0	0	0
Decimal		128	128	130	128	128	128	128	128	128

Table K.4 shows the table for the units when the quantity represents an area. The enumeration is zero to indicate that the units are derived by multiplying the exponents for the base units. Since only the base unit for distance is involved, the only non-zero exponent is for the meter. Since the units for area are m*m, the exponent is two.

Table K.4—Area (m^2)

	Enum	rad	sr	m	kg	s	A	K	mol	cd
Exponent	0	0	0	2	0	0	0	0	0	0
Decimal		128	128	132	128	128	128	128	128	128

Table K.5 shows the table for the measurement of pressure. The derived units for pressure are pascals. This involves three base units, the meter, the kilogram, and the second.

Table K.5—Pressure (pascals = m^{-1} kg $s^{-}2$)

	Enum	rad	sr	m	kg	s	A	K	mol	cd
Exponent	0	0	0	–1	1	–2	0	0	0	0
Decimal		128	128	126	130	124	128	128	128	128

The table for the representation of electrical resistance is given in Table K.6. As shown in the table, this unit requires four base units.

Table K.6—Resistance (ohms = m^2 kg s^{-3} A^{-2})

	Enum	rad	sr	m	kg	s	A	K	mol	cd
Exponent	0	0	0	2	1	–3	–2	0	0	0
Decimal		128	128	132	130	122	124	128	128	128

As shown in Table K.7, noise spectral density also involves four base units. The interesting thing about this unit is the use of the square-root function that causes the exponent for seconds to be –5/2.

Table K.7—Noise spectral density—volts per root Hertz (V/√Hz = m^2 kg $s^{-5/2}$ A^{-1})

	Enum	rad	sr	M	kg	s	A	K	mol	cd
Exponent	0	0	0	2	1	–5/2	–1	0	0	0
Decimal		128	128	132	130	123	126	128	128	128

Table K.8 describes the units for mass fraction. Mass fraction is a “dimensionless” quantity that is represented by mol/mol. Thus the enumeration 1 is used to specify a ratio of the same units.

Table K.8—Mass fraction (mol/mol)

	Enum	rad	sr	m	kg	s	A	K	mol	cd
Exponent	1	0	0	0	0	0	0	0	1	0
Decimal		128	128	128	128	128	128	128	130	128

Copyright © 2007 IEEE. All rights reserved.

Table K.9 describes the units for strain. Strain is a "dimensionless" quantity represented by m/m. Thus the enumeration 1 is used to specify a ratio of the same units. The remaining fields are the same as for the distance measurement in Table K.3.

Table K.9—strain (m/m)

	Enum	Rad	sr	m	kg	s	A	K	mol	cd
Exponent	1	0	0	1	0	0	0	0	0	0
Decimal		128	128	130	128	128	128	128	128	128

The measurement of radiated power is normally accomplished in decibels. As shown in Table K.10, the representation of this quantity using the base units uses the unit of bels that is ten decibels.

Table K.10—Power quantity—Bel (log10 W/W) W = m^2 kg s^{-3}

	Enum	Rad	sr	m	kg	s	A	K	mol	cd
Exponent	3	0	0	2	1	–3	0	0	0	0
Decimal		128	128	132	130	122	128	128	128	128

Suppose that a transducer is counting widgets as they pass a station on a conveyor belt. The appropriate unit for this transducer is "widgets." It is not possible to derive the unit "widgets" from the nine base units defined in the standard. For a "unitless" quantity, the standard requires that enumeration 0 be used to identify the units. The exponents would all be set to 0 as shown in Table K.11. Note that for this case some special software will be required to display the appropriate units.

Table K.11—Counts

	Enum	Rad	sr	m	kg	s	A	K	mol	cd
Exponent	0	0	0	0	0	0	0	0	0	0
Decimal		128	128	128	128	128	128	128	128	128

The output of the transducer could represent the settings of a bank of switches or a command to open or close a valve. The units may be things like "on," "off," "up," "closed," or almost anything else. These units cannot be expressed in terms of the nine base units defined in the standard. Since it does not represent a quantity, enumeration 4 is the proper choice. Table K.12 is an example of the units for this type of transducer. Special software will be needed in the user's system to handle this type of transducer.

Table K.12—Switch positions

	Enum	Rad	sr	m	kg	s	A	K	mol	cd
Exponent	4	0	0	0	0	0	0	0	0	0
Decimal		128	128	128	128	128	128	128	128	128

K.2 System considerations

The discussion above describes how the physical qaunity being measured or output can be identified but does not in many cases give the units that the operator wants to see displayed. This standard allows the conversion constants in the Calibration TEDS to be for any set of units that the operator wants. This leaves the issue of how can a computer determine what units the Calibration TEDS is using? To address this issue, two constants have been added to the Calibration TEDS. They are the "SI units conversion slope" and the "SI units conversion intercept." These two constants are the constants that are required to convert the output of the correction process, using the constants in the Calibration TEDS, into SI units. Now the process of determining what units are being used becomes a two-step process. The first step is to determine what is being measured or output. Analyzing the Physical Units in the TransducerChannel TEDS as described above allows this to be determined. The next step then is to look at the "SI units conversion slope." For all cases, except temperature, this constant will be different for each different set of units. For

Copyright © 2007 IEEE. All rights reserved.

example, consider possible units for pressure as listed in Table K.13. Many different sets of units are used to measure pressure. The SI units for pressure are m^{-1} kg s^{-2}, which is also called pascals. An examination of the constants in the "Conversion slope" column shows that there is a different constant for each different unit so that constant can be used to determine the units being output by the correction process. There is one problem with this table that can exist in any set of units. There is a case where the same unit is called by two different names. In the table this is true for "Millimeters of Mercury" and Torr. Nothing in the standard allows the system to determine which unit should be displayed in this case. That is up to the user.

Table K.13—Conversion constants for pressure

Units	Conversion slope
Pascals	1
Kilopascals	0.001
Pounds per square inch (PSI)	6894.757
BAR	100 000
MILLIBAR	100
Inches of water	249.082
Millimeters of water	9.80665
Inches of Mercury	3386.38
Millimeters of mercury	133.3224
Atomspheres	101 325
Kilograms per square centimeter	98066.5
Torr	133.3224

NOTE—Some of these conversion constants are temperature sensitive and will only be correct at a given temperature.

As noted, this seems to work for everything except temperature. An examination of Table K.14 reveals the problem. Both Kelvin and Celsius have the same slope. The same is true for Fahrenheit and Rankine. This requires that both the slope and the intercept be used to determine the units if temperature is what is being measured or output.

Table K.14—Conversion constants for temperature

Units	Conversion slope	Conversion intercept
Kelvin	1	0
Celsius	1	273.15
Fahrenheit	0.5555556	255.3722
Rankine	0.5555556	0

K.3 Conclusions

The standard has provided a consistent set of tools to address the issue of displaying data in a standardized way. By providing a way for a set of base units to be incorporated into the transducer, the Working Group has provided a standard way that the units are represented in the TEDS and a standard way that the calibration coefficients are represented. This means that a transducer that is built and calibrated per the standard should be able to plug into any system. That system should then be able to display most data without any other changes to the system.

K.4 Acknowledgments

Bruce Hamilton developed this method of representing Physical Units in a computer-based system with help from William Kent, Stephanie Janowski, and Dan Hepner. All were employees of Hewlett Packard Labs in Palo Alto, California. Hamilton has written a paper entitled "A Compact Representation of Physical Units" that gives the theory and a list of references for further study. The paper is available as Agilent Laboratories Technical Report HPL-96-61, 1995 from Aligent Technical Publications Department [B1].

Copyright © 2007 IEEE. All rights reserved.

Annex L

(informative)

TEDS read and write protocols

L.1 TEDS access

TEDS data structures may be quite lengthy. The command set established to read and write TEDS is purposely designed to avoid monopolizing the communications link for long periods of time. By necessity, the process of reading or writing a TEDS involves a systematic sequence of commands. The following sub-clauses describe processes that may be used to write or read the TEDS

L.2 First step in a TEDS access

The first step in writing or reading a TEDS is to issue the Query TEDS command for the particular TEDS. This command provides the following information:

— Whether it is allowable to overwrite an existing TEDS

— Whether this TEDS is supported

— The maximum size of the TEDS

— Whether this is a Text-based or binary TEDS

— Whether this TEDS is resident or virtual

— Whether the current contents of this TEDS are valid

— The current size of the TEDS

— The TEDS Checksum

— Whether the last image written to this TEDS would fit within the available memory

L.3 Writing a TEDS

Assuming that the result of the Query TEDS command indicates that the TEDS will be accepted by the TIM, the following steps should be taken:

— Take a portion of the TEDS (called a block) that is equal to or less than the maximum message size and send that block using a Write TEDS segment command. Upon receipt of a Write TEDS segment command, the TIM shall internally mark that TEDS as invalid if the permanent copy is being changed.

— Wait until a reply is received to the Write TEDS segment command from the TIM.

— Select the next appropriate block and issue another Write TEDS segment command.

— This process shall be repeated until the entire TEDS has been transmitted.

— After the TEDS data have been transmitted, issue an Update TEDS command to signal that the TEDS data have been written and wait for the reply to the Update TEDS command.

— The TIM shall then validate the full contents of the TEDS received. At minimum, this involves verifying the embedded checksum. If valid, the TIM completes the task of writing the TEDS into nonvolatile memory before replying to the Update TEDS command.

Copyright © 2007 IEEE. All rights reserved.

L.4 Reading a TEDS

If the TEDS is listed as ”Virtual,” it cannot be read using procedure detailed below, because the TEDS does not reside in the TIM. It becomes the obligation of the NCAP or the host processor to locate and read a virtual TEDS.

For a resident TEDS, provided that it is supported and its current contents are valid, entire TEDS may be read in a sequence of operations summarized as follows:

— Issue a Read TEDS segment command.

— Wait for the reply.

— Repeat the previous two steps until the entire TEDS is read.

The NCAP or host processor determines when to terminate the read process using the current TEDS size or some other mechanism. It is not necessary to read TEDS sequentially or completely. This is particularly important when reading text-based TEDS, from which the system may only be interested in the text presented in a specific language. However, it should be noted that except for text-based TEDS the reading of a partial TEDS will not allow the checksum to be used to verify that the TEDS was read correctly. For text-based TEDS, the header and each language block within the TEDS has a separate checksum. The TIM shall not return octets if data are requested beyond the end of the TEDS.

Copyright © 2007 IEEE. All rights reserved.

Annex M

(informative)

Trigger logic configurations

M.1 Trigger logic augmented with an embedded time delay actuator

In order to obtain precise simultaneous sampling across multiple TIMs, a TransducerChannel may be augmented with an embedded Time Delay Actuator to introduce a programmable delay between the time when the trigger message is received by the Data Transport Logic and when that trigger is passed to the Sample Control Logic. This process is shown in Figure M.1. This feature is optional, introduced at the discretion of the transducer manufacturer, to satisfy stringent requirements for simultaneous triggering of inputs or outputs. This implementation model may be used to compensate for propagation delays.

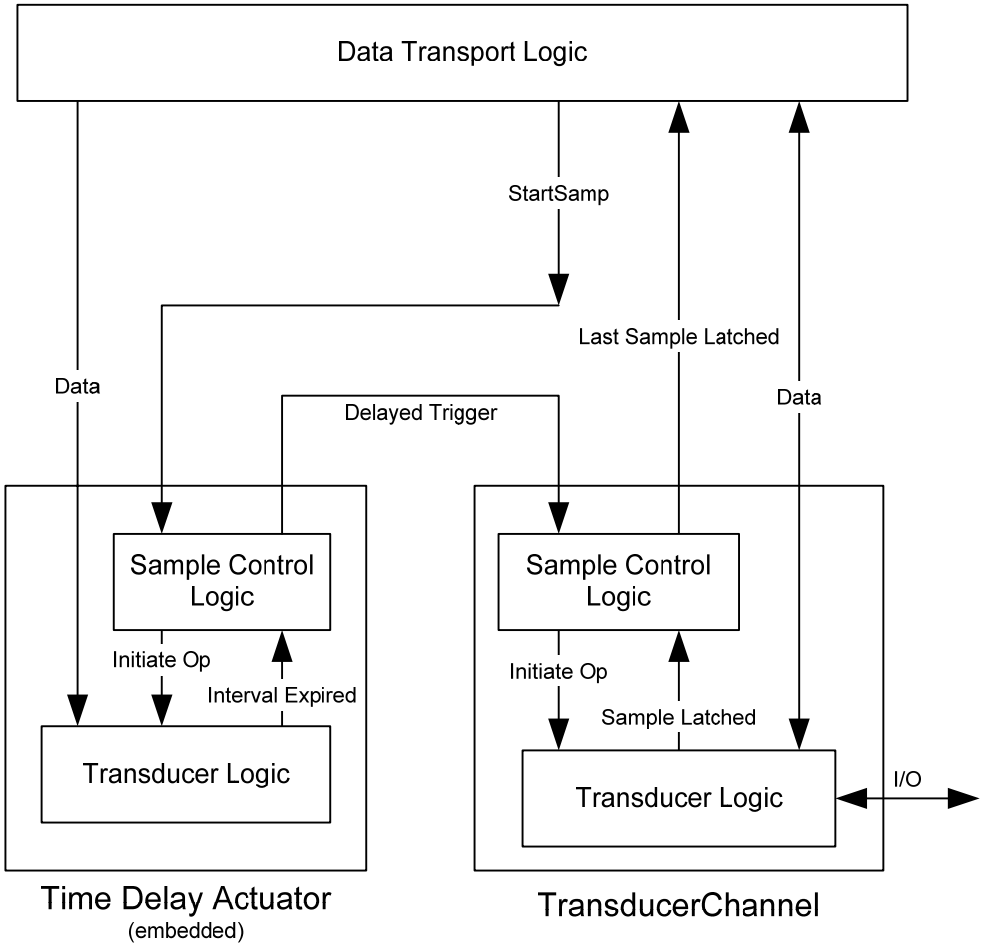


Figure M.1—TransducerChannel with a variable time delay on the trigger

Please note that the Time Delay Actuator is implemented as a TransducerChannel with a special hardware relationship to the primary TransducerChannel. It should be identified as a ChannelGroup in the Meta-TEDS. The manufacturer is not required to honor a trigger message addressed to the Time Delay Actuator.

Copyright © 2007 IEEE. All rights reserved.

As in a Nominal Triggering model, the Data Transport Logic responds to trigger messages. Upon receipt of a trigger message addressed to the primary TransducerChannel, provided that the TransducerChannel trigger is enabled, the Data Transport Logic issues a StartSamp signal, that is passed to the associated Time Delay Actuator. When an appropriate delay interval expires, that element commands the primary TransducerChannel to begin sampling.

This configuration still lacks hardware features at the TIM to assist the system to determine the time at which the sample was latched. The algorithm used to compute the time of sample is an extension of the algorithm previously discussed in 5.11, adjusted for the intentional delay introduced by the Time Delay Actuator. The time of sample is calculated as an adjustment to the time of day associated with the trigger message sent by the NCAP.

The features needed to actually measure the propagation delays are not discussed herein. Refer to the standards for specific members of the IEEE 1451 family for details. For now, assume that within a system, the propagation delay along the transducer communications medium can be dynamically determined. Then, each TIM has a measured propagation delay, PD that is unique for the TIM but constant for all TransducerChannels that reside in that TIM. Each TransducerChannel has a known value for the Incoming propagation delay through the data transport logic (tpd_i) that is specified in the TransducerChannel TEDS. The existence of an embedded Time Delay Actuator allows the system to introduce a compensatory delay to ensure precise simultaneous sampling across multiple TIMs on a transducer bus. More specifically, the system can program each Time Delay Actuator with a time delay TD_i such that ($PD_i + tpd_i + TD_i$) is a single constant value C for all TransducerChannels (for all values of i) in the collection of units being synchronized. Then the following calculation is used to obtain the time of the first sample in the data set:

$$T_1 = T_{tm} + C$$

where

T_{tm} is the time of day at which the trigger message was sent by the NCAP
C is a constant value, described above
T_1 is the time of day when the first sample in the data set was latched

The time of any other sample in the data set may be calculated using the algorithm previously discussed in 5.11.3.

M.2 Trigger logic augmented with TimeInstance sensor

When the nominal trigger model does not provide the required timing precision, the sensor may be augmented with an embedded TimeInstance sensor, as shown in Figure M.2.

Copyright © 2007 IEEE. All rights reserved.

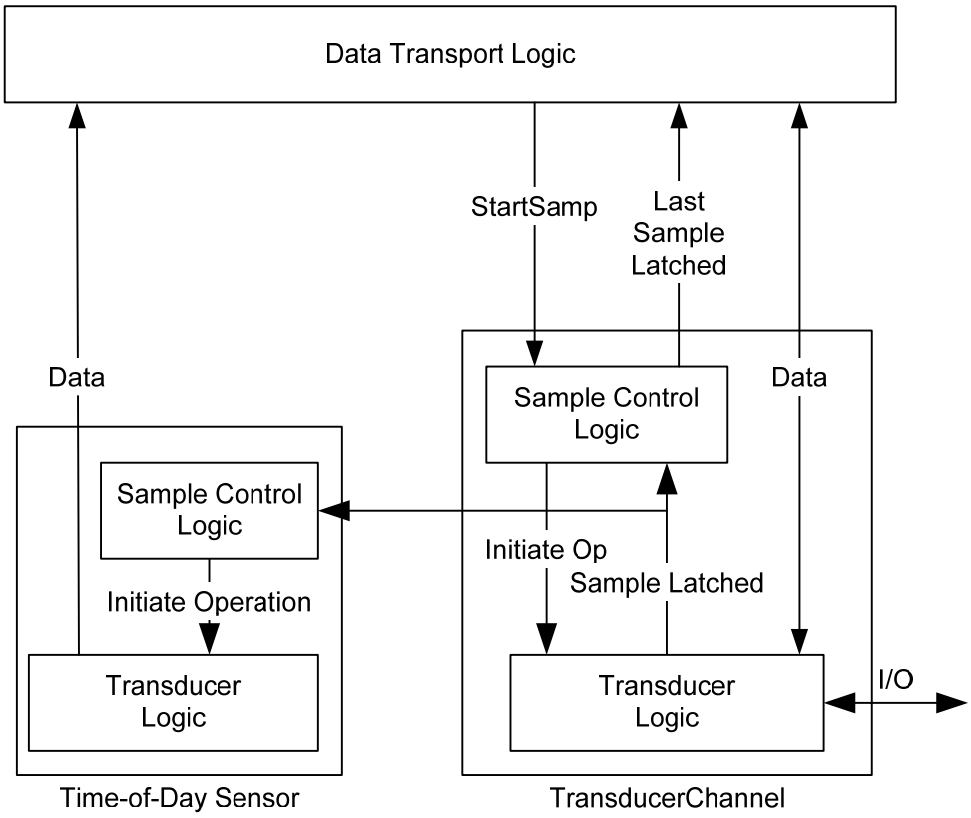


Figure M.2—TransducerChannel with a TimeInstance sensor

Figure M.2 shows a TimeInstance sensor used with a TransducerChannel. Please note that the TimeInstance Sensor is implemented as a TransducerChannel with a special hardware relationship to the primary TransducerChannel. It should be identified as a ChannelGroup in the Meta-TEDS. The manufacturer is not required to honor a trigger message addressed to the TimeInstance Sensor.

For TransducerChannels augmented with an embedded TimeInstance sensor, the TransducerChannel TEDS of the primary TransducerChannel should indicate in the Source for the time of sample field (see 8.5.2.19) that a TimeInstance sensor provides the time stamp. No further adjustments to the time value are required.

If the primary TransducerChannel is a sensor, the manufacturer may provide access to both resources through a TransducerChannel proxy. This has the advantage of bundling the data sets from both sensors into an aggregate data set that is read in a single transaction.

M.3 Trigger logic augmented with time interval sensor

When the nominal trigger model does not provide the required timing precision, the sensor may be augmented with an embedded time interval sensor. The method used should be specified in the Source for the time of sample field in the TransducerChannel TEDS (see 8.5.2.40). The time interval sensor is implemented as a TransducerChannel with a special hardware relationship to the primary TransducerChannel. It should be identified as a ChannelGroup in the Meta-TEDS. The manufacturer is not required to honor a trigger message addressed to the time interval sensor.

Copyright © 2007 IEEE. All rights reserved.

When the Source for the time of sample field indicates the presence of a time interval sensor interfaced to the trigger of the primary TransducerChannel, the time interval sensor has a relationship to the primary TransducerChannel as shown in Figure M.3.

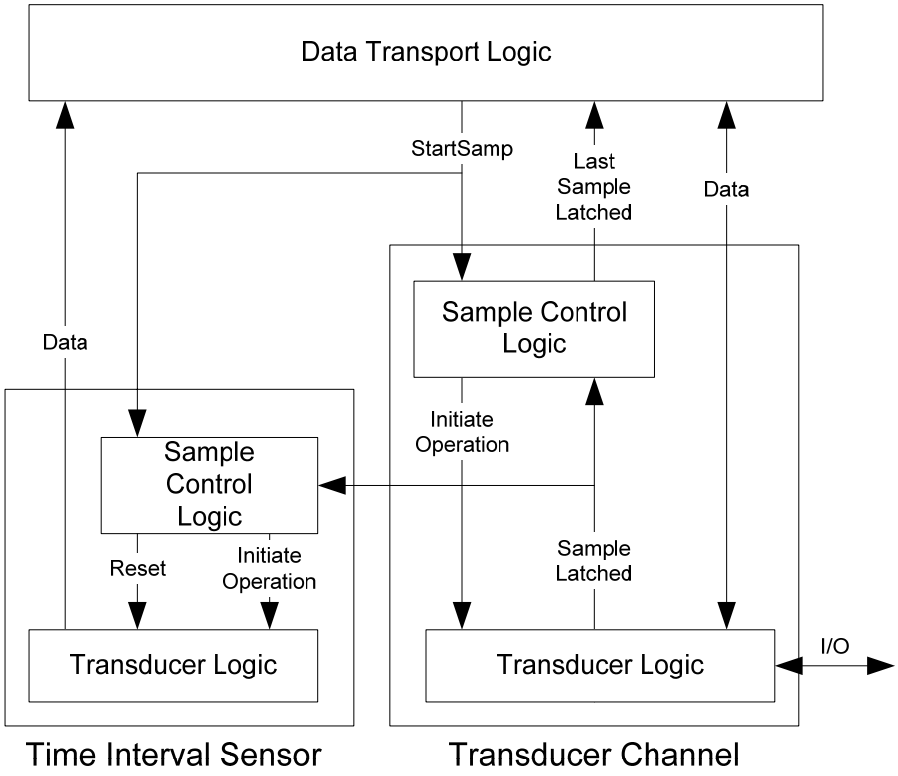


Figure M.3—TransducerChannel with a time interval sensor on sample(s)

The time interval sensor measures the interval between the time when a trigger message is received and the time that a sample is actually latched. Depending on the data model of the time interval sensor, as specified in the Maximum data repetitions field of the TransducerChannel TEDS, the manufacturer should implement this model in one of two ways.

When the Maximum data repetitions is set to 1, the manufacturer has chosen to provide a single time interval value for the entire data set, which suggests that the intersample spacing is uniform in time. In this case, the value reported should be the time interval associated with the first sample in the data set. The following calculation may be used:

$$T_1 = T_{tm} + t_{pdi} + t_{tis}$$

where

T_{tm} is the time of day at which the trigger message was sent by the NCAP
t_{pdi} is the value of the Incoming propagation delay through the data transport logic field found in the TransducerChannel TEDS
t_{tis} is the time interval reported by the time interval sensor
T_1 is the time of day when the first sample in the data set was latched

This time calculation gives the time of the first sample in the data set. The time of any other sample in the data set is easily calculated using an algorithm previously discussed in 5.11.3.

Copyright © 2007 IEEE. All rights reserved.

When the Maximum data repetitions (in the TransducerChannel TEDS, 8.5.2.28) of the time interval sensor is greater than zero and equal to the Maximum data repetitions of the associated TransducerChannel, the manufacturer has chosen to provide a time interval value for each sample in the data set, which suggests that the intersample spacing has some variation. The following calculation may be used:

$$T(i) = T_{tm} + t_{pdi} + t_{tis}(i)$$

where

T_{tm} is the time of day at which the trigger message was sent by the NCAP
t_{pdi} is the value of the Incoming propagation delay through the data transport logic field found in the TransducerChannel TEDS
$t_{tis}(i)$ is the time interval reported by the time interval sensor for sample (i)
$T(i)$ is the time of day when sample(i) was latched

If the primary TransducerChannel is a sensor, the manufacturer may provide access to both resources through a TransducerChannel proxy. This has the advantage of bundling the data sets from both sensors into an aggregate data set that is read in a single transaction.

Copyright © 2007 IEEE. All rights reserved.

Annex N

(informative)

Notation summary for IDL

The IEEE 1451.0 document uses an interface definition language to describe software interfaces used in the standard. The notation for these descriptions adheres to the Interface Definition Language (IDL) rules as standardized in the ISO 14750 document. Additional information about IDL may be available from the Object Management Group's website at www.omg.org/gettingstarted/omg_idl.htm.

N.1 Key Features of IDL

IDL is a language that provides the infrastructure to separate software interfaces from the software programming language used to implement an applications solution. The use of IDL allows the IEEE 1451.0 software interfaces to be described in an abstract way that is independent of the actual code that might be used to implement these interfaces. The use of IDL in this standard allows IEEE 1451.0 implementers to encapsulate their solutions so that only the IDL-defined interfaces are available to outside entities.

The OMG Interface Definition Language is particularly well suited to the needs of the IEEE 1451.0 layer in which complex data structures are compactly defined using several standard data types specified in IDL.

These features are particularly important when a standard such as the IEEE 1451.0 layer will be used in a heterogeneous environment potentially consisting of many different software languages, operating systems, and network protocols.

An important feature of IDL is that software interfaces described by the IDL notation are independent of the software language that might be used to implement a software application.

N.2 Example of IDL notation with explanation

The following is example of an IDL description of a software interface between a class that defines a time representation.

N.2.1 Example 1

From 4.9 of the this standard, we have selected an example IDL expression as shown in Figure N.1.

This segment of IDL is prefaced by initial comments for the reader. These optional comments are preceded by a double forward slash "//" (double solidus) as shown in this example.

In this example, the word "struct" is an IDL keyword that is a construct type specifier defined to associate the label "TimeRepresentation" with the data structure indicated in the "interface body" section of the IDL definition.

The body of the IDL expression is contained within the curly brackets "{… IDL interface body …}". Each statement within the curly brackets is terminated with a semicolon ";" character.

Copyright © 2007 IEEE. All rights reserved.

```
// This is an abstract class that defines the time representation. It is subclassed into
// TimeInstance and TimeDuration. The definition of the two arguments is given in
//Table 1"


 struct TimeRepresentation {
          UInt32 secs;
          UInt32 nsecs;
          };
```

Figure N.1—Simple example of IDL notation used to define a data structure containing two elements

The first IDL statement within the body of this IDL expression indicates that the "TimeRepresentation" data structure starts with an unsigned 32-bit integer that represents the seconds value and is labeled with the identifier "secs."

The second statement within the interface body of this IDL expression indicates that the second value in the "TimeRepresentation" data structure is an unsigned 32-bit integer that represents the nanoseconds value and is labeled with the identifier "nsecs."

The closing curly brackets indicate that the body of the IDL interface is completed.

N.2.2 Example 2

The IDL expressions illustrated in Figure N.2 includes a partial listing of the Module Communications Services description for the case of a "Point-to-Point" communications implementation.

After the initial comment line, the first line of this IDL definition is the module declaration statement. The "module" is used for variable scoping purposes to create a unique namespace so that identifiers defined within this module are only known within the ModuleCommunications interface. The contents of this module are delimited by left and right curly brackets "{ …}". The use of a module construct reduces scoping conflicts that can occur in large projects and reduces the need to create an abundance of globally defined names.

Following a few more lines of comment, the "interface" keyword identifies the section of IDL that will specify the software connections of a service implementation for clients that will use this service.

In this example, the "interface" keyword identifies the start of this description and indicates that the name of this interface will be "Comm." The right curly bracket "{" indicates the start of this interface body.

The first statement in the body of the Comm interface "Args::UInt16 init( );" is an "Operations Declaration." This statement shows that the init() operation has no parameters but returns an unsigned 16-bit integer. The Args::Uint16 indicates the return value of the init() operation.

The Args prefix followed by the double colon as "Args::___" indicates that the Uint16 identifier is defined and scoped in the "Args" module elsewhere in the document.

The Comm interface definition is terminated with a "}" left curly bracket and is followed by a comment introducing the next interface definition.

Copyright © 2007 IEEE. All rights reserved.

```
// ================ Module Communication Services ================

module ModuleCommunication

{

 // Abstract Comm interface. See P2PComm or NetComm below.

 // Provided by the IEEE1451.X implementation.

 // Used by the IEEE 1451.0 implementation.


 interface Comm

 {

  Args::UInt16 init( );

 };


 //  Point-to-Point Comm interface.


 interface P2PComm : Comm

 {

  Args::UInt16 read( in  Args::TimeDuration timeout,

                     in  Args::UInt32       maxLen,

                     out Args::OctetArray payload,

                     out Args::_Boolean   last );

 };
```

Figure N.2—IDL statements defining a module with two interface descriptions

The IDL statement “interface P2PComm : Comm” initiates the definition of the next interface definition. This interface is named “P2PComm” and inherits from the previously defined “Comm” interface. This creates the “P2PComm” interface as an interface that is derived from the base “Comm” interface. This allows the P2PComm interface to use the elements defined in the “Comm” interface as if these elements were defined in the “P2PComm” interface.

The next statement after the “P2PComm” interface statement “Args::UInt16 read(in Args::TimeDuration timeout, ….” declares the “read(…)” operation with the inputs and outputs for this operation. The text “Args:Uint16” prior to the “read(…)” text indicates that this “read” operation returns an unsigned 16-bit

Copyright © 2007 IEEE. All rights reserved.

integer, which typically indicates the success or failure of the read operation. In addition to the returned unsigned 16-bit integer value at the completion of the operation, the IDL definition further describes input and output parameters operated on by the "read" operation.

The statement "in Args::TimeDuration timeout" indicates that "read" operation requires an input parameter of type TimeDuration. This parameter is defined and scoped in the "Args" module and is identified as "timeout."

The balance of the "P2PComm" interface definition identifies the other input and output parameters required by "P2PComm." The second input parameter required for this interface is the "maxLen" identifier that is defined to be an unsigned 32-bit integer. This interface also includes descriptions of parameters that must be passed to the "read" operation but are filled by the read operation. The output values generated by this read operation will be assigned to the two identifiers "payload" and "last" as described in the last two statements of this IDL definition. As listed in Figure N.2, "payload" is an array of 8-bit octets referred to as an OctetArray and "last" is a Boolean value that may take on only one of two possible states—True or False.

The references at the beginning of this annex provide more complete description of IDL features and syntax. These references can be useful in learning more about the IDL notation used in this standard.

Copyright © 2007 IEEE. All rights reserved.

Annex O

(informative)

TEDS implementation of a simple sensor

Subclauses O.1 through O.4 describe the TEDS structures of a simple TIM. The TIM implements one channel of temperature information, sensed by a thermistor. Calibration is a simple linear regression within the operating range, and the channel is not triggered, but it will return a reading whenever queried. The sensor will output in ºC after corrections have been applied, and the operating range is –40 ºC to 80 º C ± 2 ºC.

Subclause 5.5 specifies that all TIMs shall implement the Meta-TEDS, TransducerChannel TEDS, User's Transducer Name TEDS, and PHY TEDS. The format of the PHY TEDS is beyond the scope of this standard and will not be described here. In addition, any transducer that does not output in calibrated Physical Units must also provide a Calibration TEDs. Since the output of the TransducerChannel is the 12 bit output of the ADC, a Calibration TEDS is supplied with output units of degrees Centigrade.

O.1 Meta-TEDS

The Meta-TEDS format is described in 8.4. This clause fills in the details of the structure described in Table 43 and Figure 13.

O.1.1 Meta-TEDS Identification

The format for the identification field is shown in Table O.1 (Table 41 is duplicated here.)

Table O.1—Meta-TEDS identifier

Field	Contents	Function
Type	03	Type field for the TEDS Identifier.
Length	04	This field is always set to 04 indicating that the value field contains 4 octets.
Family	00	This field identifies the member of the IEEE 1451 family of standards that defines this TEDS.
Class	01	This field identifies the TEDS being accessed. The value (found in Table 17) is 01 for the Meta-TEDS.
Version	01	This field identifies the TEDS version. The value is the version number identified in the standard. 01 indicates this TEDS conforms to the initial release of this standard.
Tuple Length	01	This field gives the number of octets in the length field of all tuples in the TEDS except this tuple. In this case, it is 01.

Converting this to TLV Octet format yields the following octet pattern:

03 04 00 01 01 01

O.1.2 UUID Field

The UUID is described in 4.12 and Table O.2. (Table O.2 is a duplicate of Table 4.)

Copyright © 2007 IEEE. All rights reserved.

Table O.2—Universal unique identification data type structure

Field	Description	Number of bits
1	Location field: The value of this field shall be chosen by the TIM manufacturer to identify a particular location on the earth, the "location,'" over which the manufacturer has physical control. This value may represent the actual location of a TIM manufacturer. A manufacturer may use different values of this field in his operation as long as all values meet the requirements of this subclause. The representation of the location field shall be 42 bits. The msb shall indicate North (asserted) or South (not asserted) latitude. The next 20 most significant bits of this field represent the magnitude of the "location" latitude as an integer number of arc seconds. The next most significant bit shall indicate East (asserted) or West (not asserted) longitude. The remaining 20 bits shall represent the magnitude of the "location" longitude as an integer number of arc seconds. Latitude magnitude values greater than 90 degrees are reserved. Longitude magnitude values of greater than 180 degrees are reserved. NOTE—One arc second at the equator is about 30 m. Thus, the range represented by each 20-bit field is 0 to 1 048 575 arcseconds or 0 to 291 degrees, which is sufficient to represent latitude and longitude on the earth's surface.	42
2	Manufacturer's field: The value of this field may be chosen by the TIM manufacturer for any purpose provided there is no interference in the use of the location field. Location field interference occurs when there is more than one manufacturer that could claim physical control over the location defined by the location field. If such interference exists, all manufacturers affected shall negotiate the use of the manufacturer's field values to resolve any interference. That is, the combination of the location field and the manufacturer's fields shall unambiguously define a specific TIM manufacturer. This negotiation shall be reopened every time there is a change in the interference.	4
3	Year field: The value of this field shall be the value of the current year. The representation of the year field shall be a 12 bit integer value. The range of this field shall be the years 0 to 4095 A.D. The beginning of the year shall be deemed to start on January 1, 00:00:00, TAI.	12
4	Time field: This value shall be chosen by the TIM manufacturer such that in combination with the location, manufacturer's, and year fields, the resulting UUID is unique for all TIMs made under the manufacturer's control. The choice of values for the time field shall be further restricted such that the values when interpreted as the time since the beginning of the year do not represent times prior to the manufacturer obtaining physical control over the "location" nor values in the future. The representation of the time field shall be a 22 bit integer. The range shall be 0 to 4 194 303. When it is necessary to interpret this field as time since the beginning of the year, the representation shall be an integer number of 10 s intervals. In this case, time values greater than one year are reserved. The beginning of the year shall be deemed to start on January 1, 00:00:00, TAI. NOTE—There are approximately 31 536 000 s per year.	22

Our TIM was produced on August 14, 2005 and was the 120th unit produced on that day. The TIM was produced at a manufacturing site located at the geospatial coordinates N40.3780 W105.0894.

The UUID will have the following values:

Location: The manufacturing plant is located at N40.3780 W105.0894. This translates to N145367 W381218 arcseconds.

There is only one manufacturer at this site, so the manufacturer's field is 0.

The Year field value is 2005.

Copyright © 2007 IEEE. All rights reserved.

The date of manufacture was August 14. The sensor was the 120th produced that day. There is some freedom in format, so this manufacturing plant uses the following format for the time field: day of the year × 1000 + sequence number (i.e., the sequential number of the unit produced that day). August 14 is the 224th day of the year, so the time field would be 224 × 1000 + 120 = 2 240 120.

Next, convert the fields to binary and combine them as shown in Table O.3.

Table O.3—UUID development

Field	Value	Binary value	Field width (bits)
N/S	N	1	1
Latitude	145367	00000011100000011111	20
E/W	W	0	1
Longitude	381218	01011101000100100010	20
Manufacturer	0	0000	4
Year	2005	011111010101	12
Date/Sequence	2240120	10001000101110011111000	22

Realigning and converting to octets yields the UUID format shown in Table O.4 (shown in hex).

Table O.4—UUID octets

Binary	Hex
10000001	81
11000000	C0
11111001	F9
01110100	74
01001000	48
10000001	81
11110101	F5
01100010	62
00101110	2E
01111000	78

Finally, this is converted into TLV format. The type field for the UUID is 04, and the length of the field is 10 octets. That causes the 2 octets 04 0A to be placed at the start of the block. Thus, the final form of this field in TLV octets is as follows:

04 0A 81 C0 F9 74 48 81 F5 62 2E 78

O.1.3 Operational timeout

The operational time-out for this TIM is 0.5 s. This is the time-out of normal operations. The number 0.5 will be represented as a F32 floating point number. Converting this into binary yields (see IEEE Std 754-1985 for details):

0 01111110 00000000000000000000000, which in turn yields the following octets in hex:

3F 00 00 00. Adding the field type of 10 (0A Hex) and the length of 4 yields the following final TLV octet pattern of

0A 04 3F 00 00 00

O.1.4 Slow-access time-out

The slow-access time-out is the same as the normal time-out, as there are no slow operations other than self-test (covered below). Therefore, this optional field is not included.

Copyright © 2007 IEEE. All rights reserved.

O.1.5 Self-test time-out

The self-test time-out for this TIM is 5.0 s. This is the time necessary to perform a self-test. The number 5.0 is represented as a F32 floating point number. Converting this to binary yields

1 10000001 01000000000000000000000, which in turn yields the following octets in hex:

C0 A0 00 00. Adding the field type of 12 and the length of 4 yields the following final TLV octet pattern of

0C 04 C0 A0 00 00

O.1.6 Number of TransducerChannels

This TIM contains a single channel. Therefore the value of this field is 1. The data type is U16E, and the field type is 13. This yields the final TLV octet pattern of

0D 02 00 01

O.1.7 Other fields

This TIM has only one channel, so the Meta-TEDS does not contain ControlGroup, VectorGroup, or Proxy Group Fields.

O.1.8 Final octet format of the Meta-TEDS

Finally we need to combine and add the total length information to the head of the structure. This is described in 8.1. The total number of octets in our Meta-TEDS is 36 (24 hex), including the 34 octets in the data block and the 2 checksum octets. The length field is 4 octets, so the octet value 00 00 00 24 is inserted.

The checksum is calculated by adding the octets of the data and length fields and taking the ones complement. This is described in 8.1.6. The sum of the octets is 0x72E, and the ones complement would be 0xF8D1.Our final Meta-TEDS in packed octet format is given in Table O.5.

Table O.5—Meta-TEDS example

MSB LSB	Field definition
00 00	Total length
00 24	
03 04	Header
00 01	
01 01	03 04 00 01 01 01
04 0A	UUID
81 C0	04 0A 81 C0 F9 74 48 81 F5 62 2E 78
F9 74	
48 81	
F5 62	
2E 78	
0A 04	Operational timeout
3F 00	0A 04 3F 00 00 00
00 00	
0C 04	Self-test timeout
C0 A0	0C 04 C0 A0 00 00
00 00	
0D 02	Number of channels
00 01	0D 02 00 01
F8 82	Checksum

Copyright © 2007 IEEE. All rights reserved.

O.2 TransducerChannel TEDS

The format for the TransducerChannel TEDS is described in 8.5.2, Table 49, and Figure 14.

O.2.1 TransducerChannel TEDS identification

Table O.6 lists the format for the identification field is used (from Table 41).

Table O.6—TransducerChannel TEDS identifier

Field	Contents	Function
Type	03	Type field for the TEDS Identifier.
Length	04	This field is always set to 04 indicating that the value field contains four octets.
Family	00	This field identifies the member of the IEEE 1451 family of standards that defines this TEDS.
Class	03	This field identifies the TEDS being accessed. The value (found in Table 17) is 03 for the TransducerChannel TEDS.
Version	01	This field identifies the TEDS version. The value is the version number identified in the standard, and 01 indicates this TEDS conforms to the initial release of this standard.
Tuple Length	01	This field gives the number of octets in the length field of all tuples in the TEDS except this tuple. In this case, it is 01.

Converting this to TLV Octet format yields the following octet pattern:

03 04 00 03 01 01

O.2.2 Calibration key

This TIM will provide calibration information that needs to be applied at the NCAP or application. The TIM itself will output ADC counts. The NCAP uses the calibration information to convert to temperature. This corresponds to a cal key value of 1 (CAL_SUPPLIED).

The field type for Calibration key is 10 (0A Hex). Converting this to the TLV octet format yields the following octet pattern:

0A 01 01

O.2.3 TransducerChannel type key

This TIM has one sensor channel, so the transducer type is 0 (Sensor). The field type for TransducerChannel Type is 11. Converting this to TLV Octet format yields the following octet pattern:

0B 01 00

O.2.4 Physical Units

This TIM outputs temperature in degrees Celsius after corrections are applied. The description of units is shown in 4.11. The first field of the UNIT descriptor is set to 0 (PUI_SI_UNITS). Fields 2–10 are set to 128, except field 8 (Kelvin), which is set to 130 to show linear temperature. Handling the conversion to degrees Celsius is handled in the Calibration TEDS. Thus the unit descriptor has the form listed in Table O.7.

Copyright © 2007 IEEE. All rights reserved.

Table O.7—Physical Units

Field	Description	TLV field type	Value	Hex
1	Physical Units interpretation—see Table 4	50 (32Hex)	0	0
2	(2 × <exponent of radians>) + 128	51 (33Hex)	128	80
3	(2 × <exponent of steradians>) + 128	52 (34Hex)	128	80
4	(2 × <exponent of meters>) + 128	53 (35Hex)	128	80
5	(2 × <exponent of kilograms>) + 128	54 (36Hex)	128	80
6	(2 × <exponent of seconds>) + 128	55 (37Hex)	128	80
7	(2 × <exponent of amperes>) + 128	56 (38Hex)	128	80
8	(2 × <exponent of kelvins>) + 128	57 (39Hex)	130	82
9	(2 × <exponent of moles>) + 128	58 (3AHex)	128	80
10	(2 × <exponent of candelas>) + 128	59 (3BHex)	128	80

The field type for Physical Units is 12. Because this is a temperature sensor only, Kelvins need to be entered in the TEDS. The values for all other base units default to 128. Converting this to TLV octet format yields the following octet pattern:

0C 06 32 01 00 39 01 82

O.2.5 Design operational lower limit

This TIM has an operational lower limit of –40 ºC. However, the operational lower limit must be specified in SI units of Kelvins. Therefore we must convert using the formula Deg-K = Deg-C + 273.15. Since this is an approximate limit, the field is simplified to have a value of 233. The field format is F32, so converting this to binary yields (see IEEE Std 754-1985 for details)

0 10000110 11010010000000000000000, which in turn yields the following octets in hex:

43 69 00 00. Adding the field type of 13 (0D Hex) and the length of 4 yields the following final TLV octet pattern of:

0D 04 43 69 00 00

O.2.6 Design operational upper limit

This TIM has an operational upper limit of 80 ºC. However, the operational upper limit must be specified in SI units of Kelvins. Therefore we must convert using the formula Deg-K = Deg-C + 273.15. Since this is an approximate limit, the field is simplified to have a value of 353. The field format is F32, so converting this to binary yields (see IEEE Std 754-1985 for details)

0 10000111 01100001000000000000000, which in turn yields the following octets in hex:

43 B0 80 00. Adding the field type of 14 (0E Hex) and the length of 4 yields the final TLV octet pattern of

0E 04 43 B0 80 00

O.2.7 Uncertainty under worst-case conditions

This TIM has a worst-case error of 2.0 ºC. Since this is an error, there is no need to convert to Kelvins. The field is F32, so converting this to binary yields

0 10000000 00000000000000000000000, which yields the following octets: 40 00 00 00. Adding the type field of 15 and length of 4 yields the final octet pattern of

0F 04 40 00 00 00

Copyright © 2007 IEEE. All rights reserved.

O.2.8 Self-test key

The TIM does have self-test available, so this field is set to 1. The field type is 16, so the octet pattern after conversion to hex is

10 01 01

O.2.9 Multi-range capability

This transducer does not have multirange capability, so this field is omitted.

O.2.10 Sample definition

Sample definition consists of three subfields. The final octet pattern will consist of the field type of 18 followed by the total length of the next 3 fields and their entire packets. The TIM outputs raw ADC counts from a 12 bit ADC.

O.2.11 Data model

The TIM outputs raw ADC counts, so the value of this field will be 0. This indicates an N-Octet integer. The field type is 40, so the final octet pattern is

28 01 00

O.2.11.1 Data model length

Since the output is from a 12 bit ADC, the field width will be 2 octets. The field type is 41, so the final octet pattern is

29 01 02

O.2.11.2 Model significant bits

The value of this field will be 12, corresponding to the 12 bit ADC. The field type is 42, so the final octet format is

2A 01 0C

O.2.12 Final octet format for sample definition

The combined length of the three subfields is 9. Combining these lengths and adding the proper length and field code for the sample definition (18) yields a final octet pattern of

12 09 28 01 00 29 01 02 30 01 0C

O.2.13 Data set definition

The TIM only produces single readings, so the repetition count is 0 and therefore this field can be omitted.

O.2.14 TransducerChannel update time

This transducer is in the free running sampling mode and is updating samples at a rate of 10/s. Therefore, the value of this field is 0.1, which is the maximum time between receipt of the read and having a sample available. The data type is F32, and the field code is 20, which yields a final octet pattern of

14 04 3D CC CC CD

Copyright © 2007 IEEE. All rights reserved.

O.2.15 TransducerChannel read setup time

The read setup time for this transducer is 25 μs. The field is F32, and the field code is 22, which yields a final octet pattern of

16 04 37 D1 B7 17

O.2.16 TransducerChannel sampling period

The sampling period is the same as the update time, i.e., 0.1 s. The field is F32, and the field code is 23, which yields a final octet pattern of

17 04 3D CC CC CD

O.2.17 TransducerChannel warm-up time

This transducer will take 30 s to warm up to specs. The field code is 24, and the parameter type is F32, which yields a final octet pattern of

18 04 41 F0 00 00

O.2.18 TransducerChannel read delay time

Worst-case delay of existing data is 25 μs. The field code is 25, and the format is F32, which yields a final octet pattern of

19 04 37 D1 B7 17

O.2.19 TransducerChannel self-test time requirement

The self-test time for this channel is the same as the total self-test time specified in the Meta-TEDS of 5 s. The field code is 26, and the format is F32, which yields a final octet pattern of

1A 04 40 A0 00 00

O.2.20 Source for time of sample

Since this sensor is free sampling and untriggered, the value of this field is 0, indicating that accurate sample latch time cannot be determined. Therefore, this field can be omitted.

O.2.21 Propagation delays and sampling uncertainty

Since the channel is untriggered and free sampling, these fields are irrelevant and are not included.

O.2.22 Sampling attribute

The transducer is never triggered and is always free running. The field code for the combined attribute is 31.

O.2.22.1 Sampling mode capability

The transducer is always running in the Free Running without Pre-trigger mode. Therefore, only bit 2 is set and the value of this setting is 2. The field code is 48, and the format is U8E, which yields a final octet pattern of

30 01 02

Copyright © 2007 IEEE. All rights reserved.

O.2.22.2 Default sampling mode

Since the only possible value for sampling mode is free run without pre-trigger, it is also the default and this field can be omitted.

O.2.22.3 Final sampling mode octet pattern

Combining the fields and adding a total length of 6 octets yields a final octet pattern of

1F 03 30 01 02

O.2.23 Buffered attribute

There is only one buffer of one reading in length. Thus the value of this field is 0 and may be omitted.

O.2.24 End of data set operation

This field is only applicable to actuators and is not included in this case.

O.2.25 Data transmission attribute

The simple temperature transducer will only return data when instructed and does not contain any streaming capability. Therefore this field may be omitted.

O.2.26 Remaining fields

The remaining fields apply to actuators, event sensors, or geospatial sensors and are not included in this TEDS.

O.2.27 Final TransducerChannel TEDS format

The final data fields for this TEDS, in order, are as follows:

```
03 04 00 03 01 01 (ID)
0A 01 01 (cal key)
0B 01 00 (type key)
0C 06 32 01 00 39 01 82 (units)
0D 04 43 69 00 00 (lower Limit)
0E 04 43 B0 80 00 (upper limit)
0F 04 40 00 00 00 (uncertainty)
10 01 01 (self test)
12 09 28 01 00 29 01 02 30 01 0C (sample definition)
14 04 3D CC CC CD (update time)
16 04 37 D1 B7 17 (read setup time)
17 04 3D CC CC CD (sampling period)
18 04 41 F0 00 00 (warmup time)
19 04 37 D1 B7 17 (read delay)
1A 04 40 A0 00 00 (self test time)
1F 03 31 01 02 (sampling)
```

The total length of the Channel TEDS, including 2 octets of checksum, is 95 octets. Combining the fields and adding 4 octets of length information yields the final format of this TEDS. The checksum is EF2C

```
00 00 00 5F 03 04 00 03 01 01
0A 01 01 0B 01 00 0C 06 32 01
00 39 01 82 0D 04 43 69 00 00
0E 04 43 B0 80 00 0F 04 40 00
00 00 10 01 01 12 09 28 01 00
```


Copyright © 2007 IEEE. All rights reserved.

```
29 01 02 30 01 0C 14 04 3D CC
CC CD 16 04 37 D1 B7 17 17 04
3D CC CC CD 18 04 41 F0 00 00
19 04 37 D1 B7 17 1A 04 40 A0
00 00 1F 03 31 01 02 EF 2C
```

O.3 Calibration TEDS

The TIM contains a single thermister that is providing a temperature reading. This thermister is placed into a voltage divider circuit and fed a 10 V signal into a 10 V a/d converter. The circuit characteristics provide a linear curve within the operating regions with the voltage characteristics of slope = 0.07625 and intercept = 2.4742. This needs to be converted to a/d counts. Our 12 bit, 10 V dac provides 4096 counts over a 10 V range, or 409.6 counts/V. Multiplying the slope and intercept values by this value provides a count slope of 312.32 and count intercept of 1013.43. Note that the intercept is shown in degrees Celsius, so it must be converted to Kelvins to show proper SI units. Since we wish to output in degrees Celsius, we will adjust this in the SI units clause.

We can use the simplified form of calibration shown in 8.6.3.10.

The Calibration TEDS for this TIM will contain the following fields:

O.3.1 Calibration TEDSidentification

Table O.8 lists the contents for the identification field (from Table 41).

Table O.8—Calibration TEDSidentifier

Field	Contents	Function
Type	03	Type field for the TEDS Identifier.
Length	04	This field is always set to 04 indicating that the value field contains 4 octets.
Family	00	This field identifies the member of the IEEE 1451 family of standards that defines this TEDS.
Class	05	This field identifies the TEDS being accessed. The value (found in Table 17) is 03 for the Calibration TEDS.
Version	01	This field identifies the TEDS version. The value is the version number identified in the standard, and 01 indicates this TEDS conforms to the initial release of this standard.
Tuple length	01	This field gives the number of octets in the length field of all tuples in the TEDS except this tuple.

Converting this to TLV Octet format yields the following octet pattern:

03 04 00 05 01 01

O.3.2 Last calibration date

This TIM was last calibrated as part of the manufacturing process, so the cal date is the same as the manufacturing date. Thus the date is August 14, 2005 and the time was 14:00:00. Placing this into the Time format described in 4.15 yields a total number of seconds since January 1, 1970 of 1124114400. Converting to hex and timeInstance yields a value of 43 00 9F E0 00 00 00 00.

The field header is 10, and the length is 8, which yields a final octet pattern of

0A 04 43 00 9F E0 00 00 00 00

Copyright © 2007 IEEE. All rights reserved.

O.3.3 Calibration interval

This TIM has a calibration interval of 1 year. Converting to seconds yields 365 × 24 × 60 × 60 = 31 536 000 or an octet value of 01 E1 33 80. Putting this into the timeDuration format yields 01 E1 33 80 00 00 00 00.

The field header is 11, and the length is 8, which yields a final octet pattern of

0B 08 01 E1 33 80 00 00 00 00.

O.3.4 SI units conversion constants

Since the cal data are going to output in degrees Celsius, we need to provide the SI conversion factor to Kelvins. Kelvins = deg C + 273.15. Thus the multiplier is 1, and the offset is +273.15.

O.3.4.1 Slope

Since the slope is 1, this field can be omitted and the default value can be assumed.

O.3.4.2 Intercept

The intercept is 273.15. The field code is 31, and the field type is F32. Therefore the final octet format of this field is as follows:

1F 04 43 88 93 33

O.3.4.3 SI units conversion final format

The field code for the combined field is 12, and the total length is 6. Thus the final octet format is as follows:

0C 06 1F 04 43 88 93 33

O.3.5 Upper and lower limits and uncertainty

The TIM will use the equivalent fields described in the Meta-TEDS. Therefore, these fields are omitted.

O.3.6 Pre- and post-conversion operations

No pre- or post-conversion operations are required. Therefore these fields are omitted.

O.3.7 Linear conversion method

This TEDS will be using the linear conversion method, which is represented by field code 20. Therefore, the TEDS will not include field codes 21, 22, or any of their subfields. It will need to specify the coefficients specified in 8.6.3.20 but will omit both the data channel and channel key as there is only one channel that only returns raw a/d counts.

O.3.7.1 Set of coefficients

The TEDS will be using a single set of coefficients representing slope and intercept. This is described in 8.6.3.20.2. Each of these values is an F32, and the pair represents an array of F32. The first value of slope is 312.32. Converting to F32 yields an octet value of 43 9C 28 F6. The intercept value is 1013.43. Converting to F32 yields an octet value of 44 7D 5B 85.

Copyright © 2007 IEEE. All rights reserved.

The field type is 51, and the combined length is 8, which yields a final octet pattern of

33 08 43 9C 28 F6 44 7D 5B 85

O.3.7.2 Final linear conversion octet format

The field code is 20, and the total length of the subfield is 10, which yields a final octet format of

14 0A 33 08 43 9C 28 F6 44 7D 5B 85

O.3.8 Final Calibration TEDSformat

No other fields apply to a simple linear regression calibration, so they are omitted. The fields included are as follows:

```
03 04 00 05 01 01 (header)
0A 08 43 00 9F E0 00 00 00 00 (last cal date)
0B 08 01 E1 33 80 00 00 00 00. (Cal Interval)
0C 06 1F 04 43 88 93 33 (SI Units Conversion)
14 0A 33 08 43 9C 28 F6 44 7D 5B 85 (Linear Conversion)
```

This yields a total length of 48 octets, including 2 octets of checksum. The checksum calculates out as F6 88. Therefore the final octet format of the Calibration TEDS is

```
00 00 00 30 03 04 00 05 01 01
0A 08 43 00 9F E0 00 00 00 00
0B 08 01 E1 33 80 00 00 00 00
0C 06 1F 04 43 88 93 33
14 0A 33 08 43 9C 28 F6 44 7D
5B 85 F6 88
```

O.4 User's transducer name TEDS

The User's Transducer Name TEDS is the last required TEDS.

O.4.1 User's transducer name TEDS identification

Table O.9 lists the contents of the identification field (from Table 41).

Table O.9—User's transducer name TEDS identification

Field	Contents	Function
Type	03	Type field for the TEDS Identifier.
Length	04	This field is always set to 04 indicating that the value field contains 4 octets.
Family	00	This field identifies the member of the IEEE 1451 family of standards that defines this TEDS.
Class	12	This field identifies the TEDS being accessed. The value (found in Table 17) is 12 for the User's Transducer Name TEDS.
Version	01	This field identifies the TEDS version. The value is the version number identified in the standard, and 01 indicates this TEDS conforms to the initial release of this standard.
Tuple length	01	This field gives the number of octets in the length field of all tuples in the TEDS except this tuple.

Converting this to TLV Octet format yields the following octet pattern:

03 04 00 0C 01 01

Copyright © 2007 IEEE. All rights reserved.

O.4.2 Format

To simplify implementation, this transducer will use a user-defined text field. Therefore, the value of this field will be 0. The field code is 4, and therefore, the final octet format for this field is

04 01 00

O.4.3 TCName

The TCName will contain the model number of this TIM. In this example the model number will be "ACME-100." Since this is user dependent, there is no TLV tagging the octet pattern for this field is simply the ASCII character values of

41 43 4D 45 2D 31 30 30

O.4.4 Final user's transducer TEDS format

The combined fields are as follows:

```
03 04 00 0C 01 01 (header)
04 01 00 (format)
41 43 4D 45 2D 31 30 30 (TCName)
```

The total length of this TEDS is 19 octets, including the 2 octets of checksum information. The checksum is FE 2E, which yields a final octet pattern for this TEDS of

```
00 00 00 13 03 04 00 0C 01 01
04 01 00 41 43 4D 45 2D 31 30
30 FD FE
```

O.5 Required commands

This TIM will implement the minimum command set required to operate as a legal IEEE 1451 sensor.

O.5.1 Commands common to the TIM and TransducerChannel

Table O.10 lists the required commands that will be implemented as shown in Table 16. (Table O.10 reproduces most of Table 16.)

Table O.10—Required commands

Function	Command	State		Reply expected	Required/ optional
		TransducerChannel	TIM		
0	Reserved	—	—	—	—
1	Query TEDS	Any	Active	Yes	Required
2	Read TEDS segment	Any	Active	Yes	Required
3	Write TEDS segment	Any	Active	No	Required
4	Update TEDS	Any	Active	Yes	Required
5	Run self-test	Idle	Active	No	Required
6	Write service request mask	Any	Active	No	Required
7	Read service request mask	Any	Active	Yes	Required
8	Read status-event register	Any	Active	Yes	Required
9	Read status-condition register	Any	Active	Yes	Required
10	Clear status-event register	Any	Active	No	Required
11	Write status-event protocol state	Idle	Active	No	Required
12	Read status-event protocol state	Any	Active	Yes	Required

Copyright © 2007 IEEE. All rights reserved.

O.5.1.1 Query TEDS command

This command will return the status of the specified TEDS. The single parameter to this command contains the TEDS identifier. This TIM will accept the values of 1, 3, 5, 12, and 13. If any other value is received, the command will return zeroes for fields, except the attribute field. The attribute field will return a UINT8 with bits 1 and 2 set and all other bits cleared.

A query of the Meta-TEDS, PHY TEDS, and transducer channel TEDS shall return attributes with the read-only bit set. The Calibration TEDS and User's Transducer Name TEDS shall return attributes with the read-only bit cleared. All other fields will be implemented as described in 7.1.1.1. Note that only channel 1 will be allowed when addressing the TransducerChannel TEDS and Calibration TEDS and that channel 0 will be required for the other allowable TEDS.

O.5.1.2 Read/write/update TEDS segment

These commands are implemented as described in 7.1.1.2, 7.1.1.3, and 7.1.1.4.

Again, the only allowable channel numbers are 1 for the Calibration and TransducerChannel TEDS and 0 for the other allowable TEDS.

O.5.1.3 Run self-test

This command will accept only channel 0 to test the entire TIM. The test will consist of communications, A/D, and test of thermistor connection.

O.5.1.4 Read/write service rRequest mask.

These commands will read or set the service request mask for either the entire TIM or for the Channel 1.

All bits in the service request mask will be writable in the mask. However, not all status bits will be implemented in the event register. The following bit definitions are notable.

O.5.1.4.1 Trigger acknowledged bit

The TIM does not accept triggers, so the trigger-acknowledged bit will never be set for either the channel or the TIM.

O.5.1.4.2 Missed data or event bit

When read data command is sent to Channel 1, an acquisition will take place before data are returned. If another read data command is received before the previous operation has completed, the missed data bit will be set for both the Channel and the TIM.

O.5.1.4.3 Bits not implemented

The following bits are not applicable to the TIM and the event register will always return 0:

Data/event

Hardware error

Data available/data processed

Corrections disabled

Consumables exhhausted

Not first read of data set

Copyright © 2007 IEEE. All rights reserved.

O.5.1.5 Read status event/condition register

These commands are implemented as described.

O.5.1.6 Clear status event register

Implemented as described.

O.5.1.7 Read/write status event protocol state

This event will not provide status streams, so these commands are not implemented.

O.5.2 Transducer idle state commands

Table O.11 lists tha idle state commands that are implemented in this TIM.

Table O.11—Transducer idle state commands

Function	Command	Address			Reply expected	Required/ optional
		TransducerChannel	Proxy	Group		
3	AddressGroup definition	Yes	Yes	No	No	Required
10	Calibrate TransducerChannel	Yes	Yes	Yes	Yes	Optional

O.5.2.1 AddressGroup definition

This command will allow Channel 1 to be assigned to the specified AddressGroup. All other channels will cause an error to be generated.

O.5.2.2 Calibrate TransducerChannel

This command will cause a calibration of the sensor to occur.

O.5.3 Transducer operating state commands

Table O.12 lists the commands that this transducer will execute when it is in the operating state.

Table O.12—Transducer operating state commands

Function	Command	Address			Reply expected	Required/ optional
		TransducerChannel	Proxy	Group		
1	Read TransducerChannel data-set segment	Yes	Yes	No	Yes	See note
3	Trigger command	Yes	Yes	Yes	No	Required

O.5.3.1 Read TransducerChannel data set segment

The parameters to this command will be restricted to 0 for offset and 1 for length. All other values will cause a command error to be generated. When this command is received, the channel will acquire data from the a/d and will return the data in response to the command.

O.5.3.2 Trigger

This command is required, but it will be treated as a NOP.

Copyright © 2007 IEEE. All rights reserved.

O.5.4 Transducer idle or operating state commands

The commands listed in Table O.13 will be implemented as described in 7.1.4, except for the Read TransducerChannel trigger state.

Table O.13—Transducer idle or operating state commands

Function	Command	Address			Reply expected	Required/ optional
		TransducerChannel	Proxy	Group		
1	TransducerChannel Operate	Yes	Yes	Yes	No	Required
2	TransducerChannel Idle	Yes	Yes	Yes	No	Required
4	Read TransducerChannel trigger state	Yes	Yes	Yes	No	Required
7	Read AddressGroup assignment	Yes	Yes	No	Yes	Required

O.5.4.1 Read TransducerChannel Trigger State

This command will always return False.

O.5.5 TIM sleep state commands

The wake-up command as listed in Table O.14 will be executed as described in 7.1.5.1.

Table O.14—TIM sleep state commands

Function	Command	Address		Reply expected	Required/ optional
		TIM	Global		
1	Wake-up	Yes	No	No	Optional

O.5.6 TIM active state commands

The active state commands listed in Table O.15 will be implemented as described in 7.1.6.

Table O.15—TIM active state commands

Function	Command	Address		Reply expected	Required/ optional
		TIM	Global		
1	Read TIM version	Yes	No	Yes	Required
2	TIM Sleep	Yes	No	No	Optional
3	Store operational setup	Yes	No	No	Required
4	Recall operational setup	Yes	No	No	Required
5	Read IEEE 1451.0 version	Yes	No	Yes	Required

O.5.6.1 Read TIM version

This command will always return a value of 1 for the initial release of the TIM.

O.5.6.2 TIM sleep

This command is implemented as described in 7.1.6.2. Since this is a wireless TIM, a power OFF mode is important for battery conservation.

Copyright © 2007 IEEE. All rights reserved.

O.5.6.3 Store state

This command is implemented as described in 7.1.6.3. There is only a single state, labeled state 0. There are no settable parameters for this TIM, so the store state is the same as the reset state and this operation is a NOP.

O.5.6.4 Recall state

This command is implemented as described in 7.1.6.4. There is only a single state, labeled state 0. There are no settable parameters for this TIM, so the store state is the same as the reset state and this operation is a NOP.

O.5.6.5 Read IEEE 1451.0 version

This command is implemented as described in 7.1.6.5. The command will always return a 1.

O.5.7 TIM any state commands

The following commands may be executed when the TIM is in any state. Table O.16 reproduces the relevant portions of Table 38 that gives the list of the commands in this class.

All commands in this class require a destination TransducerChannel number of zero. If destination TransducerChannel number in the octet array is not zero, the command rejected bit in the TransducerChannel Status-Condition Register (see 5.13) shall be set and the command shall be ignored.

Table O.16—TIM any state commands

Function	Command	Address		Reply expected	Required/ optional
		TIM	Global		
1	Reset	Yes	No	No	Optional

O.5.7.1 Reset

This command is implemented and causes the initialization sequence to occur.

Copyright © 2007 IEEE. All rights reserved.