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

Sponsored by the Technical Committee on Sensor Technology TC-9 of the IEEE Instrumentation and Measurement Society

Prepared by the Common Functionality and TEDS Working Group

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 set up and control of the TIM as well as reading and writing the data used by the system. Application Programming Interfaces (API) 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.

Copyright © 2007 by the Institute of Electrical and Electronics Engineers, Inc.
Three Park Avenue
New York, New York 10016-5997, USA
All rights reserved.

This document is an unapproved draft of a proposed IEEE-SA Standard – USE AT YOUR OWN RISK. As such, this document is subject to change. Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only. Prior to submitting this document to another standard development organization for standardization activities, permission must first be obtained from the Manager, Standards Licensing and Contracts, IEEE Standards Activities Department. Other entities seeking permission to reproduce portions of this document must obtain the appropriate license from the Manager, Standards Licensing and Contracts, IEEE Standards Activities Department. The IEEE is the sole entity that may authorize the use of IEEE owned trademarks, certification marks, or other designations that may indicate compliance with the materials contained herein.
Introduction

This introduction is not part of IEEE 1451.0, 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 IEEE p1451.0 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 the IEEE p1451.0. IEEE Std. 1451.2 and IEEE Std. 1451.3 will also be modified to interface with IEEE p1451.0. When these changes are made, the functions of an IEEE 1451 transducer will be as defined in IEEE p1451.0 as will the commands and TEDS. IEEE p1451.5, that may use any of several different wireless communications media, and IEEE p1451.6 are being written around the functions, commands and TEDS as described in IEEE p1451.0. 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 IEEE p1451.0. Items shown with a gray background are items that are not covered by any of the IEEE 1451 family of standards but which may be used.

<table>
<thead>
<tr>
<th>1451.1 Application</th>
<th>Simple URL Application</th>
<th>Other Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1451.0 Functions, Commands &amp; TEDS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>145L2</td>
<td>145L3</td>
<td>P145L5</td>
</tr>
<tr>
<td>145L4</td>
<td>145L4</td>
<td></td>
</tr>
<tr>
<td>Analog/Mixed Mode</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The underlying purpose of this family of standards is to allow manufacturers to build elements of a system that are interoperable. To accomplish this, the IEEE 1451 family of standards divide 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 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 IEEE p1451.0 they should be interoperable.

The IEEE p1451.0 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 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 set up and control of the transducer modules as well as reading and writing the data used by the system. Commands are also provided for reading and writing the transducer electronic data sheets (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. There are several TEDS defined in the standard. Four of these TEDS are required and the remainders 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.

The following persons were members of the committee at the time of balloting:

Officers:
Kang Lee, Chair
Victoria Sweetser, Vice Chair
Jay Nemeth-Johannes, Secretary
Lee Eccles, Editor
Jeff Burch, API Architect

Pat Blakely
Murat Bog
Charles Jones
Ed Koch
Dan Maxwell

David Perrussel
Robert Sinclair
Andrew Segal
Mallikarjun Shankar
Eugene Song

Rich Valde
Larry Whipple
Jamie Wiczer
Darold Wobschall

Other individuals who have contributed to this draft are:

Thurston Brooks
Wayne Catlin
Baozhi Chen
Stephen Chen
Ryan Coleman
Ken Cornett

Phil Ellerbrock
Peter Flitner
Fernando GenKuong
Robert Johnson
Alice Law

Myung Lee
Larry Malchodi
Loren Rittle
Jay Warrior
Stan Woods
The following persons were on the balloting committee:

(To be provided by IEEE editor at time of publication.)
## Contents

1 Introduction.................................................................................................................................................... ii  
2 1 Overview ...............................................................................................................................................12  
3 1.1 Scope .............................................................................................................................................13  
4 1.2 Purpose ..........................................................................................................................................13  
5 1.3 Conformance .................................................................................................................................13  
6 2 References .............................................................................................................................................14  
7 3 Definitions, acronyms and abbreviations ..............................................................................................15  
8 3.1 Acronyms and abbreviations .........................................................................................................15  
9 3.2 Definitions .....................................................................................................................................16  
10 4 Data types...........................................................................................................................................19  
11 4.1 Unsigned octet integer....................................................................................................................19  
12 4.2 Unsigned 16-bit integer ..................................................................................................................19  
13 4.3 Signed 32-bit integer .....................................................................................................................19  
14 4.4 Unsigned 32-bit integer ..................................................................................................................19  
15 4.5 Single precision real .....................................................................................................................20  
16 4.6 Double precision real.....................................................................................................................20  
17 4.7 String .............................................................................................................................................20  
18 4.8 Boolean..........................................................................................................................................20  
19 4.9 IEEE1451Dot0::Args::TimeRepresentation ..................................................................................21  
20 4.10 Datatypes for associated applications ..........................................................................................22  
21 4.11 Physical units...............................................................................................................................22  
22 4.12 Universal unique identification ....................................................................................................24  
23 4.13 Arbitrary octet array ....................................................................................................................25  
24 4.14 String array...................................................................................................................................26  
25 4.15 Boolean array ................................................................................................................................26
<table>
<thead>
<tr>
<th></th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.2 Command message structure</td>
<td>64</td>
</tr>
<tr>
<td>2</td>
<td>6.3 Reply messages</td>
<td>65</td>
</tr>
<tr>
<td>3</td>
<td>6.4 TIM initiated message structure</td>
<td>65</td>
</tr>
<tr>
<td>4</td>
<td>7 Commands</td>
<td>66</td>
</tr>
<tr>
<td>5</td>
<td>7.1 Standard commands</td>
<td>66</td>
</tr>
<tr>
<td>6</td>
<td>7.2 Manufacturer-defined commands</td>
<td>88</td>
</tr>
<tr>
<td>7</td>
<td>8 Transducer Electronic Data Sheet (TEDS) specification</td>
<td>88</td>
</tr>
<tr>
<td>8</td>
<td>8.1 General format for TEDS</td>
<td>88</td>
</tr>
<tr>
<td>9</td>
<td>8.2 Order of Octets in numeric fields</td>
<td>90</td>
</tr>
<tr>
<td>10</td>
<td>8.3 TEDS identification header</td>
<td>90</td>
</tr>
<tr>
<td>11</td>
<td>8.4 Meta-TEDS</td>
<td>91</td>
</tr>
<tr>
<td>12</td>
<td>8.5 TransducerChannel TEDS</td>
<td>101</td>
</tr>
<tr>
<td>13</td>
<td>8.6 Calibration TEDS</td>
<td>125</td>
</tr>
<tr>
<td>14</td>
<td>8.7 Frequency Response TEDS</td>
<td>142</td>
</tr>
<tr>
<td>15</td>
<td>8.8 Transfer Function TEDS</td>
<td>145</td>
</tr>
<tr>
<td>16</td>
<td>8.9 Text-based TEDS</td>
<td>156</td>
</tr>
<tr>
<td>17</td>
<td>8.10 End User Application Specific TEDS</td>
<td>161</td>
</tr>
<tr>
<td>18</td>
<td>8.11 User’s Transducer Name TEDS</td>
<td>161</td>
</tr>
<tr>
<td>19</td>
<td>8.12 Manufacturer-defined TEDS</td>
<td>163</td>
</tr>
<tr>
<td>20</td>
<td>8.13 PHY TEDS</td>
<td>164</td>
</tr>
<tr>
<td>21</td>
<td>9 Introduction to IEEE p1451.0 Application Program Interface (API)</td>
<td>164</td>
</tr>
<tr>
<td>22</td>
<td>9.1 API Goals</td>
<td>165</td>
</tr>
<tr>
<td>23</td>
<td>9.2 API Design Decisions:</td>
<td>165</td>
</tr>
<tr>
<td>24</td>
<td>9.3 IEEE1451Dot0</td>
<td>168</td>
</tr>
<tr>
<td>25</td>
<td>10 Transducer services API</td>
<td>180</td>
</tr>
<tr>
<td>26</td>
<td>10.1 IEEE1451Dot0::TransducerServices::TimDiscovery</td>
<td>180</td>
</tr>
<tr>
<td>27</td>
<td>10.2 IEEE1451Dot0::TransducerServices::TransducerAccess</td>
<td>182</td>
</tr>
<tr>
<td>28</td>
<td>10.3 IEEE1451Dot0::TransducerServices::TransducerManager</td>
<td>187</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>10.4</td>
<td>IEEE1451Dot0::TransducerServices::TedsManager</td>
<td></td>
</tr>
<tr>
<td>10.5</td>
<td>IEEE1451Dot0::TransducerServices::CommManager</td>
<td></td>
</tr>
<tr>
<td>10.6</td>
<td>IEEE1451Dot0::TransducerServices::AppCallback</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Module Communications Applications Program Interface (API)</td>
<td></td>
</tr>
<tr>
<td>11.1</td>
<td>IEEE1451Dot0::ModuleCommunication::Comm</td>
<td></td>
</tr>
<tr>
<td>11.2</td>
<td>IEEE1451Dot0::ModuleCommunication::P2PComm</td>
<td></td>
</tr>
<tr>
<td>11.3</td>
<td>IEEE1451Dot0::ModuleCommunication::NetComm</td>
<td></td>
</tr>
<tr>
<td>11.4</td>
<td>IEEE1451Dot0::ModuleCommunication::Registration</td>
<td></td>
</tr>
<tr>
<td>11.5</td>
<td>IEEE1451Dot0::ModuleCommunication::P2PRegistration</td>
<td></td>
</tr>
<tr>
<td>11.6</td>
<td>IEEE1451Dot0::ModuleCommunication::NetRegistration</td>
<td></td>
</tr>
<tr>
<td>11.7</td>
<td>IEEE1451Dot0::ModuleCommunication::Receive</td>
<td></td>
</tr>
<tr>
<td>11.8</td>
<td>IEEE1451Dot0::ModuleCommunication::P2PReceive</td>
<td></td>
</tr>
<tr>
<td>11.9</td>
<td>IEEE1451Dot0::ModuleCommunication::NetReceive</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>HTTP protocol</td>
<td></td>
</tr>
<tr>
<td>12.1</td>
<td>IEEE 1451.0 HTTP API</td>
<td></td>
</tr>
<tr>
<td>12.2</td>
<td>Discovery API</td>
<td></td>
</tr>
<tr>
<td>12.3</td>
<td>Transducer access API</td>
<td></td>
</tr>
<tr>
<td>12.4</td>
<td>TEDS Manager API</td>
<td></td>
</tr>
<tr>
<td>12.5</td>
<td>Transducer Manager API</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>(informative) Guidance to transducer services interface</td>
<td></td>
</tr>
<tr>
<td>B.1</td>
<td>Non-blocking Read example</td>
<td></td>
</tr>
<tr>
<td>B.2</td>
<td>Generic SendCommand( ) mechanism</td>
<td></td>
</tr>
<tr>
<td>B.3</td>
<td>IEEE p1451.0 processing details</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>(informative) Guidance to module communication interface</td>
<td></td>
</tr>
<tr>
<td>C.1</td>
<td>Module communication API discussion</td>
<td></td>
</tr>
<tr>
<td>C.2</td>
<td>Symmetric APIs</td>
<td></td>
</tr>
<tr>
<td>C.3</td>
<td>Implementation choices</td>
<td></td>
</tr>
<tr>
<td>C.4</td>
<td>Implementation examples</td>
<td></td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>C.5</td>
<td>254</td>
</tr>
<tr>
<td>2</td>
<td>C.6</td>
<td>255</td>
</tr>
<tr>
<td>3</td>
<td>C.7</td>
<td>255</td>
</tr>
<tr>
<td>4</td>
<td>C.8</td>
<td>255</td>
</tr>
<tr>
<td>5</td>
<td>C.9</td>
<td>256</td>
</tr>
<tr>
<td>6</td>
<td>C.10</td>
<td>256</td>
</tr>
<tr>
<td>7</td>
<td>C.11</td>
<td>259</td>
</tr>
<tr>
<td>8</td>
<td>Annex C</td>
<td>262</td>
</tr>
<tr>
<td>9</td>
<td>D.1</td>
<td>262</td>
</tr>
<tr>
<td>10</td>
<td>D.2</td>
<td>262</td>
</tr>
<tr>
<td>11</td>
<td>D.3</td>
<td>263</td>
</tr>
<tr>
<td>12</td>
<td>D.4</td>
<td>270</td>
</tr>
<tr>
<td>13</td>
<td>D.5</td>
<td>274</td>
</tr>
<tr>
<td>14</td>
<td>D.6</td>
<td>275</td>
</tr>
<tr>
<td>15</td>
<td>D.7</td>
<td>276</td>
</tr>
<tr>
<td>16</td>
<td>D.8</td>
<td>277</td>
</tr>
<tr>
<td>17</td>
<td>D.9</td>
<td>282</td>
</tr>
<tr>
<td>18</td>
<td>D.10</td>
<td>283</td>
</tr>
<tr>
<td>19</td>
<td>Annex E</td>
<td>285</td>
</tr>
<tr>
<td>20</td>
<td>E.1</td>
<td>285</td>
</tr>
<tr>
<td>21</td>
<td>E.2</td>
<td>285</td>
</tr>
<tr>
<td>22</td>
<td>E.3</td>
<td>285</td>
</tr>
<tr>
<td>23</td>
<td>Annex F</td>
<td>287</td>
</tr>
<tr>
<td>24</td>
<td>F.1</td>
<td>287</td>
</tr>
<tr>
<td>25</td>
<td>F.2</td>
<td>287</td>
</tr>
<tr>
<td>26</td>
<td>F.3</td>
<td>287</td>
</tr>
<tr>
<td>27</td>
<td>Annex G</td>
<td>289</td>
</tr>
<tr>
<td>28</td>
<td>G.1</td>
<td>289</td>
</tr>
</tbody>
</table>
M.1 Trigger logic augmented with an embedded time delay actuator ................................................307
M.2 Trigger logic augmented with TimeInstance sensor .................................................................308
M.3 Trigger logic augmented with time interval sensor .................................................................309
Annex N (informative) Notation Summary for IDL ....................................................................312
N.1 Key Features of IDL .................................................................................................................312
N.2 Example of IDL notation with explanation ...........................................................................312
Annex O (Informative) TEDS implementation of a simple sensor ..............................................316
O.1 Meta-TEDS .............................................................................................................................316
O.2 TransducerChannel TEDS .......................................................................................................320
O.3 Calibration TEDS ....................................................................................................................325
O.4 User’s transducer name TEDS ...............................................................................................328
O.5 Required commands ..............................................................................................................329
DRAFT 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 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.

A network capable application processor (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 IEEE p1451.0. An API is also provided between IEEE p1451.0 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 NCAPs with a single set of hardware and software without regard to distinguishing separate interfaces between IEEE p1451.0 functionality and IEEE p1451.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 API 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 used without having to add special drivers, profiles or make any other changes to the system. This is referred to as “plug and play” operation. The primary features that enable 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.

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
Clause 2: “References” lists references to other standards and documents that are useful in applying this standard.

Clause 3: “Definitions” 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: “Transducer Electronic Data Sheet (TEDS) specification” specifies the transducer electronic data sheet structure and content.

Clause 9: “Introduction to the IEEE p1451.0 Application Program Interfaces” gives the common features of the two APIs.

Clause 10: “Transducer services API” gives the API that an application would use to utilize IEEE p1451.0.

Clause 11: “Module communications API” gives the API that IEEE p1451.0 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.

1.1 Scope

This project develops a set of common functionality for the family of IEEE P1451 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 transducer electronic data sheet (TEDS) formats. It defines a set of implementation-independent application programming interfaces (API). 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 there is no common set of functions, communications protocols, and TEDS formats that 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 IEEE p1451.0 provided the following requirements are met:
  — The TIM supports the required functional specifications identified in clause 5
  — The TIM supports the message structures specified in clause 6
  — The TIM supports the required commands specified in clause 7
  — The TIM supports required TEDS that have the format and content specified in clause 8
  — The TIM supports one of the communications protocols and physical media defined by another member of the IEEE 1451 family of standards.

NOTE—There are several features that are highly desirable, and are supported by this standard, but are not practical to make into hard requirements. It is desirable that the sense element for a sensor be an integral part of the TIM but for sensing elements like structural strain gages and thermocouples this is not practical so it has not been made into a hard requirement. In addition, it is very desirable that the TEDS be located within the TIM but there are systems where the...
environment and/or the physical size make this impractical so the standard allows the TEDS to be located remote from the TIM.

An NCAP implementation shall be deemed in conformance with IEEE p1451.0 provided the following requirements are met:
— The NCAP supports the required functional specifications identified in clause 5
— The NCAP supports the message structures specified in clause 6
— The NCAP supports the required commands specified in clause 7
— The NCAP supports one of the communications protocols and physical media defined by another member of the IEEE 1451 family of standards.

1.3.1 Conformance keywords

Several keywords are used to differentiate among various levels of requirements and optionality, as follows:

1.3.1.1 Shall

The keyword “shall” indicates a mandatory requirement. Designers are required to implement all such mandatory requirements to ensure interoperability with other products that conform to IEEE p1451.0.

1.3.1.2 Shall not

The keyword “shall not” indicates a mandatory exclusion. Designers are required NOT to implement all such exclusions to ensure interoperability with other products that conform to IEEE p1451.0.

1.3.1.3 Recommended

Recommended is a keyword indicating flexibility of choice with a strong preference alternative. The word “should” has the same meaning.

1.3.1.4 Should

Should is a keyword indicating flexibility of choice with a strong preference alternative. The phrase it is recommended has the same meaning.

1.3.1.5 Should not

Should not is a keyword indicating flexibility of choice with a strong preference that a given alternative not be implemented.

1.3.1.6 May

May is a keyword that indicates flexibility of choice with no implied preference.

2 References

This standard shall be used in conjunction with the following normative references. When the following references are superseded by an approved revision, the revision shall apply.
3 Definitions, acronyms and abbreviations

This clause provides definitions that are either not found in other standards, or have been modified for use within this standard.

3.1 Acronyms and abbreviations

This subclause contains abbreviations of key terms as they are used in this standard. IEEE Std 100, The Authoritative Dictionary of IEEE Standard Terms, should be referenced for acronyms and abbreviations not defined in this clause.

1 ANSI publications are available from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.
2 Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, 6 October 2000.
3 HTTP URL syntax (RFC 2616), HyperText Transfer Protocol (W3C).
5 IEEE Std 802.3™-2002 IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) access method and physical layer specifications.
AD: anno domini
CRS: Coordinate Reference System
DTD: document type declaration, ref. Extensible Markup Language (XML)
MJD: modified Julian date
NaN: not a number
NCAP: network capable application processor
OMG: Object Management Group, a consortium committed to developing technically excellent, commercially viable and vendor independent specifications for the software industry
PHY: physical layer
SI: international system of units, reference The International System of Units (SI) [A6]
TAI: International Atomic Time
TIM: transducer interface module
TEDS: transducer electronic data sheet
USASCII: U.S. American standard code for information interchange (ANSI X3.4-1986 (Reaff 1992))
UTC: universal coordinated time
UUID: universal unique identifier
Xdcr: transducer
XML: extensible markup language

3.2 Definitions

Within this standard, the following terms and definitions apply. IEEE Std 100, The Authoritive Dictionary of IEEE Standard Terms, should be referenced for terms not defined in this clause.

3.2.1 Actuator: A transducer that accepts a data sample or samples and converts them into an action. The action may be completely contained within the TIM or may change something outside of the TIM.

3.2.2 Address: A character or group of characters that identifies a register, a particular part of storage, or some other data source or destination.

3.2.3 AddressGroup: A collection of TransducerChannels that respond to a single address.

3.2.4 Buffer: An intermediate data storage location used to compensate for the difference in rate of flow of data or time of occurrence of events when transmitting information from one device to another.

3.2.5 Calibration: The process used to determine the information that resides in the Calibration TEDS to support correction.

3.2.6 ControlGroup: Manufacturer specifications that define the inherent relationships between the TransducerChannels of a multi-channel TIM. This ControlGrouping information is not normally used by the TIM itself. This information is used to identify TransducerChannels that are used to control some characteristic of another TransducerChannel. For example, a ControlGroup could be used to identify an actuator that is used to set the threshold of an analog event sensor.

3.2.7 Correction: The evaluation of a multinomial function using information from the Calibration TEDS together with data from one or more transducers.

3.2.8 Data model: The numeric format in which the TIM shall output or accept data.

3.2.9 Data-set: The collection of samples acquired by a sensor (or applied by an actuator) in response to a trigger command.

3.2.10 Data sheet: A set of information on a device that defines the parameters of operation and conditions of usage (usually produced by the device's manufacturer).
3.2.11 Digital interface: A communications media and a protocol for transferring information by binary means only.

3.2.12 Electronic data sheet: A data sheet stored in some form of electrically readable memory (as opposed to a piece of paper).

3.2.13 Embedded transducer: A device that behaves as a transducer from the point of view of the NCAP even though nothing outside of the TIM is sensed or changed. Embedded transducers are useful for setting or reading operating parameters of other transducers.

3.2.14 Enumeration: The assignment of a numeric value to a specific meaning within the context of a specific data field. Binary numbers are usually expressed in decimal terms for human convenience. Not all possible numeric values need have a specific meaning. Values without meaning are declared to be unused or reserved for future use. Enumeration is the process of declaring the encoding of human interpretable information in a manner convenient for digital electronic machine storage and interchange. Any subclause that defines a TEDS data field to be enumerated shall contain a table that defines the meaning of the data field for each numeric value possible. The meanings encoded in each data field shall be specific and unique to that data field and only that data field. The value becomes meaningless if not associated with the data field and its defining table.

3.2.15 Event sensor: A sensor that detects a change of state in the physical world. The fact that a change of state has occurred and/or instant in time of the change of state, not the state value, is the "measurement."

3.2.16 Hot swap: The act of connecting or disconnecting a TIM from a transducer interface medium without first turning off the power that is supplied to the TIM over the medium.

3.2.17 Least significant bit (lsb): The bit in the binary notation of a number that is the coefficient of the lowest exponent possible.

3.2.18 Message: Information that is to be passed between devices as a single logical entity. A message may occupy one or more packets.

3.2.19 Meta-: A Greek prefix meaning: that which pertains to the whole or overall entity or that which is in common or shared with all member entities comprising the whole.

3.2.20 Meta-TEDS: The collection of those TEDS data fields that pertain to the whole or overall entity or those that are in common or shared with all member entities (TransducerChannels) comprising the whole product.

3.2.21 Multinomial: A linear sum of terms involving powers of more than one variable.

$$
\sum_{i_1=0}^{\infty} \sum_{i_2=0}^{\infty} \ldots \sum_{i_m=0}^{\infty} A \left( i_1, i_2, \ldots, i_m \right) X_1^{i_1} X_2^{i_2} \ldots X_m^{i_m}
$$

3.2.22 Network capable application processor (NCAP): A device between the Transducer modules and the network. The NCAP performs network communications, TIM communications, and data conversion or other processing functions.

3.2.23 Not-a-number (NaN): As defined in IEEE Std 754-1985, a bit pattern of a single or double precision real number data type that is a result of an invalid floating point operation. For single precision real numbers, the bit pattern shall have an exponent of 255 (decimal) and a non-zero mantissa. The sign is not considered in determining if a value is not a number.

3.2.24 Octet: A group of eight bits. (In the US an octet is usually referred to as a byte.)
3.2.25 **Packet**: A block of information that is to be passed by the physical layer between devices in a single transmission.

3.2.26 **Sample latched**: The term is used in a sensor to signal that the sample has been acquired. This may be when a sample and hold circuit switches to the hold mode or other similar operation. For an actuator, it signals that the sample has been moved to the output logic.

3.2.27 **Sensor**: A transducer that converts a physical, biological, or chemical parameter into an electrical signal.

3.2.28 **Setup time**: The time between the initial request for a function to be performed and when the task is actually initiated.

3.2.29 **Signal conditioning**: The transducer signal processing that involves operations such as amplification, compensation, filtering, and normalization.

3.2.30 **Smart transducer**: A transducer that provides functions beyond those necessary for generating a correct representation of a sensed or controlled quantity. This functionality typically simplifies the integration of the transducer into applications in a networked environment.

3.2.31 **Synchronization signal**: For the purposes of this standard, a signal transmitted by the NCAP to provide clock or time synchronization signals to a TIM or group of TIMs. Synchronization signals are not available with all physical layer standards. Low cost and performance TIMs may not implement a receiver for the synchronization signal.

3.2.32 **Transducer**: A device that converts energy from one domain into another. The device may either be a sensor or an actuator.

3.2.33 **Transducer interface module (TIM)**: A module that contains the TEDS, logic to implement the Transducer interface, the transducer(s) or connection to the transducer(s) and any signal conversion or signal conditioning.

3.2.34 **TransducerChannel**: A transducer and all of the signal conditioning and conversion components associated with that transducer.

3.2.35 **TransducerChannel number**: An sixteen-bit number assigned to an individual TransducerChannel within a TIM by the manufacturer.

3.2.36 **TransducerChannel proxy**: A device that is created to allow a collection of TransducerChannels to be treated as a single entity. A TransducerChannel proxy is similar to a normal TransducerChannel except that it does not require a TransducerChannel TEDS, it cannot have a Calibration TEDS, Transfer Function TEDS or Frequency Response TEDS. It may support other TEDS. A TransducerChannel proxy may respond to commands.

3.2.37 **Transducer electronic data sheet (TEDS)**: An electronic data sheet describing a TIM or a TransducerChannel. The structures of multiple TEDS are described in this standard.

3.2.38 **Transfer**: The act or process of moving information from one digital device to another.

3.2.39 **Transfer function**: The function that defines the response of a TransducerChannel in both amplitude and frequency.

3.2.40 **Trigger**: A signal or message that is used to start an action.

3.2.41 **VectorGroup**: Manufacturer specifications that define the inherent relationships between the TransducerChannels of a multi-channel TIM. This VectorGrouping information is not normally used by the...
3.2.4 Virtual TEDS: A TEDS that is stored permanently in a location other than the TIM.

4 Data types

All data types used throughout the remainder of this standard are defined in subordinate subclauses.

4.1 Unsigned octet integer

Symbol: UInt8

Size: 1 octet

IDL: typedef octet UInt8;

This data type represents positive integers from zero to 255.

4.2 Unsigned 16-bit integer

Symbol: UInt16

Size: 2 octets

IDL: typedef unsigned short UInt16;

A UInt16 may take any value between 0 to 65,535.

4.3 Signed 32-bit integer

Symbol: Int32

Size: 4 octets

IDL: typedef long Int32;

This data type is used to represent a signed integer from -2 147 483 648 to 2 147 483 647.

4.4 Unsigned 32-bit integer

Symbol: UInt32

Size: 4 octets

IDL: typedef unsigned long UInt32;

This data type is used to represent positive integers from zero to 4,294,967,295.
4.5 Single precision real

Symbol: Float32
Size: 4 octets

IDL: typedef float Float32;

A single precision real number is a thirty-two bit binary sequence that encodes real numbers as specified in IEEE Std 754-1985.

4.5.1 Floating-point not-a-number (NaN) in TEDS

According to IEEE Std 754-1985 a single precision number with an exponent of 255 and a fractional portion (mantissa) that is non-zero is NaN regardless of the sign bit. The recommended value for use in a TEDS field for NaN is 0x7FFFFFFF (hex).

4.6 Double precision real

Symbol: Float64
Size: 8 octets

IDL: typedef double Float64;

A double precision real number is a sixty-four bit binary sequence that encodes real numbers as specified in IEEE Std 754-1985.

4.7 String

Symbol: _String
Size: variable

IDL: typedef string _String; // Leading '_' to escape reserved IDL keyword

The use of text strings, as type string as a basic type that is defined in IEEE 1451.2, has been replaced with the use of OMG standards for IDL.

The Extensible Markup Language (XML) is used in 8.9 the text-based TEDS. The controlling document for XML is the W3C Recommendation Extensible Markup Language (XML) 1.0 (Second Edition). All text strings in the TEDS shall conform to this recommendation or a future update to it.

4.8 Boolean

Symbol: _Boolean
Size: 1 octet

IDL: typedef boolean _Boolean; // Leading '_' to escape reserved IDL keyword

The type Boolean is a basic type that is defined in the OMG standards for IDL.
For the purposes of this standard, a bit or octet with a non-zero value is considered True. A zero value represents the False state of the variable.

### 4.9 IEEE1451Dot0::Args::TimeRepresentation

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.

**IDL:**
```
struct TimeRepresentation {
    UInt32 secs;
    UInt32 nsecs;
};
```

**NOTE**—This format is corrected for the flaw in negative time representation discovered in IEEE Std 1588-2002.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secs</td>
<td>UInt32</td>
<td>Seconds—This is an unsigned thirty-two bit number that represent the number of seconds from the beginning of the epoch (normally 0 hours on 1 January 1970).</td>
</tr>
<tr>
<td>nsecs</td>
<td>UInt32</td>
<td>Sign, Nanoseconds—This is an unsigned thirty-two-bit integer that is comprised of two smaller fields. The most significant bit will be interpreted as the sign bit of the entire time value. The least significant 31 bits represents the number of nanoseconds to be added to the value identified by the Seconds field before the sign is applied. The value specified in the Nanoseconds field is constrained to the domain 0 to 999 999 999, inclusive.</td>
</tr>
</tbody>
</table>

This standard uses the time representation specified in IEEE Std. 1588-2002. One of the consequences of this choice is that time is measured in TAI seconds instead of UTC. TAI is the international standard for time based on the SI second as realized on the rotating geoid. TAI is implemented by a suite of atomic clocks and forms the timekeeping basis for other time scales in common use. Of these, UTC is the time scale of most engineering and commercial interest. The UTC representation is specified by ISO 8601 as YYYY-MM-DD for the date and hh:mm:ss for the time in each day.

The duration of a second in UTC time is identical to the duration of a second of TAI. UTC time differs from the TAI time by a constant offset. This offset is modified on occasion by adding or subtracting leap seconds.

Starting on 0 hours of 1 January 1972 (MJD 41,317.0), the world’s standard time systems began the implementation of leap seconds to allow only integral second correction between UTC Seconds and TAI. Time in both UTC and TAI is expressed in days, hours, minutes and seconds. Leap second corrections, that are applied to UTC but not to TAI, are made preferably following second 23:59:59 of the last day of June or December. The first such correction, a single positive leap second correction, was made following 23:59:59 on 30 June 1972, and UTC was 11 seconds behind TAI following that instant.

### 4.9.1 IEEE1451Dot0::Args::TimeDuration

This is a sub-class of time representation. It is used to specify a time interval rather than a time value. The definition of the two arguments is the same as shown in 4.9.

**IDL:**
```
struct TimeDuration {
    UInt32 secs;
    UInt32 nsecs;
};
```
4.9.2 IEEE1451Dot0::Args::TimeInstance

Time values are represented by this struct in the IEEE p1451.0 system. This struct is appropriate when trying to represent a time value and not time duration. A time value that occurs before the epoch is specified by a negative “nsec” field.

IDL:
```cpp
struct TimeInstance {
  UInt32 secs;
  UInt32 nsecs;
};
```

The TimeInstance is based on an epoch that is defined as starting at midnight Jan 1, 1970. The definition of the two arguments is the same as shown in 4.9.

4.10 Datatypes for associated applications

The datatypes in this clause are not used in IEEE p1451.0 but are provided to allow the applications using this standard to pass data in these formats.

4.10.1 Eight-bit signed integer

Symbol: Int8

Size: 1 octet

IDL: typedef char Int8;

This data type represents integers from -128 to 127.

4.10.2 Sixteen-bit signed integer

Symbol: Int16

Size: 2 octets

IDL: typedef short Int16;

This data type represents integers from -32768 to 32767.

4.11 Physical units

Symbol: UNITS

Size: 11 octets

IDL: struct Units {
  UInt8 interpretation;
  UInt8 radians;
  UInt8 steradians;
  UInt8 meters;
  UInt8 kilograms;
  UInt8 seconds;
  UInt8 amperes;
  UInt8 kelvins;
  UInt8 moles;
  UInt8 candelas
  UInt8 Units Extension TEDS Access Code
};
Physical units are a binary sequence of ten octets that encode physical units as described in Table 2 and Table 3. Each of the fields shall be interpreted as an unsigned integer. A unit shall be represented as a product of the SI base units, plus radians and steradians, each raised to a rational power. This structure encodes only the exponents; the product is implicit. For examples of Physical units see Annex J. For further information, see Hamilton [A1].

Table 2—Physical units data type structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Data Type</th>
<th>Number of octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Physical units interpretation -- see Table 3</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>((2 \times \text{&lt;exponent of radians&gt;}) + 128)</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>((2 \times \text{&lt;exponent of steradians&gt;}) + 128)</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>((2 \times \text{&lt;exponent of meters&gt;}) + 128)</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>((2 \times \text{&lt;exponent of kilograms&gt;}) + 128)</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>((2 \times \text{&lt;exponent of seconds&gt;}) + 128)</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>((2 \times \text{&lt;exponent of amperes&gt;}) + 128)</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>((2 \times \text{&lt;exponent of kelvins&gt;}) + 128)</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>((2 \times \text{&lt;exponent of moles&gt;}) + 128)</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>((2 \times \text{&lt;exponent of candelas&gt;}) + 128)</td>
<td>UInt8</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3—Physical units interpretation

<table>
<thead>
<tr>
<th>Value</th>
<th>Manifest constant</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PUI_SI_UNITS</td>
<td>Unit is described by the product of SI base units, plus radians and steradians, raised to the powers recorded in fields 2 through 10. Units for some quantities, such as the number of people through a turnstile, cannot be represented using these units. Enumeration zero, with fields 2-10 set to 128, is the appropriate enumeration for these cases when a quantity is being defined.</td>
</tr>
<tr>
<td>1</td>
<td>PUI_RATIO_SI_UNITS</td>
<td>Unit is (\frac{U}{U}), where (U) is described by the product of SI base units, plus radians and steradians, raised to the powers recorded in fields 2 through 10.</td>
</tr>
<tr>
<td>2</td>
<td>PUI_LOG10_SI_UNITS</td>
<td>Unit is (\log_{10}(U)), where (U) is described by the product of SI base units, plus radians and steradians, raised to the powers recorded in fields 2 through 10.</td>
</tr>
<tr>
<td>3</td>
<td>PUI_LOG10_RATIO_SI_UNITS</td>
<td>Unit is (\log_{10}(\frac{U}{U})), where (U) is described by the product of SI base units, plus radians and steradians, raised to the powers recorded in fields 2 through 10.</td>
</tr>
<tr>
<td>4</td>
<td>PUI_DIGITAL_DATA</td>
<td>The associated quantity is digital data (for example, a bit vector) and has no unit. Fields 2-10 shall be set to 128. The &quot;digital data&quot; type applies to data that does not represent a quantity, such as the current positions of a bank of switches.</td>
</tr>
<tr>
<td>5</td>
<td>PUI_ARBITRARY</td>
<td>The associated physical quantity is represented by values on an arbitrary scale (for example, hardness). Fields 2-10 are reserved, and shall be set to 128.</td>
</tr>
<tr>
<td>6-255</td>
<td>—</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

1 The numbers in brackets correspond to those in the bibliography in Annex A.

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
The U/U forms (enumerations one and three) are for expressing “dimensionless” units such as strain (meters per meter) and concentration (moles per mole). The numerator and denominator units are identical, each being specified by subfields two through ten.

Boolean data (values in \{0, 1\} or \{False, True\}) shall be represented as digital data (enumeration 4).

4.12 Universal unique identification

Symbol: UUID

Size: 10 octets

IDL: typedef UUID Short [5];

The UUID is an identification field associated with the TIM whose value is unique on the planet. The UUID field shall be 10 octets long and consists of four sub-fields (from most to least significant order: location, manufacturer’s, year, and time) defined in Table 4.

There shall be no requirement that the interpretation of the UUID reflect the actual place or time of manufacture of the TIM. The use of time and location in the algorithm shall be used only to ensure uniqueness. Thus a manufacturer shall be allowed to use the fields defined by the algorithm as desired provided that when interpreted according to Table 4, there shall be no other possible manufacturer that could claim use of the same UUID for a TIM.
Table 4—Universal unique identification data type structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Number of bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>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 TIM manufacture. 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 meters. Thus the range represented by each 20-bit field is 0 to 1 048 575 arc seconds or 0 to 291 degrees which is sufficient to represent latitude and longitude on the earth’s surface</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>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 AD. The beginning of the year shall be deemed to start on January 1, 00:00:00, TAI.</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>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 ten-second 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. 1There are approximately 31 536 000 seconds per year.</td>
<td>22</td>
</tr>
</tbody>
</table>

4.13 Arbitrary octet array

Symbol: OctetArray

Size: varies

This data type is comprised of an arbitrary number of octets, treated as an aggregate entity that may or may not be interpreted as a number. An OctetArray can be a structure comprised of one or more primitive data types, arrays of primitive data types, or smaller OctetArrays.
4.14 String array
Symbol: StringArray
Size: varies
This data type is comprised of an arbitrary number of string data types (see 4.7), treated as an aggregate entity.

4.15 Boolean array
Symbol: BooleanArray
Size: varies
This data type is comprised of an arbitrary number of boolean data types (see 4.7), treated as an aggregate entity.

4.16 Array of eight bit signed integers
Symbol: Int8Array;
Size: varies
This data type is comprised of an arbitrary number of octets, treated as an aggregate entity made up of eight bit signed integers (Int8).

4.17 Array of sixteen bit signed integers
Symbol: Int16Array;
Size: varies
This data type is comprised of an arbitrary number of sixteen bit signed integers (Int16), treated as an aggregate entity.

4.18 Array of thirty-two bit signed integers
Symbol: Int32Array;
Size: varies
This data type is comprised of an arbitrary number of thirty-two bit signed integers (Int16), treated as an aggregate entity.

4.19 Array of eight bit unsigned integers
Symbol: UInt8Array;
Size: varies
This data type is comprised of an arbitrary number of octets, treated as an aggregate entity made up of eight bit unsigned integers (Int8).

4.20 Array of sixteen bit unsigned integers

Symbol: UInt16Array;
Size: varies
This data type is comprised of an arbitrary number of sixteen bit unsigned integers (Int16), treated as an aggregate entity.

4.21 Array of thirty-two bit unsigned integers

Symbol: UInt32Array;
Size: varies
This data type is comprised of an arbitrary number of thirty-two bit unsigned integers (Int16), treated as an aggregate entity.

4.22 Array of single precision real numbers

Symbol: Float32Array;
Size: varies
This data type is comprised of an arbitrary number of single precision real numbers as specified in IEEE Std 754-1985, treated as an aggregate entity.

4.23 Array of double precision real numbers

Symbol: Float64Array;
Size: varies
This data type is comprised of an arbitrary number of double precision real numbers as specified in IEEE Std 754-1985, treated as an aggregate entity.

4.24 Array of TimeDuration data types

Symbol: TimeDurationArray;
Size: varies
This data type is comprised of an arbitrary number of TimeDuration data types as specified in 4.9.1, treated as an aggregate entity.

4.25 Array of TimeInstance data types

Symbol: TimeInstanceArray;
5 Smart transducer functional specification

This clause of the standard provides a description of the functions expected to be in an IEEE 1451 smart
transducer. The details of these functions are provided in clause 6 through clause 11.

5.1 IEEE 1451 family reference model

The reference model shown in Figure 1 and Figure 2 shows the relationships between the various IEEE
1451 standards family members. They are to be used to help define what is in each of the standards in the
family. It is not intended to imply a required implementation of devices that comply with the standard.
Sub-clauses 5.1.1 through 5.1.15 give a further description of the elements shown in Figure 1 and Figure 2.

5.1.1 User’s Network

The user’s network is outside the scope of the IEEE 1451 family of standards. It may be anything that the
user desires. The only requirement that this places on the NCAP is that the NCAP have the appropriate
network interface hardware and software. Different NCAPs are required for networks using different
physical communications media or protocols.

5.1.2 Network access module

The network access module is the part of the NCAP that provides the interface to the user’s
communications network. This communications network is not defined as part of any of the IEEE 1451
family of standards. Since the communications network is not defined, the functions of this module are not
fully defined by any IEEE 1451 standard. IEEE Std. 1451.1-1999 provides a logical model for this module
but the use of IEEE 1451.1 is not required for an NCAP to be in compliance with IEEE p1451.0 or other
members of the IEEE 1451 family based on IEEE p1451.0.

5.1.3 Transducer services interface

This software interface is defined in IEEE p1451.0 in terms of an application program interface or API.
See clause 10 for the details of this interface.

5.1.4 NCAP IEEE p1451.0 services

The block defines the functions and services provided to the NCAP communications module and to the
NCAP Applications. The functions provided by this module are defined in IEEE p1451.0. These functions
include the command set and the transducer electronic data sheets (TEDS). Features are included in the
standard to support capabilities such as triggering and synchronous sampling of sensors but the detailed
implementation is left up to the other standards in the family.
5.1.5 Module Communications Interface

The Module Communications Interface is defined in IEEE p1451.0 in terms of an API that is used to pass information across the interface between the NCAP IEEE p1451.0 services and the NCAP IEEE 1451.X Communication Module. The physical aspects of this interface are not described in the standard. That is left up to the manufacturers. See clause 11 for details of this interface.

5.1.6 NCAP IEEE P1451.0/X Communication Module

Other members of the IEEE 1451 family of standards cover the NCAP communications module. IEEE Std 1451.2-1997, IEEE Std 1451.3-2003, and IEEE p1451.5 should all be examples of these standards. This module provides the low levels of the communications protocol stack and any other services that the NCAP requires to function in the system.

5.1.7 Physical Interface (PHY)

The physical interface between the NCAP and the TIM is left up to the other members of the standards family to define. It is not described in IEEE p1451.0. Functions needed across this interface such as synchronization, if applicable, are described but not their implementation.

5.1.8 TIM IEEE 1451.X Communication Module

The TIM IEEE P1451.0/X Communication Module is logically the complement of the NCAP IEEE P1451.0/X Communication Module but not identical since there are services required in the TIM that are different than the services required in the NCAP. As with the NCAP IEEE P1451.0/X Communication Module this module is not defined in IEEE p1451.0 but is defined by other members of the family of standards.

5.1.9 TIM IEEE p1451.0 Services

IEEE p1451.0 describes the services that are performed in this function. Most of the services performed by the NCAP services have counterparts within this block.

5.1.10 Transducer Measurement API

This interface is left up to the manufacturer and is out of scope for IEEE p1451.0.

5.1.11 Signal conditioner and data conversion functions

The details of the signal conditioning and data conversion functions are outside of the scope of IEEE p1451.0.

5.1.12 Transducer analog interface

This interface is the physical interface between the signal conditioning and data conversion functions and is out of scope for all members of the IEEE 1451 family of standards except IEEE Std. 1451.4.

5.1.13 Transducer

The transducer itself is considered out of scope by all members of the IEEE 1451 family of standards.

5.1.14 Transducer electronic data sheets (TEDS)

The basic transducer electronic data sheets (TEDS) are defined in IEEE p1451.0 along with the access methods for them. However, some information that relates to the physical interface is defined in other
standards in the IEEE 1451 family. These TEDS are defined in the other standards. See clause 8 for
detailed descriptions of the TEDS.

5.1.15 The use of IEEE Std. 1451.4-2004 with this reference model

IEEE p1451.0 is defined for the transfer of digital information between modules in a system. The IEEE
p1451.0 TEDS are used to describe the entire TIM including the transducer, signal conditioner and data
converters. The analog information that is being described as transported using these interfaces is not
covered by any of this family of standards. However, IEEE Std. 1451.4-2004 adds a TEDS in parallel with
the analog output from a transducer. This TEDS give the characteristics of the analog information that may
be obtained using an IEEE Std. 1451.4 sensor. The use of IEEE Std. 1451.4 Transducers is not required by
any other member of the IEEE 1451 family of standards. However, it is allowed as shown in this reference
model. To use an IEEE 1451.4 device with an IEEE p1451.0 compatible device requires the appropriate
signal conditioner for the transducer and an IEEE Std. 1451.4 TEDS Translator. The TEDS Translator is
required to combine the contents of the IEEE Std. 1451.4 TEDS with the characteristics of the signal
conditioner to allow the IEEE p1451.0 TEDS to describe the overall TIM. Both sets of TEDS are described
in the standards for the family members but the details of how to combine the TEDS is not covered in any
of the standards.

IEEE Std. 1451.4 transducers are commonly used with interfaces that do not comply with the other
members of the IEEE 1451 family of standards. In these cases this reference model does not apply.

5.2 Plug and play capability

A TIM(s) and NCAP that are built to this standard shall be able to be connected using a compatible
physical communications media and be able to operate without any changes to the system software. There
shall be no need for different drivers, profiles or other system software changes to provide basic operation
of the transducer. Applications that use, produce or display the data associated with a transducer are not
included in this statement. Features that go beyond the requirements of this standard are allowed but they
impact the plug and play capability of the TIM or NCAP.

5.3 Addresses

There are two levels of addressing used in this standard. One level of addressing is associated with the
physical layer implementation. The details of this address are covered in the members of the IEEE 1451
family that define the communications media. The Module Communications API links this address to the
“destId” with the registerDestination call described in 11.7.2. This level of addressing allows messages, see
clause 6, to be sent between a TIM and an NCAP or between TIMs. For more details about this address see
the standard for the physical layer communications.

The second level of addressing is the TransducerChannel number for a given TransducerChannel within
the TIM. These sixteen bit numbers are used in command messages as described in 6.1.2 as the DestChannel
TransducerChannel number or in reply messages, see 6.1.4, as source TransducerChannel number. This
level is used to specify to the TIM how the message should be directed or to tell the NCAP where it came
from within the TIM. There are rules associated with the interpretation of a TransducerChannel number
that are given in Table 5. Each row in the table defines a name for a class of addresses and that name is
used throughout the standard.

Not all commands may be honored by all address classes. The address class that honors a particular
command is defined in clause 7 along with the definition of the commands.
Table 5—Rules for TransducerChannel number usage

<table>
<thead>
<tr>
<th>Address class</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>0xFFFF</td>
<td>Global addressing is a special GroupAddress that pertains to all TransducerChannels on the TIM.</td>
</tr>
<tr>
<td>AddressGroup</td>
<td>0x8000≤A≤0xFFFE</td>
<td>There are two types of AddressGroups as defined in Table 6. The Bit Mapped AddressGroups are used when a few AddressGroups are needed and it is desirable to send a command to multiple AddressGroups at the same time. Binary AddressGroup are used when a large number of different AddressGroups are required. Address 0x8000 is a nonfunctional address.</td>
</tr>
<tr>
<td>TransducerChannel</td>
<td>1≤A≤0x7FFF</td>
<td>An address with the most significant bit set to zero. The remaining 15 bits identify the TransducerChannel for which the message is intended.</td>
</tr>
<tr>
<td>TIM</td>
<td>Zero</td>
<td>An address within the message of zero indicates that the message is intended for the TIM and not an individual TransducerChannel.</td>
</tr>
</tbody>
</table>

Table 6—AddressGroups

<table>
<thead>
<tr>
<th>Address class</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Mapped</td>
<td>0x8000≤A≤0xBFFF</td>
<td>The most significant bit is set to one, the next most significant bit is zero and at least one other bit is set to a one. The pattern in the two most significant bit is set to indicate that this is a group address. The other bit or bits identify the group(s).</td>
</tr>
<tr>
<td>Binary</td>
<td>0x000≤A≤0x7FFF</td>
<td>The two most significant bits are set to one and the remaining fourteen bits form a binary pattern identifying a single address group.</td>
</tr>
</tbody>
</table>

5.3.1 Global addresses

Commands addressed to the Global address shall be received and honored by all TransducerChannels on the addressed TIM(s). If the received command is not implemented by a TIM or TransducerChannel, the command shall be ignored and no error generated.

5.3.2 AddressGroup addresses

Commands issued to an AddressGroup shall be honored by all TransducerChannels that have been initialized as members of that AddressGroup. See the AddressGroup definition command in 7.1.2.3 for how to assign or clear an AddressGroup assignment. If the received command is not implemented by a TIM or TransducerChannel, the command shall be ignored and the command rejected bit shall be set in the TIM or TransducerChannel.

The use of AddressGroups across multiple TIMs requires that a method of addressing multiple TIMs be established before TransducerChannels can be assigned to AddressGroups. See 10.2.3 and 10.2.4 for the details of the method required to make these assignments.

5.3.2.1 Bit mapped AddressGroup addresses

Each AddressGroup is represented by having the most significant bit in the AddressGroup assignment set to one, the next most significant bit set to zero and one other bit set in the TransducerChannel number field of the address class. The one, zero pattern in the two most significant bits of the TransducerChannel
number field indicates that the address is a bit mapped group address. Using bitwise OR logic, one
command may be issued to multiple AddressGroups simultaneously. This allows for 14 different
AddressGroups.

5.3.2.2 Binary AddressGroup addresses

Each AddressGroup is represented by having the two most significant bits in the AddressGroup assignment
set to one. The remaining fourteen bits in the TransducerChannel number field of the address class identify
the AddressGroup. This allows for 16,383 different AddressGroups.

5.3.3 TransducerChannel number

Command and reply messages that have a number between one and 0x7FFF, inclusive, in the
TransducerChannel number field are addressed to a TransducerChannel. They shall be received and
honored by the TransducerChannel to which they are addressed. If the received command is not
implemented by the TransducerChannel the command shall be ignored and an error generated as specified
for the individual command.

5.3.4 TIM addresses

Command and reply messages that have zero in the TransducerChannel number field are addressed to a
TIM. They shall be received and honored by the TIM to which it is addressed by the “destId.” They are
intended for the interface module and not for any particular TransducerChannel within the TIM. If the
received command is not implemented by a TIM the command shall be ignored and an error generated as
specified for the individual command.

5.4 Common characteristics

The following subclauses define the operating characteristics that are common to all TransducerChannel
types.

5.4.1 Operating states

A high-level state transition diagram for a TransducerChannel is provided as Figure 3. There are two basic
operating states for a TransducerChannel after it comes out of initialization. The TransducerChannel enters
the “Transducer Idle” state when it comes out of initialization. A TransducerChannel accepts most
commands when it is in the Transducer Idle state. A TransducerChannel enters the Transducer Operating
state upon receipt of a TransducerChannel Operate (see 7.1.4.2) command and remains in this state until
the TransducerChannel Idle (see 7.1.7.1) command is received.

As shown in Figure 4, a TIM has three states. A TIM is placed in the TIM Initialization state by a reset (see
7.1.7.1) command or by a power up event. Once it completes the initialization process it transitions to the
TIM Active state. A TIM may be placed in the TIM Sleep state by the TIM Sleep (see clause 7.1.6.2)
command. The only command that a TIM shall accept when in the TIM Sleep state is the Wake-up (see
7.1.5.1) command that returns it to the TIM Active state.

For more details about what commands may be accepted by a TransducerChannel in each state, see the
detailed description of each command in clause 7.
5.4.2 Enabling and disabling triggers

A TransducerChannel may have its ability to be triggered enabled or disabled by means of commands.

Unless otherwise noted, all operations described in this document are for TransducerChannels with triggers enabled.

5.4.3 Diagnostics

One prime reason for using Smart Transducers is to be able to have devices that can perform self-check or diagnostics. The standard provides a mechanism to engage diagnostics, but there are no requirements imposed on the scope of the diagnostic logic.
5.4.4 Structures used to store and transmit data

There are three structures that are used to store and transmit data in this standard. They are the data-set, the message and the packet. The following sub-clauses describe each of these structures. The applications above the protocol stack deal with data-sets. The upper layers of the protocol stack deal with messages. If a data-set contains more octets than can be sent with a single message it shall be broken up by the application into multiple messages.

5.4.4.1 Data-sets

All TransducerChannels operate with "data-sets." Three fields within the TransducerChannel TEDS define a data-set. The Maximum data repetitions field (see 8.5.2.28) defines the maximum number of individual data samples in a data-set. The actual number of samples may be made lower than the number in the Maximum data repetitions field by the optional Set TransducerChannel data repetition count command. When reset or powered up, the unit uses the default value. If the data repetition count is programmable, the default value shall be zero. If the data repetition count is not programmable, the default value shall be the same as the number in the Maximum data repetitions field of the TEDS. The second field is the Series increment field (see 8.5.2.30). This field is used to determine the interval between samples and it may be overridden by a manufacturer-defined command or an embedded actuator. The third field is the Series units field (see 8.5.2.31). This field defines the units for the Series increment field. The implication of the Series units field is that the units of the Series increment field does not need to be time and that if this is the case the time interval between samples may not be uniform.

Example: The Series units are Kelvins and the Series increment is 0.5. This combination would cause a sensor to acquire data every 0.5 degrees. The samples would be at uniform temperature intervals instead of uniform time intervals.

5.4.4.2 Messages and packets

Messages may contain up to 65,535 octets plus the octets in headers. However, the data link and physical layers of the protocol stack transmit packets. The maximum packet length is defined in the standard defining the physical and data link layers. If a message is too long to fit within a single packet it is the responsibility of the data link layer in the protocol stack to break messages down into multiple packets for transmission.

5.5 Transducer Electronic Data Sheets

TEDS are blocks of information in one of the formats defined in one of the subclauses in clause 8. They are intended to be stored in non-volatile memory within a TIM. However, there are applications where this is not practical, so the standard allows them to be stored in other places in the users system. When stored in some location other than the TIM, they are referred to as "virtual" TEDS. The manufacturer of the TransducerChannel is expected to provide the virtual TEDS but not to store them in the TIM. It is the responsibility of the users system to provide a link between the only information that is guaranteed to be available from the TIM, that is the UUID, and the file that contains the virtual TEDS in the system. The NCAP or host processor provides this service if it is used. A TIM may provide a mix of TEDS stored in the TIM and virtual TEDS. The response to the Query TEDS command (see 7.1.1.1) contains flags that indicate whether or not the TEDS is supported and whether or not it is virtual. A virtual TEDS is considered "supported" if the manufacturer provided a file on some suitable media containing the information, regardless of whether or not the user loaded it into the system.

As a general rule a TEDS is not changed once the manufacturer or the user establish the contents of a TEDS. However, it is possible to design TransducerChannels that can change the contents of a TEDS during operation. This change can be caused by the use of an embedded actuator to change what would normally be a constant in the TEDS or by logic within the TIM or TransducerChannel that makes a change within the device requiring a change to the contents of a TEDS. To be able to handle this case the TEDS attributes contain a bit, the "Adaptive" bit, that identifies whether or not the TEDS can be changed by logic.
internal to the TIM or TransducerChannel. A status bit is set if the contents of a TEDS is changed without
writing the TEDS.

There are four TEDS that are required for all TIMs and others are optional. The required TEDS are as
follows:

- Meta-TEDS
- TransducerChannel TEDS
- User’s Transducer Name TEDS
- PHY TEDS

See clause 8 for the detailed specifications for all of the TEDS defined by this standard.

### 5.5.1 Required TEDS

A high level description of the required TEDS along with the purpose of these TEDS is given in the sub-
clauses 5.5.1.1 through 5.5.1.4.

#### 5.5.1.1 Meta-TEDS

The meta-TEDS give some worst-case timing parameters that are available to be used by the NCAP to set
time-out values in the communications software to determine when the TIM is not responding. The
remainder of this TEDS describes the relationships between the TransducerChannels that exist within the
TIM. See 8.4 for the specifications for this TEDS.

#### 5.5.1.2 TransducerChannel TEDS

The TransducerChannel TEDS gives detailed information about a specific transducer. It gives what
physical parameter is being measured or controlled, the range over which the TransducerChannel operates,
the characteristics of the digital I/O, the operational mode(s) of the unit and the timing information. See 8.5
for the specifications for this TEDS.

#### 5.5.1.3 User’s Transducer Name TEDS

The User’s Transducer Name TEDS is intended to be used to provide a place for the user of the transducer
to store the name by which the system will know the transducer. The structure of this TEDS is
recommended in the standard but since the contents are user defined it cannot be required. All the
manufacturer of the TIM needs to do is to provide the non-volatile memory that the user may write using
the standard TEDS access methods. See Error! Reference source not found. for further details of this
TEDS.

NOTE—The User’s Transducer Name TEDS is intended to support “Object Tags,” as defined in IEEE 1451.1 or other
similar uses.

#### 5.5.1.4 PHY TEDS

The PHY TEDS is dependent upon the physical communications media used to connect the TIM to the
NCAP and is not defined in the standard although the method of accessing it is defined

### 5.5.2 Optional TEDS

A high level description of the optional TEDS along with the purpose of these TEDS is given in the sub-
clauses 5.5.2.1 through 5.5.2.10.
5.5.2.1 Calibration TEDS

The Calibration TEDS provides the calibration constants necessary to convert the output of a sensor into engineering units or to convert an engineering units value into the form required by an actuator. Two methods are supported: one is the common linear method that uses the general form of $y=mx+b$. The other method is a completely general form that can do almost anything. See 8.5.2.55 for the specifications for this TEDS.

5.5.2.2 Frequency Response TEDS

The Frequency Response TEDS uses a table to provide the frequency response of the TransducerChannel. See 8.7 for the specifications for this TEDS.

5.5.2.3 Transfer Function TEDS

The Transfer Function TEDS describes a way to link a series of individual transfer functions together to describe the frequency response of a TransducerChannel in an algorithmic form. See 8.8 for the specifications for this TEDS.

5.5.2.4 Text-based TEDS

Text-based TEDS are a family of TEDS that provide text-based information about a TIM or TransducerChannel. These TEDS can be in one or more languages. They consist of a directory that facilitates access to a particular language sub-block within the TEDS followed by the blocks of TEDS that are defined using XML. See 8.9 for the specifications for this TEDS.

5.5.2.5 Commands TEDS

The commands TEDS is a Text-based TEDS that provides a way for the manufacturer to specify additional commands beyond those included in the standard. These commands are intended primarily to allow the commands necessary to set up a specific transducer and signal conditioner. Annex D contains an example schema for this TEDS and Annex H gives an example of this TEDS.

5.5.2.6 Identification TEDS

IEEE Std. 1451.2-1997 identified a Meta-Identification TEDS, a Channel Identification TEDS and a Calibration Identification TEDS. These were intended to provide textual information about the STIM, the TransducerChannel and the calibration. In this standard these TEDS have all been lumped into the category of Text-based TEDS and are not individually defined. The Schema in Annex D and examples in Annex E, Annex F and Annex G may be used to duplicate the intent of these TEDS as defined in IEEE Std. 1451.2. Provisions are made in this standard to access TEDS with these names.

5.5.2.7 Geographic location TEDS

The Geographic location TEDS is a Text-based TEDS that contain static geographic location information. It is expected to be written by the user to indicate the location at which the TIM is installed. The contents of this TEDS shall conform to the definition of a Text-based TEDS as specified in 8.9. The data block shall be in the Geography Markup Language (GML) developed by the Open Geospatial Consortium (OGC) and specified in ISO/DIS 19136.

The Geographic Location TEDS specifies the location of the sensor. The sensor location shall be defined in accordance with the Geography Markup Language (GML) schema for a point: “gml:point”.

Gml:point is an instance of gml:PointType that is included in the GML Schema: geometryBasic0d1d.xsd

The GML schema geometryBasic0d1d.xsd is identified by this URI:
Use of gml:point in this specification shall be restricted in following ways:

- gml:point shall use the DirectPositionType, i.e., gml:pos
- gml:point shall not use gml:coordinates or gml:coord
- gml:pos shall include the attribute gml:SRSReferenceGroup
- Note that values of gml:pos are constrained by the Coordinate Reference System (CRS) identified in SRSReferenceGroup
- gml:SRSReferenceGroup shall include srsName
- gml:SRSReferenceGroup may include srsDimension
- gml:SRSReferenceGroup may include gml:SRSInformationGroup

In this specification srsName shall be given a namespace Uniform Resource Name (URN). An example for the World Geodetic System 1984 (WGS 84) CRS:

```gml
urn:ogc:def:crs:OGC:1.3:CRS84
```

The WGS 84 CRS contains geographic latitude, then longitude, that is, the X axis corresponds to latitude, and the Y axis to longitude.

It is expected that the manufacturer will provide space for this TEDS and the user will provide the actual information that goes into it. The amount of memory required is a function of the actual GML location type. This standard does not specify the amount of memory to allot to this TEDS but a typical single language block for this TEDS is expected to occupy 60 to 80 octets of memory if using a single octet for each character.

For the GeoLocTEDS, a simple approach is to specify use of “GML Point.” This point specifies the location of the sensor once it has been installed. In the GML Point a Coordinate Reference System (CRS) needs to be identified. Without a CRS the location of the point may be inaccurate with the possibility of a hundred meter difference between two geographic CRS origins. It is recommended that the writer of this TEDS restrict the specification to be an identifier of the CRS as specified on GML Point.

### 5.5.2.8 Units extension TEDS

There are cases, especially in chemical sensors, where it is not possible to completely express the physical units in SI units. For example a sensor sensing the percentage of a certain chemical in a sample would use a ratio of units, enumeration 1 in the units type field, and the units would Moles. However this does not identify Moles of what chemical. To solve this issue this [Text-based TEDS](#) is provided to give a place to include text that would extend the SI units.

### 5.5.2.9 End User Application Specific TEDS

The [End User Application Specific TEDS](#) is much like the User’s Transducer Name TEDS in that it is a block of memory that the users may use to store anything that they desire in any format that they want to use. See Error! Reference source not found. for the specifications for this TEDS.

### 5.5.2.10 Manufacturer defined TEDS

Provisions are included in the standard to allow a manufacturer to define TEDS that are not included in the standard. The use and structure of [manufacturer defined TEDS](#) are left entirely up to the manufacturer. The manufacturer is not required to make these TEDS accessible by the user. See 8.11.2.3 for the specifications for this TEDS.
5.6 TransducerChannel type descriptions

This subclause specifies the general behavior of three TransducerChannel types. Attributes in the TransducerChannel TEDS describe the capabilities of the various TransducerChannels to the system. The detailed timing and control of these TransducerChannel types are specified in later subclauses of this standard. The TransducerChannel types are:

- Sensor
- Event sensor
- Actuator

The term TransducerChannel as used in this subclause refers to the physical transducer and all electronics in the path that connects the transducer to the communications functions.

5.6.1 Sensor

A sensor shall measure some physical parameter and return digital data representing that parameter. On the receipt of a trigger, if triggering is enabled, the sensor shall start the collection and storing of a data-set within the TIM. The timing of the individual samples in the data-set shall be controlled by the TIM and is a function of the operating mode of the sensor. A sensor, in the transducer operating state, shall respond to a Read TransducerChannel data-set segment command (see 7.1.3.1) by returning the appropriate data-set. If a new data-set is not available, the TransducerChannel shall respond with the same data that was returned on the previous Read TransducerChannel data-set segment command.

5.6.2 Event sensor

An event sensor differs from a sensor in that it does not determine the magnitude of some physical phenomena but determines when a change of state has occurred. This change of state may be an analog signal crossing some threshold or it a set of discrete bits that match or fail to match a given bit pattern. Thus the output of an event sensor indicates the state of its input. The two allowable states are one and zero. There are two pieces of information that may be determined from the output of an event sensor. One is the current state of the input and the other is the time that a change in state occurred.

The TEDS definition for an event sensor is the same as for any other transducer.

5.6.2.1 Event sensor output

The data model of an event sensor output is defined in the TransducerChannel TEDS in the same way that the output of any other sensor is defined. However the magnitude of the output shall be either zero or one.

5.6.2.2 Event time

The event sensor itself only reports the fact that an event has occurred. Other elements in the system are needed to determine when the event occurred. There are multiple ways that this may be accomplished and just which method is chosen depends upon the requirements of the application. If polling is used to determine when the event occurred the time of the event can only be known to within the polling interval. In the streaming data transmission mode the NCAP can determine when the event occurred by the time of arrival of the message. The physical layer is a major factor in determining how much lag there is between the event and the NCAP recognition of the event. On the other end of the spectrum the time of occurrence of the event may be determined to within a microsecond or less by the use of an embedded time interval sensor or a TimeInstance sensor in the TIM with the event sensor. See Annex M.2 and Annex M.3 for a discussion of time interval and TimeInstance sensors.

5.6.2.3 Analog event sensors

Analog event sensors have an input signal that is in an analog form. An event is determined when the input crosses a threshold. For an analog event sensor, upper and lower thresholds are defined. The lower
threshold differs from the upper by a quantity called hysteresis. The hysteresis value shall be zero or
greater. During steady state operation a rising transition (i.e. the condition of the input that causes the
output state to transition from zero to one) shall be when output state is zero and the analog input value
passes through the upper threshold. A falling transition (i.e. the condition of the input that causes the
output state to transition from one to zero) shall be when the output state is one and the analog input value
passes through the lower threshold. Figure 5 shows this behavior graphically. Changes of the output state
shall only be reported after the device has been armed by receiving a trigger. The upper threshold and the
hysteresis may be fixed at the time of manufacturing the event sensor or they may be made programmable.
The recommended method of making them programmable is to use embedded actuators to set the upper
threshold and hysteresis.

Initialization of an event sensor will depend upon the design. However, the following process is
recommended. For Event Sensors that report only falling transitions the output should be initialized to the
one state if the analog input value is greater than or equal to the upper threshold. If the input is less than the
upper threshold the output should be initialized to the zero state. For Event Sensors that report only rising
transitions, the output state should be initialized to the zero state if the analog input is less than or equal to
the lower threshold. If the input is greater than the lower threshold it should be initialized to the one state.
For Event Sensors that can report both transitions pick one of these methods and recognize that if the input
is within the hysteresis band the first transition may be missed. More elaborate schemes such as controlling
the analog input value during initialization are required to avoid this problem.

Digital event sensors have one or more discrete input signals. An event is determined when the input
matches a defined pattern for a manufacturer-defined fixed period of time. A rising transition shall be when
the inputs match a specific digital pattern without change for a fixed period of time. The period of time
should be determined by the manufacturer to match the characteristics of the event but may be controlled

Figure 5—Analog event sensor behavior
by manufacturer unique commands or an embedded actuator. A falling transition shall be when the inputs
to cease to match the pattern for some fixed time.

A digital event sensor may have several associated embedded TransducerChannels. This may include a
sensor(s) to return the digital input on command and actuators to setup a complex event sensor. Embedded
actuators may be used to define patterns, masks, time delays and combinations of several discrete inputs
that are combined into one event. The TransducerChannel TEDS for these embedded TransducerChannels
shall be appropriate to their function. The function of each embedded TransducerChannel is defined by
ControlGroups in the Meta-TEDS that are defined in 8.4.2.8.

5.6.2.5 Transitions reported

An event sensor may be designed to signal an event on falling transitions or rising transitions, or both. The
Edge-to-Report command (see 7.1.2.9) allows the system to select which transitions to report as events.
The edge-to-report capabilities of the event sensor along with the default condition are defined in the
TransducerChannel TEDS (see 8.5). If the event sensor is in the transducer operating state when an event
occurs, an event sensor shall set the data/event bit in the status register and issue a service request message
if the status-event protocol is enabled (see 7.1.1.11). If status-event protocol is disabled, the data/event bit
in the status register shall be set but no message shall be transmitted.

Whether or not an event has occurred since the last status read may be determined by examining the
data/event status bit as described in 5.13.6.

5.6.2.6 Event sensor status

An event sensor shall set the TransducerChannel data/event status bit whenever an event is detected in
either the transducer operating or transducer idle states. If the event sensor is in the transducer operating
state but has not been triggered, it shall set the TransducerChannel missed data or event status bit
coincident with any event that occurs. The TransducerChannel missed data or event status bit shall not be
set if the event sensor is in the transducer idle state.

Consistency checks may be made by the TIM if the bit pattern is changeable for a digital event sensor, or
the upper threshold or hysteresis is changeable for an analog event sensor. Inconsistent values are
signaled by setting the hardware error status bit (see 5.13.7). If inconsistent values are checked, the check
shall be made immediately following a change in any of these parameters. The check shall consist of
verifying the following relationships:

For analog values:

Maximum sensed input > upper threshold ≥ (upper threshold - hysteresis) > minimum sensed input value

For digital patterns: the pattern is a legal input.

NOTE—Consistency checking by the TIM shall only use data in the data model specified by the TransducerChannel
TEDS of the embedded TransducerChannels.

5.6.2.7 Continuous sampling event sensors

Like sensors and actuators, event sensors may be operated in the continuous sampling mode (see 5.10.1.6).
In the continuous sampling mode, an event sensor in the transducer operating state shall detect and place in
a buffer the fact that an event occurred without subsequent triggers.

5.6.2.8 Event sensors in the streaming data transmission mode

Like sensors and actuators, event sensors may be operated in the streaming data transmission mode. In the
Streaming when a buffer is full mode (see 5.10.2.2), an event sensor in the transducer operating state shall
transmit a message each time an event occurs. In the Streaming at a fixed interval data transmission mode (see 5.10.2.3) the event sensor shall store in a buffer the fact that an event or events has occurred and transmit the buffer at the appropriate time.

5.6.2.9 Time between events

The minimum time between events that an event sensor can detect is limited by the design of the event sensor. The minimum time between events that can be detected by the event sensor shall be specified in the TransducerChannel TEDS field, TransducerChannel sampling period, 8.5.2.36.

5.6.3 Actuator

An actuator shall cause a physical or embedded output action to occur. The actuator output state changes to match the appropriate data-set when a triggering event occurs. If there is more than one data point in a data-set, the interval between issuing the individual data points shall be under the control of the TIM.

An actuator may be built that does not require a data-set be written to it before it performs an action. This type of device always uses a default data-set or no data at all and performs a predefined action upon the receipt of a trigger.

5.7 Embedded TransducerChannels

Embedded TransducerChannels are TransducerChannels whose functionality is completely contained within a TIM. An embedded TransducerChannel does not sense or control any function outside of the TIM. For example, embedded actuators may be used to set the threshold and hysteresis of an event sensor. An embedded digital event sensor may be set up to detect and report any change in the status within a TIM.

Embedded TransducerChannels appear to the system as normal TransducerChannels. They respond to commands, have TEDS and are counted when determining the number of TransducerChannels in a TIM.

NOTE—The advantage of using "embedded" TransducerChannels over control commands is that "embedded" TransducerChannels are supported by TEDS, which gives the overall system a standard way to obtain the details of operation of the TIM. They should be used when it is difficult to specify the resolution of a control command in a standard way.

5.8 TransducerChannel groups

There are two types of TransducerChannel groups defined in this standard. They are ControlGroups and VectorGroups. They are similar in implementation but are used for two different functions.

5.8.1 ControlGroups

ControlGroups are used to define collections of TransducerChannels when one TransducerChannel is the primary channel and the remaining TransducerChannels in the group provide either additional information about the primary TransducerChannel or are used to control some aspect of the primary TransducerChannel. For example, a ControlGroup may be used to define up to three additional TransducerChannels associated with an analog event sensor. One is a sensor that is used to measure the analog input to the event sensor. The second is an actuator that is used to set the threshold for the event sensor. The third is an actuator that may be used to set the hysteresis for the event sensor.

5.8.2 VectorGroups

VectorGroups are used to define relationships between TransducerChannels within a single multi-channel TIM that imply a display or mathematical relationship between the TransducerChannels. For example, they may be used to identify the relationships between the components of a three-axis accelerometer.
VectorGroups are used by the software in the NCAP or host processor to group the outputs of the individual TransducerChannels into vectors for display or computational purposes.

NOTE—VectorGroups are in some ways similar to TransducerChannel proxies. TransducerChannel proxies are used to identify TransducerChannels that are grouped together for reasons such as efficient transmission of the data and/or simultaneous triggering. Proxies may or may not represent a vector for display or computational purposes. However, it is recommended that all vectors be implemented as proxies, especially in spatial vector applications (like velocity or acceleration), where the measurements of the three components at different points in time could lead to misinterpretation.

5.9 TransducerChannel proxy

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 read or written but it does not have the other characteristics of a TransducerChannel. This means that a proxy does not have a TransducerChannel TEDS, a Calibration TEDS, a Transfer Function TEDS or a Frequency Response TEDS. The TransducerChannel proxies that exist in a TIM are defined in the Meta-TEDS (see 8.4.2.14).

Like any other transducer, a TransducerChannel proxy is permanently assigned a TransducerChannel number by the manufacturer. Thus, a proxy may be addressed in a manner consistent with all other TransducerChannels implemented on the TIM.

A proxy shall not represent a collection of incompatible transducer types. In other words, a proxy represents a group of sensors or a group of actuators but shall not represent a group containing both transducer types.

Reading or writing the data from the individual members of a proxy may be allowed or disallowed at the manufacturer’s discretion. If the manufacturer chooses to disallow reading or writing the individual members of a proxy, then the receipt of one of those commands shall set the appropriate TransducerChannel command rejected bit (see 5.13.4) in the status word.

There are two methods that may be used to combine the data-sets for the proxy. They are the “block” and “interleave” methods. These two methods are shown graphically in Figure 6. The block method allows the data-sets to be of different lengths. With the interleave method all data-sets shall have the same size.
5.10 Attributes and operating modes

The subclauses 5.10.1 through 5.10.7.3 define the operating modes that are described by the attributes found in the TransducerChannel TEDS. Each attribute indicates whether or not a TransducerChannel supports the associated operating mode. Some of these modes of operation are mutually exclusive and others may be enabled at the same time.

5.10.1 Sampling modes

A sensor or actuator may be operated in one of five sampling modes. The sampling modes have different relationships between the trigger and the sampling of the data by a sensor or the application of a sample to the output of an actuator. The mode or modes that a TransducerChannel is capable of operating in is defined by the attributes given in the TransducerChannel TEDS. For the possible values of these attributes, see 8.5.2.44.

The trigger-initiated and free-running sampling modes are mutually exclusive. The TransducerChannel shall be operated in one of these two modes. The other three modes are variations on these two basic operating modes.

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 is 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 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 zero 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.

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 of the buffers are full, the data in the oldest buffer shall be discarded regardless of whether or not 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 or not 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 or not the
current buffer is full.

In the continuous sampling mode an actuator shall apply all of the data in its current buffer when the first
trigger is received at a rate controlled by the TransducerChannel. Once all of the 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

There are three data transmission modes defined in this standard as shown in Table 7.
### Table 7—Data transmission mode

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Argument name</th>
<th>Data Transmission Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>XmitMode.reserved[0]</td>
<td>Reserved</td>
<td>A TIM shall transmit a data-set only in response to a Read TransducerChannel data-set segment command (see 7.1.3.1).</td>
</tr>
<tr>
<td>1</td>
<td>XmitMode.OnCommand</td>
<td>Only when commanded</td>
<td>Data is 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).</td>
</tr>
<tr>
<td>2</td>
<td>XmitMode.BufferFull</td>
<td>Streaming when a buffer is full</td>
<td>Data buffers are transmitted at a fixed interval. The TransducerChannel shall stop using the current buffer regardless of how much data is in it, begin storing data in another buffer and transmit the buffer it was using without waiting for a Read TransducerChannel data-set segment command (see 7.1.3.1).</td>
</tr>
<tr>
<td>3</td>
<td>XmitMode.Interval</td>
<td>Streaming at a fixed interval</td>
<td>Data is transmitted 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.</td>
</tr>
<tr>
<td>4-127</td>
<td>XmitMode.reserved[N]</td>
<td>Reserved</td>
<td>The method of defining the sample interval is defined in the appropriate transmission media standard and is not covered in this standard.</td>
</tr>
<tr>
<td>128-255</td>
<td>XmitMode.open[N]</td>
<td>Open to manufacturers</td>
<td></td>
</tr>
</tbody>
</table>

#### 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.

#### 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 while other buffers...
are available to be filled. A characteristic of a TransducerChannel operating in the buffered mode is that the data available to be read or applied is always the data that was 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 zero 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 Error! Reference source not found.) was received have been read, the TransducerChannel or TransducerChannel proxy and shall return zero 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. There are two possible operating modes described in subclauses 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. While this 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 utilize all of the samples in a data-set and then continue to utilize 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 be associated with the trigger-initiated sampling mode.

5.10.4.2 Recirculate

An actuator operating in the recirculate mode shall apply all of the samples in a data-set to the output, then return to the beginning of the data-set and 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 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 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 show the triggering states of an actuator.

5.11.1 Sensor triggering

Movement from state to state is dependent upon 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.

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 upon 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.

There are several methods 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.

Figure 7—Sensor trigger states
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:

- a TransducerChannel
- a TransducerChannel Proxy
- a TIM
- an AddressGroup
- the 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...
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.

![Figure 9—Simple TransducerChannel functional blocks](image)

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.

$$T_i = T_N - (N - i)t_{si}$$

for $i = 1$ to $N$

where

- $N$ is the number of samples in the data-set
- $T_i$ is the time of sample $i$
- $T_N$ is the time of the last sample in the data set
- $t_{si}$ is the sample interval

**Equation 1**

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:

$$T_i = T_{m} + t_{pdii}$$

where

- $T_{m}$ is the time of day at which the trigger command was sent by the NCAP.
- $t_{pdii}$ is the value of the Incoming propagation delay through the data transport logic field found in the TransducerChannel TEDS.
- $T_i$ is the time of day when the first sample in the data-set was latched.

**Equation 2**

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.

$$T_i = T_{m} + i * t_{si}$$

for $i = 1$ to $N$

where
\[ N \] is the number of samples in the data-set
\[ T_i \] is the time of sample \( i \)
\[ T_{in} \] is the time of the first sample in the data set
\[ t_{si} \] is the sample interval

**Equation 3**

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.

![Functional block diagram](image)

**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 upon the operating mode of the TransducerChannel as described in 5.10.

<table>
<thead>
<tr>
<th>Operating mode</th>
<th>Response to the trigger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Streaming</td>
<td>Yes</td>
</tr>
<tr>
<td>Buffered</td>
<td>Don’t care</td>
</tr>
<tr>
<td>No</td>
<td>Ignore all except the first trigger</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Treat as a pre-trigger</td>
</tr>
<tr>
<td>No</td>
<td>Ignore triggers until previous operation complete</td>
</tr>
</tbody>
</table>

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 represent an accurate temporal “snapshot” of the measured parameters. There are several features built into this standard that are intended to aid in accomplishing this end. The following subclauses 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 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 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.

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.

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.

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.

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 four 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.”

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.

---

Figure 11—Status message generation logic
Table 9—Status bits

<table>
<thead>
<tr>
<th>Bit</th>
<th>TIM Status Bits</th>
<th>TransducerChannel Status Bits</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Service request</td>
<td>Service request</td>
<td>required</td>
</tr>
<tr>
<td>1</td>
<td>TEDS changed</td>
<td>TEDS changed</td>
<td>optional</td>
</tr>
<tr>
<td>2</td>
<td>Invalid command</td>
<td>reserved</td>
<td>optional</td>
</tr>
<tr>
<td>3</td>
<td>Command rejected</td>
<td>Command rejected</td>
<td>required</td>
</tr>
<tr>
<td>4</td>
<td>Missed data or event</td>
<td>Missed data or event</td>
<td>see 5.13.4</td>
</tr>
<tr>
<td>5</td>
<td>Data/event</td>
<td>Data/event</td>
<td>see 5.13.5</td>
</tr>
<tr>
<td>6</td>
<td>Hardware error</td>
<td>Hardware error</td>
<td>see 5.13.6</td>
</tr>
<tr>
<td>7</td>
<td>Not operational</td>
<td>Not operational</td>
<td>required</td>
</tr>
<tr>
<td>8</td>
<td>Protocol error</td>
<td>reserved</td>
<td>required</td>
</tr>
<tr>
<td>9</td>
<td>Data available/data processed</td>
<td>Data available/data processed</td>
<td>optional</td>
</tr>
<tr>
<td>10</td>
<td>Busy</td>
<td>Busy</td>
<td>optional</td>
</tr>
<tr>
<td>11</td>
<td>Failed calibration</td>
<td>Failed calibration</td>
<td>optional</td>
</tr>
<tr>
<td>12</td>
<td>Failed self-test</td>
<td>Failed self-test</td>
<td>optional</td>
</tr>
<tr>
<td>13</td>
<td>Data over range or under range</td>
<td>Data over range or under range</td>
<td>optional</td>
</tr>
<tr>
<td>14</td>
<td>Corrections disabled</td>
<td>Corrections disabled</td>
<td>optional</td>
</tr>
<tr>
<td>15</td>
<td>Consumables exhausted</td>
<td>Consumables exhausted</td>
<td>optional</td>
</tr>
<tr>
<td>16</td>
<td>reserved</td>
<td>Not-the-first-read-of-this-data-set</td>
<td>required</td>
</tr>
<tr>
<td>17-23</td>
<td>reserved</td>
<td>reserved</td>
<td>—</td>
</tr>
<tr>
<td>24-31</td>
<td>open to manufacturers</td>
<td>open to manufacturers</td>
<td>—</td>
</tr>
</tbody>
</table>

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.

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.

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 5.6.2).

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 of the 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 has been read. For an actuator it shall remain set until new data has been written to the device or the status is read.

The TransducerChannel data available/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 of the TransducerChannel busy bits in a TIM.
This bit is not affected by a change of operating state.

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 of the 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 of the TransducerChannel self-tests 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 of the 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 of the TransducerChannel consumables exhausted bits in a TIM. This status bit shall only be set while the condition persists.

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 four 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 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 all the TransducerChannels’ status 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.
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 IEEE p1451.0 but is dependent upon the capabilities of the underlying physical layers ability to detect such changes and 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>
<thead>
<tr>
<th>1-Octet</th>
<th>1-Octet</th>
<th>1-Octet</th>
<th>1-Octet</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2 1 0</td>
<td>7 6 5 4 3 2 1 0</td>
<td>7 6 5 4 3 2 1 0</td>
<td>7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>1st octet transmitted</td>
<td>2 3</td>
<td>4</td>
<td>5 6 7 8</td>
</tr>
</tbody>
</table>
| ... | ...

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
Whenever an octet represents a numeric quantity, the left-most bit 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>
<thead>
<tr>
<th>Bits</th>
<th>Value=170 (decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

6.2 Command message structure

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

<table>
<thead>
<tr>
<th>1-Octet</th>
<th>Destination TransducerChannel Number (most significant octet)</th>
<th>Destination TransducerChannel Number (least significant octet)</th>
<th>Command Class</th>
<th>Command Function</th>
<th>Length (most significant octet)</th>
<th>Length (least significant octet)</th>
<th>Command dependent octets</th>
</tr>
</thead>
</table>

6.2.1 Destination TransducerChannel number

This field gives the sixteen-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.

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>
<thead>
<tr>
<th>Table 13—Reply message structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Octet</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>Success/Fail Flag</td>
</tr>
<tr>
<td>Length (most significant octet)</td>
</tr>
<tr>
<td>Length (least significant octet)</td>
</tr>
<tr>
<td>Reply dependent octets</td>
</tr>
<tr>
<td>.</td>
</tr>
</tbody>
</table>

6.3.1 Success/Fail flag

If this octet is non-zero 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.

<table>
<thead>
<tr>
<th>Table 14—TIM initiated message structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Octet</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>Source TransducerChannel Number (most significant octet)</td>
</tr>
<tr>
<td>Source TransducerChannel Number (least significant octet)</td>
</tr>
<tr>
<td>Command Class</td>
</tr>
<tr>
<td>Command Function</td>
</tr>
<tr>
<td>Length (most significant octet)</td>
</tr>
<tr>
<td>Length (least significant octet)</td>
</tr>
<tr>
<td>Command dependent octets</td>
</tr>
<tr>
<td>.</td>
</tr>
</tbody>
</table>

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
6.4.1 Source TransducerChannel number

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

6.4.2 Command class

This is the same as 6.2.2.

6.4.3 Command function

This is the same as 6.2.3.

6.4.4 Length

This is the same as 6.2.4.

6.4.5 Command dependent octets

This 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 two 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.

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 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 utilized.

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.
### Table 15—Standard command classes

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

#### 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.

### Table 16—Common commands

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

#### 7.1.1.1 Query TEDS command

Argument attribute name: TEDSAccessCode datatype 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.
### Table 17—TEDS access codes

<table>
<thead>
<tr>
<th>TEDS Code</th>
<th>Access</th>
<th>TEDS name attribute</th>
<th>TEDS</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>—</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>MetaTEDS</td>
<td>Meta-TEDS1</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>MetaIdTEDS</td>
<td>Meta-Identification TEDS2</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ChanTEDS</td>
<td>TransducerChannel TEDS1</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ChanIdTEDS</td>
<td>TransducerChannel Identification TEDS2</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CalTEDS</td>
<td>Calibration TEDS1</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>CalIdTEDS</td>
<td>Calibration Identification TEDS2</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>EUASTEDS</td>
<td>End Users’ Application Specific TEDS3</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>FreqRespTEDS</td>
<td>Frequency Response TEDS1</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>TransferTEDS</td>
<td>Transfer Function TEDS1</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>CommandTEDS</td>
<td>Commands TEDS2</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>TitleTEDS</td>
<td>Location and Title TEDS2</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>XdcrName</td>
<td>User’s Transducer Name TEDS3</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>PHYTEDS</td>
<td>PHY TEDS1</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>GeoLocTEDS</td>
<td>Geographic location TEDS2</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>UnitsExtention</td>
<td>Units extention TEDS2</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>16 – 127</td>
<td>—</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>128 – 255</td>
<td>—</td>
<td>Manufacturer defined TEDS</td>
<td>Optional</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES**

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

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 or not the TEDS access code selects an implemented TEDS. The attributes field in the reply (see Table 19) shall indicate if the TEDS is supported, if it may be changed, if the current contents are valid and if 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.
Table 18—Query TEDS response in the data field

<table>
<thead>
<tr>
<th>Field</th>
<th>datatype</th>
<th>Field attribute name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UInt8</td>
<td>TEDSAttrib</td>
<td>TEDS Attributes (see Table 19)</td>
</tr>
<tr>
<td>2</td>
<td>UInt8</td>
<td>TEDSStatus</td>
<td>TEDS Status</td>
</tr>
<tr>
<td>3</td>
<td>UInt32</td>
<td>TEDSSize</td>
<td>Current size of the TEDS</td>
</tr>
<tr>
<td>4</td>
<td>UInt16</td>
<td>TEDSCkSum</td>
<td>TEDS Checksum</td>
</tr>
<tr>
<td>5</td>
<td>UInt32</td>
<td>MaxTEDSSize</td>
<td>Maximum TEDS Size</td>
</tr>
</tbody>
</table>

Table 19—TEDS attributes

<table>
<thead>
<tr>
<th>Bit</th>
<th>datatype</th>
<th>Field attribute name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (lsb)</td>
<td>Boolean</td>
<td>TEDSAttrib.ReadOnly</td>
<td>Read-only – Set to true if TEDS may be read but not written</td>
</tr>
<tr>
<td>1</td>
<td>Boolean</td>
<td>TEDSAttrib.NotAvail</td>
<td>Unsupported – Set to true if TEDS is not supported by this TransducerChannel.</td>
</tr>
<tr>
<td>2</td>
<td>Boolean</td>
<td>TEDSAttrib.Invalid</td>
<td>Invalid – Set to true if the current TEDS image is invalid</td>
</tr>
<tr>
<td>3</td>
<td>Boolean</td>
<td>TEDSAttrib.Virtual</td>
<td>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.</td>
</tr>
<tr>
<td>4</td>
<td>Boolean</td>
<td>TEDSAttrib.TextTEDS</td>
<td>Text TEDS – Set to true if the TEDS is text based.</td>
</tr>
<tr>
<td>5</td>
<td>Boolean</td>
<td>TEDSAttrib.Adaptive</td>
<td>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.</td>
</tr>
<tr>
<td>6</td>
<td>Boolean</td>
<td>TEDSAttrib.MfgrDefine</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 20—TEDS status

<table>
<thead>
<tr>
<th>Bit</th>
<th>datatype</th>
<th>Field attribute name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (lsb)</td>
<td>Boolean</td>
<td>TEDSStatus.TooLarge</td>
<td>Too Large – The last TEDS image received was too large to fit in the memory allocated to this TEDS.</td>
</tr>
<tr>
<td>1</td>
<td>Boolean</td>
<td>TEDSStatus.Reserved[1]</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>Boolean</td>
<td>TEDSStatus.Open[5]</td>
<td>Open to manufacturers</td>
</tr>
<tr>
<td>6</td>
<td>Boolean</td>
<td>TEDSStatus.Open[6]</td>
<td>Open to manufacturers</td>
</tr>
<tr>
<td>7 (msb)</td>
<td>Boolean</td>
<td>TEDSStatus.Open[7]</td>
<td>Open to manufacturers</td>
</tr>
</tbody>
</table>

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
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.

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 zero 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 zero octets.

Table 21—Data field for a read TEDS segment command

<table>
<thead>
<tr>
<th>Field</th>
<th>datatype</th>
<th>Field attribute name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Uint8</td>
<td>TEDSAccessCode</td>
<td>TEDS access code, as defined in Table 17.</td>
</tr>
<tr>
<td>2</td>
<td>UInt32</td>
<td>TEDSOffset</td>
<td>TEDS Segment offset (0 to ([\text{current size} – 1])) – This is the address relative to the beginning of the TEDS at which the block of data shall be read.</td>
</tr>
</tbody>
</table>

Table 22—Data field for a TEDS segment command reply

<table>
<thead>
<tr>
<th>Field</th>
<th>datatype</th>
<th>Field attribute name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UInt32</td>
<td>TEDSOffset</td>
<td>TEDS Segment Offset (0 to ([\text{current size} – 1]))</td>
</tr>
<tr>
<td>2</td>
<td>OctetArray</td>
<td>RawTEDSBlock</td>
<td>TEDS data octets</td>
</tr>
</tbody>
</table>

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.

Table 23—Data field for a write TEDS command

<table>
<thead>
<tr>
<th>Field</th>
<th>datatype</th>
<th>Field attribute name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Uint8</td>
<td>TEDSAccessCode</td>
<td>TEDS access code, as defined in Table 17.</td>
</tr>
<tr>
<td>2</td>
<td>UInt32</td>
<td>TEDSOffset</td>
<td>TEDS Segment offset (0 to ([\text{current size} – 1]))</td>
</tr>
<tr>
<td>3</td>
<td>OctetArray</td>
<td>RawTEDSBlock</td>
<td>TEDS Contents</td>
</tr>
</tbody>
</table>
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 datatype 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 datatype 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 eight 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.

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

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.

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Argument attribute name</th>
<th>Diagnostic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Test2Run.ConfidenceTest</td>
<td>Short confidence test</td>
</tr>
<tr>
<td>1</td>
<td>Test2Run.TestAll</td>
<td>Test all</td>
</tr>
<tr>
<td>2 – 255</td>
<td>Test2Run.RunTest[N] 2&lt;=N&lt;=255</td>
<td>Manufacturer-defined</td>
</tr>
</tbody>
</table>

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 datatype UInt32

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

7.1.1.7 Read service request mask

Reply argument attribute name: SRMask datatype 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 datatype 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 datatype 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.

7.1.1.11 Write status-event protocol state

Argument attribute name: SEProtocol datatype 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 datatype 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.
### Table 25—Transducer Idle state commands

<table>
<thead>
<tr>
<th>cmdFunctionId</th>
<th>Command</th>
<th>Address class</th>
<th>Proxy</th>
<th>Group</th>
<th>Reply Expected</th>
<th>Required</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td>——</td>
<td>——</td>
<td>——</td>
<td>——</td>
<td>——</td>
<td>——</td>
</tr>
<tr>
<td>1</td>
<td>Set TransducerChannel data repetition count</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Set TransducerChannel pre-trigger count</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>AddressGroup definition</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Sampling mode</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Data Transmission mode</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Buffered state</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>End-of-data-set operation</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Actuator-halt mode</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Edge-to-report</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Calibrate TransducerChannel</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Zero TransducerChannel</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Write corrections state</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>see 7.1.2.12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Read corrections state</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>see 7.1.2.13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Write TransducerChannel initiate trigger state</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Write TransducerChannel initiate trigger configuration</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>16 – 127</td>
<td>Reserved</td>
<td>——</td>
<td>——</td>
<td>——</td>
<td>——</td>
<td>——</td>
<td>——</td>
</tr>
<tr>
<td>128 – 255</td>
<td>Open for manufacturers</td>
<td>——</td>
<td>——</td>
<td>——</td>
<td>——</td>
<td>——</td>
<td>——</td>
</tr>
</tbody>
</table>

**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 datatype 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 sixteen-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 datatype 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).

The only argument to this command is a sixteen-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** datatype 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 in 5.3 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 of the AddressGroups to which it has been assigned.

### 7.1.2.4 Sampling mode

Argument attribute name: **SampleMode** datatype UInt8

The NCAP uses this command when changing the sampling mode of a TransducerChannel. Not all of the 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

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Argument attribute name</th>
<th>Operating Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SampleMode.Reserved[0]</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>SampleMode.TriggerInit</td>
<td>Trigger-initiated</td>
</tr>
<tr>
<td>2</td>
<td>SampleMode.FreeNoPre</td>
<td>Free-running without pre-trigger</td>
</tr>
<tr>
<td>3</td>
<td>SampleMode.FreePreTrig</td>
<td>Free-running with pre-trigger</td>
</tr>
<tr>
<td>4</td>
<td>SampleMode.Continuous</td>
<td>Continuous Sampling</td>
</tr>
<tr>
<td>5</td>
<td>SampleMode.Immediate</td>
<td>Immediate Operation</td>
</tr>
<tr>
<td>6-127</td>
<td>SampleMode.Reserved[N]</td>
<td>6&lt;=N&lt;=127 Reserved</td>
</tr>
<tr>
<td>128-255</td>
<td>SampleMode.Open[N]</td>
<td>128&lt;=N&lt;=255 Open to manufacturers</td>
</tr>
</tbody>
</table>

### 7.1.2.5 Data Transmission mode

Argument attribute name: **XmitMode** datatype UInt8

The command is used to control the data transmission mode of operation as described in 5.10.2. Whether or not 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.

7.1.2.6 **Buffered state**

Argument attribute name: BufferOnOff datatype Boolean

The command is used to turn on or off buffered operation as described in 5.10.3. Whether or not 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 datatype 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 or not 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>
<thead>
<tr>
<th>Enumeration</th>
<th>Argument attribute name</th>
<th>Operating Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EndDataSetMode.Reserved[0]</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>EndDataSetMode.Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>2</td>
<td>EndDataSetMode.Recirc</td>
<td>Recirculate</td>
</tr>
<tr>
<td>3-127</td>
<td>EndDataSetMode.Reserved[N] 3&lt;=N&lt;=127</td>
<td>Reserved</td>
</tr>
<tr>
<td>128-256</td>
<td>EndDataSetMode.Open[N] 128&lt;=N&lt;=255</td>
<td>Open to manufacturers</td>
</tr>
</tbody>
</table>

7.1.2.8 **Actuator halt operating mode**

Argument attribute name: ActuatorHalt datatype 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

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Argument attribute name</th>
<th>Operating Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ActuatorHalt.Reserved[0]</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>ActuatorHalt.Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>2</td>
<td>ActuatorHalt.Finish</td>
<td>Finish data-set</td>
</tr>
<tr>
<td>3</td>
<td>ActuatorHalt.Ramp</td>
<td>Ramp</td>
</tr>
<tr>
<td>4-127</td>
<td>ActuatorHalt.Reserved[N]</td>
<td>4&lt;=N&lt;=127</td>
</tr>
<tr>
<td>128-256</td>
<td>ActuatorHalt.Open[N]</td>
<td>128&lt;=N&lt;=255</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Open to manufacturers</td>
</tr>
</tbody>
</table>

7.1.2.9 Edge-to-Report

Argument attribute name: EdgeReported datatype 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

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Argument attribute name</th>
<th>Operating Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EdgeReported.Reserved[0]</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>EdgeReported.Rising</td>
<td>Report Rising Edges</td>
</tr>
<tr>
<td>2</td>
<td>EdgeReported.Falling</td>
<td>Report Falling Edges</td>
</tr>
<tr>
<td>3</td>
<td>EdgeReported.BothEdges</td>
<td>Report both Edges</td>
</tr>
<tr>
<td>4-127</td>
<td>EdgeReported.Reserved[N]</td>
<td>4&lt;=N&lt;=127</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>128-256</td>
<td>EdgeReported.Open[N]</td>
<td>128&lt;=N&lt;=255</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Open to manufacturers</td>
</tr>
</tbody>
</table>

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 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 datatype 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.

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 datatype 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 datatype 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 datatype 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

<table>
<thead>
<tr>
<th>Field</th>
<th>datatype</th>
<th>Argument attribute name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UInt16</td>
<td>InitTrig.destId</td>
<td>The “destId” specifies the desired destination</td>
</tr>
<tr>
<td>2</td>
<td>UInt16</td>
<td>InitTrig.channelId</td>
<td>The “channelId” specifies the desired TransducerChannel.</td>
</tr>
<tr>
<td>3</td>
<td>Struct</td>
<td>InitTrig.qosParams</td>
<td>The “qosParams” is the desired quality of service parameters. See 9.3.1.3 for details.</td>
</tr>
</tbody>
</table>
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.

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

<table>
<thead>
<tr>
<th>cmdFunctionId</th>
<th>Command</th>
<th>Address class</th>
<th>Proxy</th>
<th>Group</th>
<th>Global</th>
<th>Reply</th>
<th>Expected</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Read TransducerChannel data-set segment</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>See note</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Write TransducerChannel data-set segment</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>See note</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Trigger command</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Abort Trigger</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 – 127</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128 – 255</td>
<td>Open for manufacturers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 datatype 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, that 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 zero 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 zero octets.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data type</th>
<th>Argument attribute name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UInt32</td>
<td>ReadSensorOffset</td>
<td>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.</td>
</tr>
<tr>
<td>2</td>
<td>N*UInt8</td>
<td>ReadSensorData</td>
<td>The block of data read from the sensor.</td>
</tr>
</tbody>
</table>

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 65,536 and the data model, that 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>
<thead>
<tr>
<th>Field</th>
<th>Datatype</th>
<th>Argument attribute name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UInt32</td>
<td>WriteActuator.Offset</td>
<td>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.</td>
</tr>
<tr>
<td>2</td>
<td>N*UInt8</td>
<td>WriteActuator.DataBlock</td>
<td>Data block</td>
</tr>
</tbody>
</table>

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 \textit{trigger commands} in 5.11.2.1.

There are no arguments to this command.

It may be addresses 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 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.

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 sub-clause 5.4.1 for a description of the operating states of a TransducerChannel.

This command has no arguments.
### Table 34—TransducerChannel idle or operational state commands

<table>
<thead>
<tr>
<th>cmdFunctionId</th>
<th>Command</th>
<th>Address class</th>
<th>Proxy</th>
<th>Group</th>
<th>Global</th>
<th>Reply</th>
<th>Expected</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TransducerChannel Operate</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Required</td>
</tr>
<tr>
<td>2</td>
<td>TransducerChannel Idle</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Required</td>
</tr>
<tr>
<td>3</td>
<td>Write TransducerChannel trigger state</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Optional</td>
</tr>
<tr>
<td>4</td>
<td>Read TransducerChannel trigger state</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Required</td>
</tr>
<tr>
<td>5</td>
<td>Read TransducerChannel data repetition count</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>See Note 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Read TransducerChannel pre-trigger count</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>See Note 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Read AddressGroup assignment</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Read Sampling mode</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
</tr>
<tr>
<td>9</td>
<td>Read data transmission mode</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
</tr>
<tr>
<td>10</td>
<td>Read buffered state</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
</tr>
<tr>
<td>11</td>
<td>Read end-of-data-set operation</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
</tr>
<tr>
<td>12</td>
<td>Read actuator halt mode</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
</tr>
<tr>
<td>13</td>
<td>Read edge-to-report mode</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
</tr>
<tr>
<td>14</td>
<td>Read TransducerChannel initiate trigger state</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
</tr>
<tr>
<td>15</td>
<td>Read TransducerChannel initiate configuration</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
</tr>
<tr>
<td>16</td>
<td>Device clear</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Optional</td>
</tr>
<tr>
<td>17 – 127</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128 – 255</td>
<td>Open for manufacturers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

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 sub-clause 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 datatype 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 datatype 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.

7.1.4.5 Read TransducerChannel data repetition count

Reply argument attribute name: RepCount datatype 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 datatype 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 datatype 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 datatype 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 datatype 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).

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 datatype 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 datatype 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 datatype 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 datatype 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 datatype Boolean

This command allows an NCAP to determine whether or not 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.
7.1.4.15 Read TransducerChannel initiate trigger configuration

Reply argument attribute name: InitTrig datatype Boolean

This command is used to read the information needed to allow a TransducerChannel to 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. See Figure 7 for sensor trigger states and Figure 8 for actuator trigger state diagrams for more information on device clear.

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>
<thead>
<tr>
<th>cmdFunctionId</th>
<th>Command</th>
<th>Address class</th>
<th>Reply</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td>——</td>
<td>——</td>
<td>——</td>
</tr>
<tr>
<td>1</td>
<td>Wake-up</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>2 – 127</td>
<td>Reserved</td>
<td>——</td>
<td>——</td>
<td>——</td>
</tr>
<tr>
<td>128 – 255</td>
<td>Open for manufacturers</td>
<td>——</td>
<td>——</td>
<td>——</td>
</tr>
</tbody>
</table>

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.

7.1.6 TIM active state commands

The following commands may only be executed when the TIM is in the active state. 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.

Table 35—Sleep state commands

<table>
<thead>
<tr>
<th>cmdFunctionId</th>
<th>Command</th>
<th>Address class</th>
<th>Reply</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td>——</td>
<td>——</td>
<td>——</td>
</tr>
<tr>
<td>1</td>
<td>Wake-up</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>2 – 127</td>
<td>Reserved</td>
<td>——</td>
<td>——</td>
<td>——</td>
</tr>
<tr>
<td>128 – 255</td>
<td>Open for manufacturers</td>
<td>——</td>
<td>——</td>
<td>——</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>cmdFunctionId</th>
<th>Command</th>
<th>Address class</th>
<th>TIM</th>
<th>Global</th>
<th>Reply Expected</th>
<th>Required</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>Read TIM version</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TIM Sleep</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Store operational Setup</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Recall operational Setup</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Read IEEE p1451.0 Version</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 – 127</td>
<td>Reserved—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>128 – 255</td>
<td>Open for manufacturers</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 7.1.6.1 Read TIM version

Reply argument attribute name: TIMVersion datatype 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<=N<=255 datatype 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. There are two conditions that will cause the state to be restored. They are as follows:

- Receipt of the recall state command or
- Return from a power failure.

NOTE—This should include the signal conditioning parameters, the operation modes, etc.

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 datatype 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 IEEE p1451.0 version

Argumet attribute name: StandardVersion datatype UInt8

This command is used to read the version of IEEE p1451.0 supported by this TIM.

This command has no arguments. The reply to this command shall have a single octet of the IEEE p1451.0 version number as shown in Table 37 in the data field.

Table 37—Enumeration of IEEE p1451.0 version numbers

<table>
<thead>
<tr>
<th>Standard version</th>
<th>IEEE P1451.0 standard version</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved for prototype or other non standard IEEE p1451.0 Versions</td>
</tr>
<tr>
<td>1</td>
<td>This will correspond to the original release of IEEE p1451.0.</td>
</tr>
<tr>
<td>2-127</td>
<td>Reserved for future releases of the standard.</td>
</tr>
<tr>
<td>128-255</td>
<td>Open to manufacturers</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>cmdFunctionId</th>
<th>Command</th>
<th>Address class</th>
<th>Reply Expected</th>
<th>Required Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>Reset</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>2 – 127</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>128 – 255</td>
<td>Open for manufacturers</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

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 non-standard 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 utilize these commands.

8 Transducer Electronic Data Sheet (TEDS) specification

This clause specifies the contents of all of the 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 four octet unsigned integer. The next block is the information content for the TEDS. Depending upon 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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th># octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>TEDS length</td>
<td>UInt32</td>
<td>4</td>
</tr>
<tr>
<td>1 to N</td>
<td>Data block</td>
<td>Variable</td>
<td>Variable</td>
</tr>
<tr>
<td>—</td>
<td>Checksum</td>
<td>UInt16</td>
<td>2</td>
</tr>
</tbody>
</table>

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

Using the TLV construct as shown in Table 40, each entry is stored as a type/length/value (TLV) tuple. The "Type" field is one 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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>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.</td>
</tr>
<tr>
<td>Length</td>
<td>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.</td>
</tr>
<tr>
<td>Value</td>
<td>This field contains the TEDS information.</td>
</tr>
</tbody>
</table>

8.1.3 Unused type codes

Within the definition of each TEDS there are some type codes that 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 IEEE p1451.0 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 did 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 p1451.0 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 did 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 p1451.0 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.

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.

\[
\text{checksum} = 0xFFFF - \sum_{i=1}^{\text{TotalOctets}-2} \text{TEDSOctet}(i)
\]
Equation 4—Checksum calculation

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 sixteen bit number. If the sixteen bit number overflows, ignore the overflow and keep only the lower sixteen bits. Take the logical (one’s) complement of the resulting sixteen-bit number.

8.2 Order of Octets in numeric fields

For numeric values requiring more than one 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 is a required field in all TEDS.

The TEDS Identifier consists of the four fields shown in Table 41 and is standard for all TEDS. This is always the first field 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 IEEE p1451.0)

TEDS Class

Version Number

Tuple Length

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>03</td>
<td>Type field for the TEDS Identifier</td>
</tr>
<tr>
<td>Length</td>
<td>04</td>
<td>This field is always set to 04 indicating that the value field contains four octets</td>
</tr>
<tr>
<td>Family</td>
<td>00</td>
<td>This field identifies the member of the IEEE 1451 family of standards that defines this TEDS.</td>
</tr>
<tr>
<td>Class</td>
<td>See Table 17</td>
<td>This field identifies the TEDS being accessed. The value is the TEDS access code found in Table 17.</td>
</tr>
<tr>
<td>Version</td>
<td>See Table 42</td>
<td>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.</td>
</tr>
<tr>
<td>Tuple Length</td>
<td>Number of octets</td>
<td>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 eight 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.</td>
</tr>
</tbody>
</table>
### Table 42—Enumeration of TEDS version numbers

<table>
<thead>
<tr>
<th>TEDS Version</th>
<th>IEEE P1451.0 Standard Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved for prototype or other non standard TEDS Versions</td>
</tr>
<tr>
<td>1</td>
<td>This will correspond to the original release of IEEE P1451.0.</td>
</tr>
<tr>
<td>2-255</td>
<td>Reserved for future releases of the standard.</td>
</tr>
</tbody>
</table>

#### 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 of the 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 (Sec 8.1).

##### 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
Data type: UUID, 10 octets

This field shall be present in the Meta-TEDS of each TIM and shall be unique. If this field is omitted the NCAP shall report a fatal TEDS error.

See clause 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 are intended for detecting non-responsive 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.

### Table 43—Structure of the Meta-TEDS data block

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Field Name</th>
<th>Description</th>
<th>Data Type</th>
<th># octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>Length</td>
<td></td>
<td>UInt32</td>
<td>4</td>
</tr>
<tr>
<td>0-2</td>
<td>Reserved</td>
<td></td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>TEDSID</td>
<td>TEDS Identification Header</td>
<td>UInt8</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>UUID</td>
<td>Globally Unique Identifier</td>
<td>UUID</td>
<td>10</td>
</tr>
<tr>
<td>5-9</td>
<td>Reserved</td>
<td></td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>10</td>
<td>OholdOff</td>
<td>Operational time-out</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>SHoldOff</td>
<td>Slow access timeout</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>TestTime</td>
<td>Self-Test Time</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>MaxChan</td>
<td>Number of implemented TransducerChannels</td>
<td>UInt16</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>CGroup</td>
<td>ControlGroup information sub-block</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>20</td>
<td>GrpType</td>
<td>ControlGroup type</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>MemList</td>
<td>ControlGroup member list</td>
<td>array of UInt16</td>
<td>NTc</td>
</tr>
<tr>
<td>15</td>
<td>VGroup</td>
<td>VectorGroup information sub-block</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>20</td>
<td>GrpType</td>
<td>VectorGroup type</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>MemList</td>
<td>VectorGroup member list</td>
<td>array of UInt16</td>
<td>NTv</td>
</tr>
<tr>
<td>16</td>
<td>GeoLoc</td>
<td>Specialized VectorGroup for geographic location</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>24</td>
<td>LocEnum</td>
<td>An enumeration defining how location information is provided.</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>GrpType</td>
<td>VectorGroup type</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>MemList</td>
<td>VectorGroup member list</td>
<td>array of UInt16</td>
<td>NTv</td>
</tr>
<tr>
<td>17</td>
<td>Proxies</td>
<td>TransducerChannel proxy definition sub-block</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>22</td>
<td>ChanNum</td>
<td>TransducerChannel number of the TransducerChannel proxy</td>
<td>UInt16</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>Organiz</td>
<td>TransducerChannel proxy data-set organization</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>MemList</td>
<td>TransducerChannel proxy member list</td>
<td>array of UInt16</td>
<td>NTp</td>
</tr>
<tr>
<td>18-19</td>
<td>Reserved</td>
<td></td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>25-127</td>
<td>Reserved</td>
<td></td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>128-255</td>
<td>Open to manufacturers</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>—</td>
<td>Checksum</td>
<td></td>
<td>UInt16</td>
<td>2</td>
</tr>
</tbody>
</table>
The * indicates that this element will exist zero or more times within the TEDS.

```
    TEDSId
     Type = 03

    UUID
     Type = 04

    OHoldOff
     Type = 10

    SHoldOff
     Type = 11

    TestTime
     Type = 12

    MaxChan
     Type = 13

    CGroup*
     Type = 14
       GrpType
        Type = 20
       MemList
        Type = 21

    VGroup*
     Type = 15
       GrpType
        Type = 20
       MemList
        Type = 21

    GeoLoc*
     Type = 16
       GrpType
        Type = 20
       MemList
        Type = 21
       ChanNum
        Type = 22

    Proxies*
     Type = 17
       Organiz
        Type = 23
       MemList
        Type = 21
```

Figure 13—Structure of the Meta-TEDS data block
8.4.2.4 Operational time-out

Field Type: 10

Field Name: OHoldOff

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

This is a required field. 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 timeout value.

8.4.2.5 Slow access timeout

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 is an optional field. If a single timeout 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 timeout value.

8.4.2.6 Transducer module self-test time requirement

Field Type: 12

Field Name: TestTime

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

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

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 sixteen 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 a 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 sub-fields (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.

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

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

#### 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.

#### 8.4.2.10 Member list

- **Field Type**: 21
Field Name: MemList

Data type: a one dimensional array of unsigned sixteen-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 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 sub-fields:

- 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 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.

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 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

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>member list order</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>—</td>
<td>an arbitrary relation</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>x component of a right hand rectangular spatial vector</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>y component of a right hand rectangular spatial vector</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>z component of a right hand rectangular spatial vector</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>ρ component of a right hand cylindrical spatial vector</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Φ component of a right hand cylindrical spatial vector</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>z component of a right hand cylindrical spatial vector</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>r component of a right hand spherical spatial vector</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>θ component of a right hand spherical spatial vector</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>φ component of a right hand spherical spatial vector</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>latitude component of a planetary coordinate system</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>longitude component of a planetary coordinate system</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>altitude component of a planetary coordinate system</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>in-phase component of a 2-dimensional vector</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>quadrature component of a 2-dimensional vector</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>red component in a color vector</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>green component in a color vector</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>blue component in a color vector</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>real component of a complex number</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>imaginary component of a complex number</td>
</tr>
<tr>
<td>8-127</td>
<td></td>
<td>Reserved for future expansion</td>
</tr>
<tr>
<td>128-255</td>
<td></td>
<td>open to manufacturers</td>
</tr>
</tbody>
</table>

### 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 sub-fields (TLV tuples):

<table>
<thead>
<tr>
<th>LocEnum</th>
<th>Group type</th>
<th>Member list</th>
</tr>
</thead>
</table>

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

### 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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No geographic location information is provided by this TIM</td>
</tr>
<tr>
<td>1</td>
<td>Static geographic location information is provided via the geographic location TEDS</td>
</tr>
<tr>
<td>2</td>
<td>Dynamic geographic location information is provided</td>
</tr>
<tr>
<td>3</td>
<td>Dynamic geographic location information is provided that is relative to the location specified in the geographic location TEDS</td>
</tr>
<tr>
<td>4-255</td>
<td>Reserved for future expansion</td>
</tr>
</tbody>
</table>

#### 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, 2, 3 or 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 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 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.

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 sub fields 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 sixteen-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: 

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.

Table 47—Enumerations for combined data-sets

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Block method</td>
</tr>
<tr>
<td>1</td>
<td>Interleave method 1</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>3-255</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

8.4.2.21 TransducerChannel proxy member list

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

This is a list of transducer channel numbers that are 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 49 shows 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.

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 48 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.
<table>
<thead>
<tr>
<th>Value</th>
<th>Calibration capability</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No calibration information needed or provided by the transducer module. This implies</td>
<td>CAL_NONE</td>
</tr>
<tr>
<td></td>
<td>that there is no Calibration TEDS associated with this TransducerChannel. If an attempt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>is made to access the Calibration TEDS, the Calibration TEDSLength shall be zero.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Calibration information is provided as a Calibration TEDS, as specified in 8.5.2.55.</td>
<td>CAL_SUPPLIED</td>
</tr>
<tr>
<td></td>
<td>Correction is performed outside of the TIM.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(reserved)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Calibration information is provided but in a form that is not described in this</td>
<td>CAL_CUSTOM</td>
</tr>
<tr>
<td></td>
<td>standard. Correction is performed outside of the TIM.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Calibration information is provided as a Calibration TEDS, as specified in 8.5.2.55.</td>
<td>TIM_CAL_SUPPLIED</td>
</tr>
<tr>
<td></td>
<td>Correction is applied in the TIM.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Calibration information is provided as a Calibration TEDS, as specified in 8.5.2.55,</td>
<td>TIM_CAL_SELF</td>
</tr>
<tr>
<td></td>
<td>and is applied in the TIM. The calibration information is adjusted by a self-calibration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>capability.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Calibration information is provided but in a form that is not described in this</td>
<td>TIM_CAL_CUSTOM</td>
</tr>
<tr>
<td></td>
<td>standard. It may be a manufacturer-defined TEDS. Correction is performed in the TIM.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If an attempt is made to access the Calibration TEDS, the Calibration TEDSLength shall</td>
<td></td>
</tr>
<tr>
<td></td>
<td>be zero.</td>
<td></td>
</tr>
<tr>
<td>7-255</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>
### Table 49—Structure of the TransducerChannel TEDS data block

<table>
<thead>
<tr>
<th>Field</th>
<th>Field Name</th>
<th>Description</th>
<th>Type</th>
<th># octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-2</td>
<td>TEDS length</td>
<td>TEDS length</td>
<td>UInt32</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>TEDSId</td>
<td>TEDS Identification</td>
<td>UInt8</td>
<td>4</td>
</tr>
<tr>
<td>4-9</td>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TransducerChannel related information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>CalKey</td>
<td>Calibration key</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>ChanType</td>
<td>TransducerChannel type key</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>PhyUnits</td>
<td>Physical units</td>
<td>UINT8</td>
<td>11</td>
</tr>
<tr>
<td>50</td>
<td>UnitType</td>
<td>Physical units interpretation enumeration</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>51</td>
<td>Radians</td>
<td>The exponent for Radians</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>52</td>
<td>SterRad</td>
<td>The exponent for Steradians</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>53</td>
<td>Meters</td>
<td>The exponent for Meters</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>54</td>
<td>Kilogram</td>
<td>The exponent for Kilograms</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>55</td>
<td>Seconds</td>
<td>The exponent for Seconds</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>56</td>
<td>Amperes</td>
<td>The exponent for Amperes</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>57</td>
<td>Kelvins</td>
<td>The exponent for Kelvins</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>58</td>
<td>Moles</td>
<td>The exponent for Moles</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>59</td>
<td>Candelas</td>
<td>The exponent for Candelas</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>60</td>
<td>UnitsExt</td>
<td>TEDS access code for units extension</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>LowLimit</td>
<td>Design operational lower range limit</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td>HiLimit</td>
<td>Design operational upper range limit</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>OError</td>
<td>Worst-case uncertainty</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>SelfTest</td>
<td>Self-test key</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>MRange</td>
<td>Multi-range capability</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Data converter related information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Sample</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>DatModel</td>
<td>Data model</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>41</td>
<td>ModLenth</td>
<td>Data model length</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>SigBits</td>
<td>Model significant bits</td>
<td>UInt16</td>
<td>2</td>
</tr>
<tr>
<td>43</td>
<td>Repeats</td>
<td>Maximum data repetitions</td>
<td>UInt16</td>
<td>2</td>
</tr>
<tr>
<td>44</td>
<td>Soorigin</td>
<td>Series origin</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>45</td>
<td>StepSize</td>
<td>Series increment</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>46</td>
<td>SUnits</td>
<td>Series units</td>
<td>UINT8</td>
<td>11</td>
</tr>
<tr>
<td>47</td>
<td>PreTrigg</td>
<td>Maximum pre-trigger samples</td>
<td>UInt16</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Timing related information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>UpdateT</td>
<td>TransducerChannel update time (t_u)</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>21</td>
<td>WSetupT</td>
<td>TransducerChannel write setup time (t_ws)</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>22</td>
<td>RSetupT</td>
<td>TransducerChannel read setup time (t_r)</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>23</td>
<td>SPeriod</td>
<td>TransducerChannel sampling period (t_sp)</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>24</td>
<td>WarmUpT</td>
<td>TransducerChannel warm-up time</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>25</td>
<td>RDelayT</td>
<td>TransducerChannel read delay time (t_d)</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>26</td>
<td>TestTime</td>
<td>TransducerChannel self-test time requirement</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Time of the sample information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>TimeSrc</td>
<td>Source for the time of sample</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>InPropDI</td>
<td>Incoming propagation delay through the data</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Attributes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Sampling</td>
<td>Sampling attribute</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Field</td>
<td>Field Name</td>
<td>Description</td>
<td>Type</td>
<td># octets</td>
</tr>
<tr>
<td>-------</td>
<td>------------</td>
<td>-------------</td>
<td>--------</td>
<td>----------</td>
</tr>
<tr>
<td>48</td>
<td>SampMod</td>
<td>Sampling mode capability</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>49</td>
<td>SDefault</td>
<td>Default sampling mode</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>DataXmit</td>
<td>Data transmission attribute</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>Buffered</td>
<td>Buffered attribute</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>EndOfSet</td>
<td>End-of-data-set operation attribute</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>35</td>
<td>EdgeRpt</td>
<td>Edge-to-report attribute</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>36</td>
<td>ActHalt</td>
<td>Actuator-halt attribute</td>
<td>UInt8</td>
<td>1</td>
</tr>
</tbody>
</table>

### Sensitivity

<table>
<thead>
<tr>
<th>Field</th>
<th>Field Name</th>
<th>Description</th>
<th>Type</th>
<th># octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>Directon</td>
<td>Sensitivity direction</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>38</td>
<td>DAngles</td>
<td>Direction angles</td>
<td>Two Float32</td>
<td>8</td>
</tr>
</tbody>
</table>

### Options

<table>
<thead>
<tr>
<th>Field</th>
<th>Field Name</th>
<th>Description</th>
<th>Type</th>
<th># octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>ESOption</td>
<td>Event sensor options</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>61-127</td>
<td>— Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>128-255</td>
<td>— Open to manufacturers</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>—</td>
<td>— checksum</td>
<td>—</td>
<td>UInt16</td>
<td>2</td>
</tr>
</tbody>
</table>

#### 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.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Sensor</td>
</tr>
<tr>
<td>1</td>
<td>Actuator</td>
</tr>
<tr>
<td>2</td>
<td>Event sensor</td>
</tr>
<tr>
<td>3-255</td>
<td>Reserved for future expansion</td>
</tr>
</tbody>
</table>
Figure 14—TransducerChannel TEDS
8.5.2.6 Physical units

Field Type: 12
Field Name: PhyUnits

This is a required field 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 4.11 or 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 eight bit integer (UInt8, 1 octet).

This is a required field 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 eight 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 eight 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 eight bit integer (UInt8, 1 octet).

This field is described in 4.11.

**8.5.2.11 The exponent for Kilograms**

Field Type: 54
Field Name: Kilograms
Data type: Unsigned eight 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 eight 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 eight 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 eight 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 eight bit integer (UInt8, 1 octet).
This field is described in 4.11.
8.5.2.16 The exponent for Candelas

Field Type: 59
Field Name: Candelas
Data type: Unsigned eight 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 substitute 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 is a required field 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 lies 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 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, that 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 is a required field 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, that 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.

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

The “Combined Standard Uncertainty” defined in [A8]: NIST Technical Note 1297, prepared by B. N. Taylor and C. E. Kuyatt, entitled “Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results,” 1994 edition, 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 is a required field 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>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No self-test function needed or provided</td>
</tr>
<tr>
<td>1</td>
<td>Self-test function provided</td>
</tr>
<tr>
<td>2-255</td>
<td>Reserved for future expansion</td>
</tr>
</tbody>
</table>

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 multi-range capability exists within this TransducerChannel.

The content of this field identifies whether or not 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 multi-range capability exists. If a multi-range 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 a required field. 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 sub-fields 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 a required field. 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.

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

There are two differences between the integer forms (enumeration zero and five) and fractional forms (enumeration three and six):

The radix point (that divides integer from fractional bits) is to the right of the least significant bit for N-octet integer or long integer. It is immediately to the right of the most significant bit for N-octet fraction or long fraction. Justification of the significant bits differs, as explained in the Model significant bits field (8.5.2.26).
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 one 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 a required field. 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 a required field. If this field is omitted the NCAP shall report a fatal TEDS error.

When the Data model is 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 comes from a 12-bit ADC, then:

\[
\begin{align*}
\text{Data model} &= 0 \quad (\text{N-octet integer}) \\
\text{Data model length} &= 2 \quad (\text{the number of octets needed to hold 12 bits}) \\
\text{Model significant bits} &= 12
\end{align*}
\]

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

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

When the Data model is N-octet fraction or long fraction, the data bits shall be left justified within the octet stream. Model significant bits specifies the number of bits that are significant.
When the Data model is single or double precision real (enumerations one or two), 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, Model significant bits is fixed and the field 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 is an optional field. If this field is omitted the Maximum data repetitions and the Maximum pre-trigger samples shall be assumed to be zero.

The sub-fields 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)

This is an optional field. 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, that 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, a mass spectrum or 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, etc.

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 non-zero 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 origin is 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 non-zero 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 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 non-zero 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 non-fatal 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, that 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, etc.

8.5.2.33 TransducerChannel update time

Field Type: 20

Field Name: UpdateT

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

This is a required field 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 (tu), 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 is a required field for all actuators 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 \( t_{\text{ws}} \), 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 is a required field 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 \( t_{\text{rs}} \), 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 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 is a required field 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 \( t_{\text{sp}} \) 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, etc., 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 set-up 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 a required field. If this field is omitted the NCAP shall report a non-fatal 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.

**8.5.2.38** TransducerChannel read delay time

Field Type: 25

Field Name: RDelayT

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

This is a required field 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 is a required field 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 non-fatal 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 is an optional field 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 require a ControlGroup to be defined in the Meta-TEDS to identify the time interval sensor or TimeInstance sensor to be used with this TransducerChannel.

Table 53—Enumeration of sample time sources

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NoHelp</td>
<td>No facilities in the transducer module are available to support determination of the time that a sample was latched.</td>
</tr>
<tr>
<td>1</td>
<td>Incoming</td>
<td>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).</td>
</tr>
<tr>
<td>2</td>
<td>Outgoing</td>
<td>The time interval between the latching of the sample and when the data is 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 in the immediate operation sampling mode (see 5.10.1.7).</td>
</tr>
<tr>
<td>3</td>
<td>MInterval</td>
<td>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.</td>
</tr>
<tr>
<td>4</td>
<td>SInterval</td>
<td>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.</td>
</tr>
<tr>
<td>5</td>
<td>ToDSense</td>
<td>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.</td>
</tr>
<tr>
<td>6 – 127</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>128 - 255</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

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 non-fatal 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 upon 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)

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 non-fatal 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 upon 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 sample being taken or applied.

The expression of the uncertainty shall be in accordance with the “Combined Standard Uncertainty” defined in [A8]: NIST Technical Note 1297, prepared by B. N. Taylor and C. E. Kuyatt, entitled “Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results,” 1994 edition, Appendix C, Section 2.2.

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 sub-fields 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.

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

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

### 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

<table>
<thead>
<tr>
<th>Bit</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (lsb)</td>
<td>Set to a one if the Trigger initiated mode (see 5.10.1.1) is the default sampling mode for this TransducerChannel.</td>
</tr>
<tr>
<td>1</td>
<td>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.</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>Set to a one if the Continuous Sampling mode (see 5.10.1.6) is the default sampling mode for this TransducerChannel.</td>
</tr>
<tr>
<td>4</td>
<td>Set to a one if the Immediate operation sampling mode (see 5.10.1.7) is the default sampling mode for this TransducerChannel.</td>
</tr>
<tr>
<td>5</td>
<td>reserved</td>
</tr>
<tr>
<td>6</td>
<td>reserved</td>
</tr>
<tr>
<td>7 (msb)</td>
<td>reserved</td>
</tr>
</tbody>
</table>

8.5.2.47 Buffered attribute

Field Type: 32
Field Name: Buffered
Data type: unsigned 8-bit integer (UInt8, 1 octet)
This is an optional field. 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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TransducerChannel has no more than one buffer available</td>
</tr>
<tr>
<td>1</td>
<td>TransducerChannel has multiple buffers available and may only be operated in the buffered mode.</td>
</tr>
<tr>
<td>2</td>
<td>TransducerChannel has multiple buffers available and may be operated either buffered or unbuffered. The unbuffered mode of operation is the default.</td>
</tr>
<tr>
<td>3</td>
<td>TransducerChannel has multiple buffers available and may be operated either buffered or unbuffered. The buffered mode of operation is the default.</td>
</tr>
<tr>
<td>4 - 255</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

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 non-fatal 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>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not applicable</td>
</tr>
<tr>
<td>1</td>
<td>This TransducerChannel holds the last value in the data-set until another trigger is received as described in 5.10.4.1.</td>
</tr>
<tr>
<td>2</td>
<td>This TransducerChannel recirculates through the last data-set until another trigger is received as described in 5.10.4.2.</td>
</tr>
<tr>
<td>3</td>
<td>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.</td>
</tr>
<tr>
<td>4</td>
<td>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.</td>
</tr>
<tr>
<td>5 - 255</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### 8.5.2.49 Data transmission attribute

Field Type: 34

Field Name: DataXmit

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

This 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>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>This TransducerChannel is only capable of being operated in the only when commanded mode (see 5.10.2.1).</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>This TransducerChannel is capable of being operated in the Streaming at a fixed interval (see 5.10.2.3) or only when commanded modes.</td>
</tr>
<tr>
<td>4</td>
<td>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.</td>
</tr>
<tr>
<td>5 - 255</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### 8.5.2.50 Edge-to-report attribute

Field Type: 35
Field Name: EdgeRpt
Data type: unsigned 8-bit integer (UInt8, 1 octet)

This 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not applicable</td>
</tr>
<tr>
<td>1</td>
<td>This event sensor only reports rising edges</td>
</tr>
<tr>
<td>2</td>
<td>This event sensor only reports falling edges</td>
</tr>
<tr>
<td>3</td>
<td>This event sensor reports on both edges</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>This event sensor is capable of reporting both edges but the default is to report rising edges.</td>
</tr>
<tr>
<td>6</td>
<td>This event sensor is capable of reporting both edges but the default is to report falling edges.</td>
</tr>
<tr>
<td>7</td>
<td>This event sensor is capable of reporting both edges and the default is to report both edges.</td>
</tr>
<tr>
<td>8 - 255</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

8.5.2.51 Actuator-halt attribute

Field Type: 36
Field Name: ActHalt
Data type: unsigned 8-bit integer (UInt8, 1 octet)

This 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 Error! Reference source not found.). Table 60 lists the allowable values for this attribute.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not applicable</td>
</tr>
<tr>
<td>1</td>
<td>Halt Immediate</td>
</tr>
<tr>
<td>2</td>
<td>Halt at the end of the data-set</td>
</tr>
<tr>
<td>3</td>
<td>Ramp to a predefined state</td>
</tr>
<tr>
<td>4 - 255</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
8.5.2.52 Sensitivity direction

Field Type: 37

Field Name: Direction

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

This is an optional field.

For sensors that measure physical phenomena with 3 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.

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 2 of the 3 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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>1</td>
<td>+X</td>
</tr>
<tr>
<td>2</td>
<td>-X</td>
</tr>
<tr>
<td>3</td>
<td>+Y</td>
</tr>
<tr>
<td>4</td>
<td>-Y</td>
</tr>
<tr>
<td>5</td>
<td>+Z</td>
</tr>
<tr>
<td>6</td>
<td>-Z</td>
</tr>
<tr>
<td>7-255</td>
<td>reserved for future expansion</td>
</tr>
</tbody>
</table>

8.5.2.53 Direction angles

Field Type: 38

Field Name: DAngles

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

This is an optional field.

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 $\rho$ in right hand cylindrical spatial coordinates expressed in radians. The second number represents $\phi$ 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.

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>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not applicable</td>
</tr>
<tr>
<td>1</td>
<td>Pattern/threshold/hysteresis not changeable</td>
</tr>
<tr>
<td>2</td>
<td>Changeable and inconsistencies detected</td>
</tr>
<tr>
<td>3</td>
<td>Changeable and inconsistencies not detected</td>
</tr>
<tr>
<td>4 - 255</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

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 is an optional field.

This field is provided to allow a text-based extension to the physical units. It shall not be used as a substitute 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 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 TEDS for 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 is the number used to represent the TransducerChannel data in the users system.

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

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, that 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, meaning 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.

\[ y = mx + b = b + mx \]

Equation 5

There are two methods 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
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.

\[
\sum_{j=0}^{D_1} \sum_{k=0}^{D_2} \sum_{p=0}^{D_3} C_{ijk} \left[ X_1 - H_1 \right] \left[ X_2 - H_2 \right]^j \left[ X_3 - H_3 \right]^p
\]

Equation 6

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

\[
y = (A_0 + A_1 x_1 + A_2 x_1^2 + \ldots + A_n x_1^n)(B_0 + B_1 x_2 + B_2 x_2^2 + \ldots + B_m x_2^m)(\ldots)(N_0 + N_1 x_n + N_2 x_n^2 + \ldots + N_p x_n^p)
\]

Equation 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.

\[
y = A_0 B_0 + A_1 B_0 x_1 + A_2 B_0 x_1^2 + A_3 B_0 x_1^3 + \ldots + A_n B_0 x_1^n + A_0 B_1 x_2 + A_1 B_1 x_2^2 + A_2 B_1 x_2^3 + \ldots + A_m B_1 x_2^m
\]

Equation 8

\[
C_0 + C_1 x_1 + C_2 x_1^2 + C_3 x_1^3 + \ldots + C_n x_1^n + C_{n+1} x_1 x_2 + C_{n+2} x_1^2 x_2 + \ldots + C_m x_1^m x_2^m.
\]

Equation 9

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

Equation 10 and Equation 11 are examples of this function in two of its simplest and very familiar forms.
Equation 10 represents the form of this equation that is used for a table look up correction. Equation 11
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.

\[ y = C_0 \]
Equation 10
\[ y = (C_0 + C_1 x) = b + mx \]

Equation 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 \( C_{ij} \).

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

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:

\[ C_{0,0} + C_{0,1} (X_2 - H_2) + C_{1,0} (X_1 - H_1) + C_{1,1} (X_1 - H_1) (X_2 - H_2) \]

Equation 12

As seen in Equation 12 there is a different \( H_i \) for each partition of \( X_i \). That is, \( H_i \) 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 \( X_2 \) which segment the measurement falls into and choose the coefficients and offsets accordingly.

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 TEDS entries 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 is read from the TIM.
- Correction shall be invoked on an actuator after new NCAP-side data is provided, and before the corrected transducer-side data is 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 is 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 is 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 is simply "counts").

The order and availability of the inputs to the correction process must be taken into consideration. The process takes the data that is available when it is required and uses it. If one parameter has been updated and another not updated, this can cause variation in the results. This 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.
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.

\[ SI = \text{Slope} \times \text{Value} + \text{Intercept} \]

\text{Equation 13}

where

- \text{Slope} is given in the SI units conversion slope field (see 8.6.3.5) in this TEDS.
- \text{Intercept} is given in the SI units conversion intercept field (see 8.6.3.6) in this TEDS.
- \text{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.

\[ \text{Value} = \frac{(SI - \text{Intercept})}{\text{Slope}} \]

\text{Equation 14}

where

- \text{Slope} is given in the SI units conversion slope field (see 8.6.3.5) in this TEDS.
- \text{Intercept} is given in the SI units conversion intercept field (see 8.6.3.6) in this TEDS.
- \text{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 shows the data structure that shall be used for Calibration TEDS. Subsequent subclauses explain each data field in the structure.

8.6.3.1 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.
8.6.3.2 Last calibration date-time

Field Type: 10
Field Name: LstCalDt
Data type: Time value (TimeInstance, 8 octets)
This is an optional field.
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 is an optional field.
The calibration interval is the length of time, in seconds, that this TransducerChannel can operate without needing another calibration and produce data that is within the Operational uncertainty specified in 8.6.3.9.
<table>
<thead>
<tr>
<th>Field Type</th>
<th>Field Name</th>
<th>Description</th>
<th>Type</th>
<th>Require d/Option al</th>
<th>Data Length in octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>TEDS length</td>
<td></td>
<td>UInt32</td>
<td>—</td>
<td>4</td>
</tr>
<tr>
<td>0-2</td>
<td>Reserved</td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>TEDSID</td>
<td>TEDS identifier</td>
<td>UInt8</td>
<td>R</td>
<td>4</td>
</tr>
<tr>
<td>4-9</td>
<td>Reserved</td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>Calibration date related information</td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>10</td>
<td>LstCalDt</td>
<td>Last calibration date-time</td>
<td>TimeInstance</td>
<td>O</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>CalInrvl</td>
<td>Calibration interval</td>
<td>TimeDuration</td>
<td>O</td>
<td>8</td>
</tr>
<tr>
<td>Units information</td>
<td></td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>12</td>
<td>SIConvrt</td>
<td>SI units conversion constants</td>
<td>—</td>
<td>O</td>
<td>—</td>
</tr>
<tr>
<td>30</td>
<td>SISlope</td>
<td>SI units conversion slope</td>
<td>Float32</td>
<td>O</td>
<td>4</td>
</tr>
<tr>
<td>31</td>
<td>Intrept</td>
<td>SI units conversion intercept</td>
<td>Float32</td>
<td>O</td>
<td>4</td>
</tr>
<tr>
<td>Operational limits and uncertainty</td>
<td></td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>13</td>
<td>LowLimit</td>
<td>Operational lower range limit</td>
<td>Float32</td>
<td>O</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td>HiLimit</td>
<td>Operational upper range limit</td>
<td>Float32</td>
<td>O</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>OError</td>
<td>Operational uncertainty</td>
<td>Float32</td>
<td>O</td>
<td>4</td>
</tr>
<tr>
<td>Mathematical conversion to be performed on the data before or after correction</td>
<td></td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>16</td>
<td>OConvert</td>
<td>Post conversion operation</td>
<td>UInt8</td>
<td>O</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>IConvert</td>
<td>Pre-conversion operation</td>
<td>UInt8</td>
<td>O</td>
<td>1</td>
</tr>
<tr>
<td>TLV tuple 20 is used when the linear method of conversion is used.</td>
<td></td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>20</td>
<td>LinOnly</td>
<td>This field is used when only a linear single section conversion is required.</td>
<td>—</td>
<td>O</td>
<td>—</td>
</tr>
<tr>
<td>TLV tuple 21 provides a description of one TransducerChannel that is involved in the correction specified in this TEDS.</td>
<td></td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>21</td>
<td>XdcrBlk</td>
<td>TransducerChannel description</td>
<td>—</td>
<td>O</td>
<td>—</td>
</tr>
<tr>
<td>40</td>
<td>Element</td>
<td>Element Number</td>
<td>UInt16</td>
<td>O</td>
<td>1</td>
</tr>
<tr>
<td>41</td>
<td>ChanNum</td>
<td>Correction input</td>
<td>UInt16</td>
<td>O</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>ChanKey</td>
<td>Correction input</td>
<td>UInt8</td>
<td>O</td>
<td>1</td>
</tr>
<tr>
<td>43</td>
<td>Degree</td>
<td>TransducerChannel degree</td>
<td>UInt8</td>
<td>O</td>
<td>1</td>
</tr>
<tr>
<td>44</td>
<td>STable</td>
<td>Segment boundary values table</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>45</td>
<td>OTable</td>
<td>Segment offset values table</td>
<td>Float32Array</td>
<td>O</td>
<td>Note 1</td>
</tr>
<tr>
<td>46</td>
<td>LoBndry</td>
<td>Array of Low Boundary Limits</td>
<td>Float32Array</td>
<td>O</td>
<td>Note 1</td>
</tr>
<tr>
<td>47</td>
<td>HiBndry</td>
<td>High Boundary Limit</td>
<td>Float32</td>
<td>O</td>
<td>4</td>
</tr>
<tr>
<td>TLV tuple 22 provides a set of coefficients for one segment of the correction specified in this TEDS.</td>
<td></td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>22</td>
<td>CoefBlk</td>
<td>Multinomial coefficient</td>
<td>—</td>
<td>O</td>
<td>—</td>
</tr>
<tr>
<td>50</td>
<td>CellNum</td>
<td>Cell number of the segment to which this set of coefficients apply</td>
<td>UInt8</td>
<td>O</td>
<td>1</td>
</tr>
<tr>
<td>51</td>
<td>CoefSet</td>
<td>The set of coefficients that applies to this cell.</td>
<td>Float32Array</td>
<td>O</td>
<td>Note 2</td>
</tr>
<tr>
<td>18-19</td>
<td>Reserved</td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>23-29</td>
<td>Reserved</td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>48-49</td>
<td>Reserved</td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>52-127</td>
<td>Reserved</td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Field Type</td>
<td>Field Name</td>
<td>Description</td>
<td>Type</td>
<td>Required</td>
<td>Data Length in octets</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>----------------------------------</td>
<td>-----------</td>
<td>----------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>128-255</td>
<td>—</td>
<td>Open to manufacturers</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Checksum</td>
<td></td>
<td>UInt16</td>
<td>—</td>
<td>2</td>
</tr>
</tbody>
</table>

Note 1: Number of segments multiplied by 4

Note 2: \(((\text{Degree of element 1})+1)^2*(\text{Degree of element 2})+1)^2*(...)^2*(\text{Degree of element n})+1)^4\n
8.6.3.4 SI units conversion constants

Field Type: 12

Field Name: SIConvrt

This is a required field. If this field is omitted the NCAP shall report a non-fatal 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 sub-fields:
- 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 comparing the values given in this TEDS to 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 is a required field. If this field is omitted the NCAP shall report a non-fatal 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.
 TEDS ID
 Type = 03

 ListCalDt
 Type = 10

 CallInrvi
 Type = 11

 SICovntr
 Type = 12

 LowLimit
 Type = 13

 HiLimit
 Type = 14

 OError
 Type = 15

 OConvert
 Type = 16

 IConvert
 Type = 17

 ChanNum
 Type = 41

 ChanKey
 Type = 42

 CoefSet
 Type = 51

 Element
 Type = 40

 ChanNum
 Type = 41

 ChanKey
 Type = 42

 XdcrBlk+
 Type = 21

 Degree
 Type = 43

 LoBndy
 Type = 46

 HiBndy
 Type = 47

 STable
 Type = 44

 OTable
 Type = 45

 CellNum
 Type = 50

 CoefSet
 Type = 51
8.6.3.6  SI units conversion intercept

Field Type: 31

Field Name: Intrcpt

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

This is a required field. If this field is omitted the NCAP shall report a non-fatal 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.

8.6.3.7  Operational lower range limit

Field Type: 13

Field Name: LowLimit

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

This is an optional field. 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.

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 lies 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, that 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 is an optional field. 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 lies 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, that 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 is an optional field. If this field is not provided the NCAP shall use the uncertainty under worst-case conditions specified in the TransducerChannel TEDS.

This field uses the “Combined Standard Uncertainty” defined in [A8]: NIST Technical Note 1297, prepared by B. N. Taylor and C. E. Kuyatt, entitled “Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results,” 1994 edition, Appendix C, Section 2.2. 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 is an optional field. 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 is an optional field. 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>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No pre or post conversion operation is required.</td>
</tr>
<tr>
<td>1</td>
<td>Inversion: Apply 1/x</td>
</tr>
<tr>
<td>2</td>
<td>Log base 10: Apply log10(x)</td>
</tr>
<tr>
<td>3</td>
<td>Exponent Base 10: Apply 10x</td>
</tr>
<tr>
<td>4</td>
<td>Natural Log : Apply ln (x)</td>
</tr>
<tr>
<td>5</td>
<td>Exponent Base e: Apply ex</td>
</tr>
<tr>
<td>6-255</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### 8.6.3.12 Linear conversion only

Field Type: 20

Field Name: LinOnly

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 sub-fields shall be omitted. It consists of the following sub-fields:

- **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 sub-fields:

- 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 TEDS applies, 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 sixteen-bit 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.

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 below.

### 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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The correction input value shall be taken from the transducer-side of the correction process.</td>
</tr>
<tr>
<td>1</td>
<td>The correction input value shall be taken from the NCAP-side of the correction process.</td>
</tr>
<tr>
<td>2-255</td>
<td>Invalid</td>
</tr>
</tbody>
</table>

For a TransducerChannel, the following choices are available depending upon whether or not both corrected and uncorrected data are available:

- If only uncorrected data is available the value for the key shall be zero.
- If only corrected data is 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 upon 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 TEDS of TransducerChannel 3.

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 sub-groups:

- Array of Low Boundary Limits
- High Boundary Limit.

### 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 is 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 is 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

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 Oₖ (thus Hₖ) 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 is 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 sixteen-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 Xₖ (the domain of values from the highest numbered element). Upon reaching the last segment for domain Xₖ, the next cell covers the lowest segment of domain Xₖ again, and the next higher segment of domain Xₖ+1. For example,
for a two TransducerChannel correction with 2 segments on the first correction input (A1, A2) and 3
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

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 C0,
followed by a slope, m or C1. 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 fast
subscript identifying the entry with the highest element number.

Each entry in the table is a coefficient C_{i_0, \ldots, i_p}, used in the multinomial:

\[
\sum_{i=0}^{D(1)} \sum_{j=0}^{D(2)} \cdots \sum_{p=0}^{D(n)} C_{i_0, \ldots, i_p} [X_1 - H_1]^i [X_2 - H_2]^j \cdots [X_n - H_n]^p
\]

Equation 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.

<table>
<thead>
<tr>
<th>C_{0,0,\ldots,0}</th>
<th>C_{0,0,\ldots,1}</th>
<th>\ldots</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_{0,0,\ldots,Dn}</td>
<td>C_{0,0,\ldots,Dn+1}</td>
<td>\ldots</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>C_{D1,D2,\ldots,D(n-1),0}</td>
<td>C_{D1,D2,\ldots,D(n-1),1}</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

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 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 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>
<thead>
<tr>
<th>Field Type</th>
<th>Field Name</th>
<th>Description</th>
<th>Type</th>
<th># octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>TEDS length</td>
<td>TEDS length</td>
<td>UInt32</td>
<td>4</td>
</tr>
<tr>
<td>0-2</td>
<td>—</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>TEDSID</td>
<td>TEDS identifier</td>
<td>UInt8</td>
<td>4</td>
</tr>
<tr>
<td>4-9</td>
<td>—</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>10</td>
<td>RefFreq</td>
<td>Reference frequency</td>
<td>Float3</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>RefAmp</td>
<td>Test amplitude</td>
<td>Float3</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>RefPhase</td>
<td>Phase at the reference frequency</td>
<td>Float3</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>Points</td>
<td>Points in the table</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>14-127</td>
<td>—</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>128-255</td>
<td>—</td>
<td>Open to manufacturers</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>—</td>
<td>—</td>
<td>Checksum</td>
<td>UInt16</td>
<td>2</td>
</tr>
</tbody>
</table>

The following fields comprise a structure that defines one data point. The structure is repeated n times, once for each data point.
Figure 18—Frequency Response TEDS structure

8.7.2.1 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.

8.7.2.2 Reference frequency

Field Type: 10
Field Name: RefFreq
Data type: single precision real (Float32, 4 octets)
This is a required field. 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 is a required field. 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 is a required field. 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
Field Name: Points
Data type: Three single precision real (Float32, 4 octets) values (12 octets)
This is a required field. 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 sub-fields:
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>
<thead>
<tr>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field type</td>
<td>Always 13 for this field</td>
</tr>
<tr>
<td>Length</td>
<td>Always 12 because there are three floating point numbers in the value field so 12 octets are required</td>
</tr>
<tr>
<td>Frequency</td>
<td>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.</td>
</tr>
<tr>
<td>Amplitude</td>
<td>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.</td>
</tr>
<tr>
<td>Phase</td>
<td>This field identifies the phase shift of the TransducerChannel output at the Frequency. The units for this parameter shall be radians.</td>
</tr>
</tbody>
</table>

8.8 Transfer Function TEDS
This 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 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:

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.

\[
H(f) = \frac{H_1(f_{ref})}{H_1(f_{ref})} \frac{H_2(f_{ref})}{H_2(f_{ref})} \ldots \frac{H_N(f_{ref})}{H_N(f_{ref})}
\]

Equation 16

Equation 16 forces \( |H(f_{ref})| = 1 \), but maintains the phase at the reference frequency. The form of each element \( H_i(f) \) shall be as described in the subclauses of 8.8.3.

Alternatively, \( H(f) \) is described as a rational function, normally referred to as the z-transform, as shown in Equation 17.

\[
H(z) = \frac{A_0 + A_1 z^{-1} + A_2 z^{-2} + \ldots + A_n z^{-n}}{1 + B_0 + B_1 z^{-1} + B_2 z^{-2} \ldots + B_m z^{-m}}
\]

Equation 17

where
\[
\begin{align*}
    z^{-1} &= e^{-j\omega T} \\
    \omega &= 2\pi f \\
    j &= \sqrt{-1}
\end{align*}
\]
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.

### 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. Subsequent subclauses 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 upon the method chosen to define the transfer function.
### Table 68—Structure of the Transfer Function TEDS data block

<table>
<thead>
<tr>
<th>Field</th>
<th>Field Name</th>
<th>Description</th>
<th>Type</th>
<th># octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>TEDS length</td>
<td><a href="#">TEDS length</a></td>
<td>UInt32</td>
<td>4</td>
</tr>
<tr>
<td>0-2</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>TEDSID</td>
<td>TEDS Identification Header</td>
<td>UInt8</td>
<td>4</td>
</tr>
<tr>
<td>4-9</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

**Transfer function related information**

<table>
<thead>
<tr>
<th>Field</th>
<th>Field Name</th>
<th>Description</th>
<th>Type</th>
<th># octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>RefFreq</td>
<td>Reference frequency</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>OneZero</td>
<td>Single zero TF SZ</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>OnePole</td>
<td>Single pole TF SP</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>ZeroPole</td>
<td>Single zero with a dependant pole</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>14</td>
<td>PoleZero</td>
<td>Single pole with a dependent zero</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>15</td>
<td>ComplexZ</td>
<td>Complex zero</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>16</td>
<td>ComplexP</td>
<td>Complex pole</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>17</td>
<td>OneZZPol</td>
<td>Single zero at zero and a single pole</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>18</td>
<td>CRSlope</td>
<td>Constant Relative Slope</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>SampleT</td>
<td>Sample/Delay Time</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>DependP</td>
<td>Single pole dependent upon a zero TF SPm (x)</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>21</td>
<td>DependZ</td>
<td>Single zero dependent on a pole TF SZm (x)</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>22</td>
<td>ComplexZF</td>
<td>Complex zero frequency</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>23</td>
<td>ComplexZQ</td>
<td>Complex zero Quality Factor</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>24</td>
<td>ComplexPF</td>
<td>Complex pole frequency</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>25</td>
<td>ComplexPQ</td>
<td>Complex pole Quality Factor</td>
<td>Float32</td>
<td>4</td>
</tr>
<tr>
<td>26-29</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>30</td>
<td>NCoeff</td>
<td>Numerator coefficients (A0, A1, … An)</td>
<td>Array of Float32</td>
<td>n*4</td>
</tr>
<tr>
<td>31</td>
<td>DCoeff</td>
<td>Denominator coefficients (B0, B1, … Bm)</td>
<td>Array of Float32</td>
<td>m*4</td>
</tr>
<tr>
<td>32-127</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>128-255</td>
<td>Open for manufacturers use</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

**Checksum**

*NOTE—The number of coefficients, n or m, for fields 30 and 31 may be determined by dividing the tuple length by 4.*
8.8.3.1 TEDS Identifier
See the description of the TEDS Header 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.

8.8.3.2 Reference frequency
Field Type: 10
Field Name: RefFreq
Data type: single precision real (Float32, 4 octets)
This field is a required field. 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.

8.8.3.3 Single zero
Field Type: 11
Field Name: OneZero
Data type: single precision real (Float32, 4 octets)
This field is an optional field. 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 –3dB point at $F_{sz}$.

$$H(f, F_{sz}) = \left(1 + \frac{j\cdot f}{F_{sz}}\right)$$

Equation 18

8.8.3.4 Single pole
Field Type: 11
Field Name: OnePole
Data type: single precision real (Float32, 4 octets)
This field is an optional field. 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 –3dB point at $F_{sp}$.
Equation 19

\[ H(f, F_{sp}) = \frac{1}{(1 + \frac{j \cdot f}{F_{sp}})} \]

8.8.3.5 Single zero of a zero/pole pair

Field Type: 13
Field Name: ZeroPole
This field is an optional field. If this field is omitted, the transfer function does not contain a zero/pole pair.
This field is composed of two sub-fields. 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 is the same as the field single zero (see 8.8.3.3).
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 upon 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.

\[ H(f, x, v) = \frac{1}{(1 + \frac{j \cdot f}{x \cdot v})} \]

Equation 20

where \( v \) is the value "Fsz" 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 an optional field. If this field is omitted, the transfer function does not contain a pole/zero pair.

This field is composed of two sub-fields. 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

The same as the single pole 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 upon 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.

The value in this field is used as "x" in the transfer function shown in Equation 21.

\[
H(f, x, v) = (1 + \frac{f}{x \cdot v})
\]

Equation 21

where \(v\) is the value "Fsp" 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 an optional field. If this field is omitted, the transfer function does not contain a complex zero.

This field has two sub-fields, 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 \(F_{z0}\) in the transfer function given in Equation 22.
\[
H(f, F_\text{zres}, Qz) = (1 + \frac{j \cdot f}{Qz \cdot F_\text{zres}} + \frac{(j \cdot f)^2}{F_\text{zres}})
\]

\text{Equation 22}

where parameter \( Qz \) 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.

This is a “unitless” quantity.

8.8.3.14 Complex pole

Field Type: 16

Field Name: ComplexZ

This field is an optional field. If this field is omitted, the transfer function does not contain a complex pole.

This field has two sub-fields, 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 \( F_\text{pres} \) in the transfer function Equation 23.

\[
H(f, F_\text{zres}, Q p) = (1 + \frac{j \cdot f}{Q p \cdot F_\text{zres}} + \frac{(j \cdot f)^2}{F_\text{zres}})
\]
where the parameter \( Q_p \) has to be given in the associated element of the array found in the \textbf{Complex pole quality factor} field (see 8.8.3.16).

The units on this field shall be Hertz.

\textbf{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 \( Q_p \) 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 is a “unitless” quantity.

\textbf{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 an optional field. 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.

\[ H(f, F_{hp}) = \frac{j \cdot f}{F_{hp}} \left(1 + \frac{j \cdot f}{F_{hp}}\right) \]

\textbf{Equation 24}

Equation 24 directly describes a high pass filter of first order with the –3dB point at \( F_{hp} \). The units on this field shall be Hertz.

\textbf{8.8.3.18 Constant relative slope}

Field Type: 18

Field Name: CRSlope

\[ H(f, F_{hp}) = \frac{j \cdot f}{F_{hp}} \left(1 + \frac{j \cdot f}{F_{hp}}\right) \]

\textbf{Equation 24}
Data type: single precision real (Float32, 4 octets).

This field is an optional field. 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.

\[
H(f, a, F_{\text{Ref}}) = \left( \frac{f}{F_{\text{Ref}}} \right)^{\frac{a}{10}}
\]

\textbf{Equation 25}

\( a \) is the relative change per frequency decade.

\( F_{\text{Ref}} \) 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, that is found to be \(-0.78\) degrees.)

\subsection{8.8.3.19 Sample/delay time}

Field Type: 19

Field Name: SampleT

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

This field is an optional field. 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.

\subsection{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 an optional field. 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 alternate 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.

\textbf{NOTE}—The number of coefficients in this field may be obtained by dividing the “tuple length” by four.

\subsection{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 an optional field. If this field is omitted, the transfer function does not contain a \textit{”z”} transform. If this field is not present but field 30 is present the NCAP shall report a fatal TEDS error.

When the alternate form of the transfer function (the \textit{”z”} transform) is used, this field contains the list of coefficients required in the denominator of the equation. See the \textit{”z”}-transform discussion in 8.8.1.1 for more details.

\textbf{NOTE—}The number of coefficients in this field may be obtained by dividing the \textit{“tuple length”} by four.

\section{8.9 Text-based TEDS}

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 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 Error! Reference source not found.

\subsection{8.9.1 Access}

They are accessed using a \texttt{Query TEDS} command, a \texttt{Read TEDS segment} command, a \texttt{write TEDS segment} command, or an \texttt{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.

\subsection{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 \texttt{XMLText} contains the displayable information, encoded as XML. The manufacturer determines the number of languages implemented. Table 70 and Figure 20 show the contents of this TEDS.
Table 69—Structure of a Text-based TEDS data block

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Field Name</th>
<th>Description</th>
<th>Type</th>
<th># octets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TEDS length</td>
<td></td>
<td>UInt32</td>
<td>4</td>
</tr>
<tr>
<td>0-2</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TEDSID</td>
<td>TEDS Identification Header</td>
<td>UInt8</td>
<td>4</td>
</tr>
<tr>
<td>4-9</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following three fields comprise a language header. The header is repeated N times, once for each language supported.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th># octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>NumLang</td>
<td>The number N of different language blocks in this TEDS</td>
<td>UInt8</td>
</tr>
<tr>
<td>11</td>
<td>DirBlock</td>
<td>Language block description This block is repeated N times.</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>LangCode</td>
<td>Language code from ISO 639 (two letters in USASCII)</td>
<td>UInt8</td>
</tr>
<tr>
<td>21</td>
<td>Offset</td>
<td>Language offset</td>
<td>UInt32</td>
</tr>
<tr>
<td>22</td>
<td>Length</td>
<td>Language length = LL</td>
<td>UInt32</td>
</tr>
<tr>
<td>23</td>
<td>Compress</td>
<td>Enumeration identifying the compression technique used.</td>
<td>UInt8</td>
</tr>
<tr>
<td>12</td>
<td>SubSum</td>
<td>Non-displayable data checksum</td>
<td>UInt16</td>
</tr>
</tbody>
</table>

The following two fields comprise a structure containing text in one language. The structure is repeated N times, once for each language defined.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th># octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLText</td>
<td>XML based text block</td>
<td>text</td>
<td>LL - 2</td>
</tr>
<tr>
<td>XMLSum</td>
<td>Text block checksum</td>
<td>UInt16</td>
<td>2</td>
</tr>
<tr>
<td>13-19</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23-127</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128-255</td>
<td>Open to manufacturers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Checksum</td>
<td>UInt16</td>
<td>2</td>
</tr>
</tbody>
</table>
8.9.2.1 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.

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.

8.9.2.3 Language directory block

Field Type: 11
Field Name: DirBlock
This field is a required field. If this field is omitted the NCAP shall report a fatal TEDS error.
This field is made up of sub-fields 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 a required field. 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 entitled: “Code for the representation of names of languages.” 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

<table>
<thead>
<tr>
<th>ISO 639 language code</th>
<th>Meaning (informative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserved</td>
<td>—</td>
</tr>
<tr>
<td>aa</td>
<td>Afar</td>
</tr>
<tr>
<td>da</td>
<td>Danish</td>
</tr>
<tr>
<td>de</td>
<td>German</td>
</tr>
<tr>
<td>en</td>
<td>English</td>
</tr>
<tr>
<td>es</td>
<td>Spanish</td>
</tr>
<tr>
<td>eu</td>
<td>Basque</td>
</tr>
<tr>
<td>fi</td>
<td>Finnish</td>
</tr>
<tr>
<td>fr</td>
<td>French</td>
</tr>
<tr>
<td>ga</td>
<td>Irish</td>
</tr>
<tr>
<td>it</td>
<td>Italian</td>
</tr>
<tr>
<td>nl</td>
<td>Dutch</td>
</tr>
<tr>
<td>no</td>
<td>Norwegian</td>
</tr>
<tr>
<td>pl</td>
<td>Polish</td>
</tr>
<tr>
<td>pt</td>
<td>Portuguese</td>
</tr>
<tr>
<td>ru</td>
<td>Russian</td>
</tr>
<tr>
<td>sv</td>
<td>Swedish</td>
</tr>
<tr>
<td>vi</td>
<td>Vietnamese</td>
</tr>
<tr>
<td>zu</td>
<td>Zulu</td>
</tr>
</tbody>
</table>

#### 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 a required field. 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 a required field. 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 one 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 is an optional field. 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.

Table 71—Enumeration identifying compression algorithms

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>This enumeration means that no compression is used in this language block of this TEDS.</td>
</tr>
<tr>
<td>1</td>
<td>WinZip</td>
</tr>
<tr>
<td>2</td>
<td>GZip</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
<tr>
<td>4-127</td>
<td>Reserved</td>
</tr>
<tr>
<td>128-255</td>
<td>Open to manufacturers</td>
</tr>
</tbody>
</table>

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 a required field. 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 a required field. 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 a required field. If this field is omitted the NCAP shall report a fatal TEDS error.
8.10 End User Application Specific TEDS

This 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 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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Type</th>
<th># octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>TEDS length</td>
<td>UInt32</td>
<td>UInt32</td>
<td>4</td>
</tr>
<tr>
<td>0-2</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>TEDSID</td>
<td>TEDS Identification Header</td>
<td>UInt8</td>
<td>UInt8</td>
</tr>
<tr>
<td>4-9</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>10</td>
<td>EndUserData</td>
<td>Variable</td>
<td>Variable</td>
<td></td>
</tr>
<tr>
<td>—</td>
<td>Checksum</td>
<td>UInt16</td>
<td>—</td>
<td>2</td>
</tr>
</tbody>
</table>

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 a required field. 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 is a required TEDS for the TIM and 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 1451.1 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.

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>
<thead>
<tr>
<th>Field Type</th>
<th>Field Name</th>
<th>Description</th>
<th>Data Type</th>
<th># octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>——</td>
<td>Length</td>
<td>—</td>
<td>UInt32</td>
<td>4</td>
</tr>
<tr>
<td>0-2</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>TEDSID</td>
<td>TEDS Identification Header</td>
<td>UInt8</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Format</td>
<td>Format description of this TEDS</td>
<td>UInt8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>TCName</td>
<td>TIM or TransducerChannel Name</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>——</td>
<td>Checksum</td>
<td>—</td>
<td>UInt16</td>
<td>2</td>
</tr>
</tbody>
</table>

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 to accommodate 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 a required field. 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 a required field. 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>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>User defined</td>
</tr>
<tr>
<td>1</td>
<td>Text-based TEDS using the format defined in clause 8.9</td>
</tr>
<tr>
<td>2-255</td>
<td>Reserved for future expansion</td>
</tr>
</tbody>
</table>
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 clause 8.9.

NOTE—The use of multiple languages in this TEDS may cause problems with a user’s application.

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 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 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 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 IEEE p1451.0.

8.12.2.1 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.

8.12.2.2 Data block contents

The contents and structure of this field are controlled by the manufacturer.
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 information common to all channels. PHY TEDS octets are constant and read-only.

This TEDS is not described further in this standard. See the standard on the family member that 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 IEEE p1451.0.

8.13.2.1 **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.

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 IEEE p1451.0 Application Program Interface (API)**

Two “application programmer interfaces” (APIs) are defined in IEEE P1451.0. This clause defines the common aspects of the APIs. The “Transducer Services Interface” defined in 10 is an NCAP-only API used by measurement and control applications to access the IEEE p1451.0 system. 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, ie those with a single set of hardware and software without regard to distinguishing separate interfaces between IEEE p1451.0 functionality and IEEE p1451.X functionality, is optional as long as the messages at visible interfaces conform to the rest of the standard. The definition of these API 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 IEEE p1451.0 and IEEE 1451.X. 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 IEEE 1451.X and called by IEEE p1451.0 to initiate communication operations. Similarly, this API contains methods that would be implemented by IEEE p1451.0 that are called by IEEE 1451.X 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:

• 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.

This included PCs functioning as NCAPS to very low-end “PIC-like” 8-bit microprocessors.

Provide escape mechanisms where IEEE 1451.X can intercept a communication invocation and thereby bypass network operations.

Provide a pass-through mechanism where knowledgeable applications can send custom commands through IEEE p1451.0 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 p1451.0 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. 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.

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 p1451.0 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

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransducerServices</td>
<td>This is the public API that measurement and control applications use to interact with the IEEE p1451.0 system. 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.</td>
</tr>
<tr>
<td>ModuleCommunications</td>
<td>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.</td>
</tr>
<tr>
<td>Args</td>
<td>This package contains all the IEEE p1451.0 arguments. The ArgumentArray data structure is defined here.</td>
</tr>
<tr>
<td>Util</td>
<td>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 alternate Codecs.</td>
</tr>
</tbody>
</table>
9.2.2 IEEE p1451.0 OctetArray Payloads

To minimize the knowledge that IEEE 1451.X needs to be concerned with, all transmitted information from IEEE p1451.0’s perspective 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 p1451.0 message, IEEE 1451.X should consider the payload as “opaque.” See clause 6.2 & .3 for the location of the length information necessary for decoding of the data. This is handled inside the encoder/decoder class.

From IEEE p1451.0’s perspective, 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 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 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 p1451.0 IDL Data Structures and ArgumentArray usage

To facilitate the use of payloads in the IEEE p1451.0 and upper measurement application layers, IEEE p1451.0 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 structures 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 p1451.0 and application layers is the generic ArgumentArray, that 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, IEEE p1451.0 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 needs. See clauses 7 and 8 for details.

In rare cases, IEEE 1451.X may need to deviate from the IEEE p1451.0 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 p1451.0 interfaces are inside the IDL module “IEEE1451Dot0”.

9.3.1 IEEE1451Dot0::Args

IDL: module Args { };

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
Fundamental and special datatypes are contained within the Args module. The fundamental datatypes and datatypes 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

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 quantity.

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. There are five entities involved in a communication
transaction: the local IEEE p1451.0 layer, the local IEEE 1451.X layer, the remote IEEE 1451.X layer, the
remote IEEE p1451.0 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>
<thead>
<tr>
<th>Bits 15 through 13</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>The error code source information is encoded in this location as described in Table 77</td>
<td></td>
</tr>
</tbody>
</table>

| 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 three bit unsigned integer.

<table>
<thead>
<tr>
<th>Value</th>
<th>Source of the error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Error from the local IEEE p1451.0 layer</td>
</tr>
<tr>
<td>1</td>
<td>Error from the local IEEE 1451.X layer</td>
</tr>
<tr>
<td>2</td>
<td>Error from the remote IEEE 1451.X layer</td>
</tr>
<tr>
<td>4</td>
<td>Error from the remote application layer</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Open to manufacturers</td>
</tr>
</tbody>
</table>

Error code values are specified in Table 78. In this case bits 12 through 0 are evaluated as a 13 bit unsigned
integer.
Table 78—Error code enumerations

<table>
<thead>
<tr>
<th>enumerations</th>
<th>Error Code Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO_ERROR</td>
<td>No error, operation successful</td>
</tr>
<tr>
<td>1</td>
<td>INVALID_COMMID</td>
<td>Invalid &quot;commId&quot;</td>
</tr>
<tr>
<td>2</td>
<td>UNKNOWN_DESTID</td>
<td>Unknown &quot;destId&quot;</td>
</tr>
<tr>
<td>3</td>
<td>TIMEOUT</td>
<td>Operation timeout</td>
</tr>
<tr>
<td>4</td>
<td>NETWORK.FAILURE</td>
<td>Destination unreachable network failure</td>
</tr>
<tr>
<td>5</td>
<td>NETWORK_CORRUPTION</td>
<td>Corrupt Communication network failure</td>
</tr>
<tr>
<td>6</td>
<td>MEMORY</td>
<td>Local out-of-memory error</td>
</tr>
<tr>
<td>7</td>
<td>QOS FAILURE</td>
<td>Network Quality-of-Service violation</td>
</tr>
<tr>
<td>8</td>
<td>MCAST_NOT_SUPPORTED</td>
<td>Multicast not supported or operation invalid for multicast</td>
</tr>
<tr>
<td>9</td>
<td>UNKNOWNGROUPID</td>
<td>Unknown “groupId”</td>
</tr>
<tr>
<td>10</td>
<td>UNKNOWN_MODULEID</td>
<td>Unknown “moduleId”</td>
</tr>
<tr>
<td>11</td>
<td>UNKNOWN_MSGID</td>
<td>Unknown “msgId”</td>
</tr>
<tr>
<td>12</td>
<td>NOTGROUP_MEMBER</td>
<td>destId not in the group</td>
</tr>
<tr>
<td>13</td>
<td>ILLEGAL_MODE</td>
<td>The mode parameter is not valid</td>
</tr>
<tr>
<td>14</td>
<td>LOCKED_RESOURCE</td>
<td>The resource being accessed is locked</td>
</tr>
<tr>
<td>15</td>
<td>FATAL_TEDS_ERROR</td>
<td>There is an error in the TEDS that makes the device unusable</td>
</tr>
<tr>
<td>16</td>
<td>NON-FATAL_TEDS_ERROR</td>
<td>The value in a field in the TEDS is unusable but the device will still function.</td>
</tr>
<tr>
<td>17</td>
<td>CLOSE_ON_LOCKEDRESOURCE</td>
<td>A warning error code returned to signal that a close on a locked resource was performed.</td>
</tr>
<tr>
<td>18</td>
<td>LOCK_BROKEN</td>
<td>If a non-blocking read or write, or measurement stream is in progress, the callback will be invoked with this error code.</td>
</tr>
<tr>
<td>19</td>
<td>NETWORKRESOURCE_EXCEEDED</td>
<td>IEEE 1451.X has reached network resource limits</td>
</tr>
<tr>
<td>20</td>
<td>MEMORYRESOURCE_EXCEEDED</td>
<td>IEEE 1451.X has reached memory resource limits</td>
</tr>
<tr>
<td>21-4095</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>4096-8191</td>
<td>Open to manufacturers</td>
<td></td>
</tr>
</tbody>
</table>

9.3.1.3 IEEE1451Dot0::Args::QosParam

“Quality of Service” attributes are organized into a data structure for efficiency. The QoSParams structure contain the information listed in Table 79.

IDL:
struct QoSParam {
    Boolean service;
    TimeDuration period;
    UInt16 transmitSize;
    UInt16 replySize;
    TimeDuration accessLatency;
    TimeDuration transmitLatency;
}
### Table 79—QoSParams descriptions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>Boolean</td>
<td>A “true” indicates guaranteed QoS. A “false” indicates “best effort.”</td>
</tr>
<tr>
<td>period</td>
<td>TimeDuration</td>
<td>Indicates the period of communication. A zero value should be used for non-periodic conditions.</td>
</tr>
<tr>
<td>transmitSize</td>
<td>UInt32</td>
<td>This parameter specifies the number of octets transmitted for each periodic communication.</td>
</tr>
<tr>
<td>replySize</td>
<td>UInt32</td>
<td>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.</td>
</tr>
<tr>
<td>accessLatency</td>
<td>TimeDuration</td>
<td>This parameter specifies the extra time latency that IEEE p1451.0 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.</td>
</tr>
<tr>
<td>transmitLatency</td>
<td>TimeDuration</td>
<td>This parameter specifies the extra time latency that IEEE p1451.0 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.</td>
</tr>
</tbody>
</table>

### 9.3.1.4 IEEE1451Dot0::Args::TypeCode

Each valid type in the IEEE p1451.0 ArgumentArray has a unique typecode.

#### IDL:
```idl
eenum TypeCode {
    UNKNOWN_TC,
    // Simple types
    UInt8_TC, UINT16_TC, UINT32_TC,
    FLOAT32_TC, FLOAT64_TC, 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:
```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_ARRAY_TC:    OCTET_ARRAY_TC;
    case BOOLEAN_ARRAY_TC:  BOOLEAN_ARRAY_TC;
    case TIME_INSTANCE_ARRAY_TC:  TIME_INSTANCE_ARRAY_TC;
    case TIME_DURATION_ARRAY_TC:  TIME_DURATION_ARRAY_TC;
};
```
case OCTET_TC:               _Octet         valueOctet;
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
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 the methods associated with ArgumentArrays.

Table 80—ArgumentArray

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE1451Dot0::Args::ArgumentArray</td>
</tr>
<tr>
<td>UInt16 getByName( in _String name, out Argument reference);</td>
</tr>
<tr>
<td>UInt16 getByIndex( in UInt16 index, out Argument reference);</td>
</tr>
<tr>
<td>UInt16 putByName( in _String name, in Argument value);</td>
</tr>
<tr>
<td>UInt16 putByIndex( in UInt16 index, in Argument value);</td>
</tr>
<tr>
<td>UInt16 stringToIndex( in String name, out UInt16 index);</td>
</tr>
<tr>
<td>UInt16 getNames( out StringArray names);</td>
</tr>
<tr>
<td>UInt16 getindexes( out UInt16Array indexes);</td>
</tr>
<tr>
<td>UInt16 size()</td>
</tr>
</tbody>
</table>

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 clauses 7 and 8 for appropriate attribute names. A reference to the desired Argument is returned. Note 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);

This method provides a lookup-by-index feature. See clauses 7 and 8 for appropriate conversions between names and indices. A reference to the desired Argument is returned. Note 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 clauses 7 and 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 clauses 7 and 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.

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 IEEE 1451.X to provide custom Encoding and Decoding of ArgumentArrays to/from OctetArrays. If registered, it will be invoked by IEEE P1451.0 automatically.

Table 81 lists the interfaces in this grouping.
### Table 81—Codec

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE1451dot0::Util::Codec::encodeCommand()</td>
<td>Encodes a command into an OctetArray. Called on the initiating node before invocation of P2PComm::write() or NetComm::writeMsg() calls.</td>
</tr>
<tr>
<td>IEEE1451dot0::Util::Codec::decodeCommand()</td>
<td>Decodes an OctetArray into command arguments. Called on the receiving node after invocation of P2PComm::read() or NetComm::readMsg() calls.</td>
</tr>
<tr>
<td>IEEE1451dot0::Util::Codec::encodeResponse()</td>
<td>Encodes a response into an OctetArray.</td>
</tr>
<tr>
<td>IEEE1451dot0::Util::Codec::decodeResponse()</td>
<td>Decodes an OctetArray into a response.</td>
</tr>
<tr>
<td>argumentArray2OctetArray()</td>
<td>Converts an ArgumentArray to an OctetArray.</td>
</tr>
<tr>
<td>octetArray2ArgumentArray()</td>
<td>Converts an OctetArray to an ArgumentArray.</td>
</tr>
</tbody>
</table>

#### 9.3.2.2 IEEE1451Dot0::Util::Codec::encodeCommand

**IDL:**

```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 IEEE p1451X and called by IEEE P1451.0 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.

Parameters:
- `destId`: Desired channel identifier.
- `channelId`: Desired command class.
- `cmdClassId`: Desired command function code.
- `cmdFunctionId`: Command specific input arguments.
- `payload`: Encoded OctetArray.

Return result: Error code.

#### 9.3.2.3 IEEE1451Dot0::Util::Codec::decodeCommand

**IDL:**

```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 IEEE p1451X and called by IEEE P1451.0 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:
- `payload`: Encoded OctetArray.
- `channelId`: Channel identifier.
- `cmdClassId`: Command class.
- `cmdFunctionId`: Command function code.
- `inArgs`: Command specific input arguments.

Return result: Error code.
Parameters:

1. The “payload” is the OctetArray.
2. The [out] “channelId” parameter is the desired channel identifier.
3. The [out] “cmdClassId” is the desired command class.
4. The [out] “cmdFunctionId” is the desired command function code.
5. 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 IEEE p1451X and called by IEEE P1451.0 to encode a response into an OctetArray. See clauses 7 and 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:

1. The “successFlag” parameter is the desired success code.
2. The “outArgs” parameters are the command response specific output arguments.
3. 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 IEEE p1451X and called by IEEE P1451.0 to decode an OctetArray into response arguments. See clauses 7 and 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:

1. The “payload” is the OctetArray.
2. The [out] “successFlag” parameter is the command success flag.
3. 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(
  in  Args::ArgumentArray   inArgs,
  out Args::OctetArray      payload);
This method is provided by IEEE p1451X and called by IEEE P1451.0 to encode an ArgumentArray into an OctetArray. See clauses 7 and 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 IEEE p1451X and called by IEEE P1451.0 to decode an OctetArray into an ArgumentArray. See clauses 7 and 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 IEEE P1451.0 and optionally called by IEEE 1451.X to register alternate Encode/Decode implementations with the 1451.0 layer. Table 82 lists the methods available in this class.

Table 82—CodecManagement

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Args::UInt16 register( in Args::UInt8 moduleId, in Codec customCodec);</td>
</tr>
<tr>
<td>Args::UInt16 unregister( in Args::UInt8 moduleId);</td>
</tr>
</tbody>
</table>

9.3.2.9 IEEE1451Dot0::Util::CodecManagement::register

IDL:

```
Args::UInt16 register(
   in Args::UInt8 moduleId,
   in Codec customCodec);
```

This method is provided by IEEE p1451.0 and optionally called by IEEE 1451.X to register a custom Encoder/Decoder instance. Only a single instance may be registered for each module.

Parameters:
- The “moduleId” parameter is the desired IEEE 1451.X ID.
- 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 IEEE p1451.0 and optionally called by IEEE 1451.X to unregister a custom Encoder / Decoder instance. After this call, the default Encoder / Decoder will be used.

Parameters:
The “moduleId” 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 IEEE p1451.0.

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 IEEE p1451.0 and are called by the measurement application. If the application desires advanced optional features, it will need to implement the “AppCallback” interface, that IEEE p1451.0 will invoke.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimDiscovery</td>
<td>Methods for applications to discover available IEEE 1451.X communications modules, TIMs, and TransducerChannels are organized in this interface.</td>
</tr>
<tr>
<td>TransducerAccess</td>
<td>When an application desires to access sensor and actuator TransducerChannels, it will use methods on this interface.</td>
</tr>
<tr>
<td>TransducerManager</td>
<td>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.</td>
</tr>
<tr>
<td>TedsManager</td>
<td>Applications use methods on this interface to read and write TEDS. This class also manages the NCAP-side TEDS cache information.</td>
</tr>
<tr>
<td>CommManager</td>
<td>Handles access to the communication module on the local device.</td>
</tr>
<tr>
<td>AppCallback</td>
<td>Applications that need advanced features need to implement this interface. For example, this allows the application to configure measurement streams and IEEE p1451.0 will invoke appropriate callbacks in the application.</td>
</tr>
</tbody>
</table>

10.1 IEEE1451Dot0::TransducerServices::TimDiscovery

IDL: interface TimDiscovery { );

The TimDiscovery interface is provided by IEEE p1451.0 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE1451Dot0::TransducerServices::TimDiscovery::reportCommModule</td>
<td>This method reports the available communication module interfaces that have been registered with IEEE p1451.0. See the IEEE1451Dot0::ModuleCommunication::NetRegistration::registerModule() method, that the IEEE 1451.X layer will invoke when it is ready for operation. In that method, IEEE p1451.0 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, IEEE 1451.5 Clause 7), Multiple interfaces of the same technology (for example, IEEE 1451.2-RS232 on COM1 and COM2), or Multiple IEEE 1451.X interfaces of different technologies (for example, IEEE p1451.5 Clause 7, and IEEE 1451.3 multidrop).</td>
</tr>
<tr>
<td>IEEE1451Dot0::TransducerServices::TimDiscovery::reportTims</td>
<td>This returns the known TIM devices on this interface. See 11.6.2, the IEEE1451Dot0::ModuleCommunication::Registration::registerDestination method, that the IEEE 1451.X layer will invoke when registering new TIMs to the NCAP.</td>
</tr>
<tr>
<td>IEEE1451Dot0::TransducerServices::TimDiscovery::reportChannels</td>
<td>This returns the TransducerChannel list and names for this TIM. This information is retrieved from the cached TEDS.</td>
</tr>
</tbody>
</table>

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 IEEE p1451.0. See the IEEE1451Dot0::ModuleCommunication::NetRegistration::registerModule() method, that the IEEE 1451.X layer will invoke when it is ready for operation. In that method, IEEE p1451.0 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, IEEE 1451.5 Clause 7), Multiple interfaces of the same technology (for example, IEEE 1451.2-RS232 on COM1 and COM2), or Multiple IEEE 1451.X interfaces of different technologies (for example, IEEE p1451.5 Clause 7, and IEEE 1451.3 multidrop).

Parameters:
The [out] “moduleIds” parameter is returned by IEEE p1451.0 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, that 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.

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 IEEE p1451.0 and 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

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Args::UInt16 open( in Args::UInt16 timId, in Args::UInt16 channelId, out Args::UInt16 transCommId);</code></td>
</tr>
<tr>
<td><code>Args::UInt16 openQoS( in Args::UInt16 timId, in Args::UInt16 channelId, inout Args::QoSParams qosParams, out Args::UInt16 transCommId);</code></td>
</tr>
<tr>
<td><code>Args::UInt16 openGroup( in Args::UInt16Array timIds, in Args::UInt16Array channelIds, out Args::UInt16 transCommId);</code></td>
</tr>
<tr>
<td><code>Args::UInt16 openGroupQoS( in Args::UInt16Array timIds, in Args::UInt16Array channelIds, inout Args::QoSParams qosParams, out Args::UInt16 transCommId);</code></td>
</tr>
<tr>
<td><code>Args::UInt16 close( in Args::UInt16 transCommId);</code></td>
</tr>
<tr>
<td><code>Args::UInt16 readData ( in Args::UInt16 transCommId, in Args::TimeDuration timeout, in Args::UInt8 SamplingMode, out Args::ArgumentArray result);</code></td>
</tr>
<tr>
<td><code>Args::UInt16 writeData ( in Args::UInt16 transCommId, in Args::TimeDuration timeout, in Args::UInt8 SamplingMode, in Args::ArgumentArray value);</code></td>
</tr>
<tr>
<td><code>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);</code></td>
</tr>
<tr>
<td><code>Args::UInt16 startWriteData( in Args::UInt16 transCommId, in Args::TimeInstance triggerTime, in Args::TimeDuration timeout, in Args::UInt8 SamplingMode, in AppCallback callback, out Args::UInt16 operationId);</code></td>
</tr>
<tr>
<td><code>Args::UInt16 startStream( in Args::UInt16 transCommId, in AppCallback callback, out Args::UInt16 operationId);</code></td>
</tr>
<tr>
<td><code>Args::UInt16 cancel( in Args::UInt16 operationId);</code></td>
</tr>
</tbody>
</table>

10.2.1 IEEE1451Dot0::TransducerServices::TransducerAccess::open

IDL: Args::UInt16 open( 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 IEEE p1451.0 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 IEEE p1451.0 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);
```

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 IEEE p1451.0 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 [out] “transCommId” parameter is returned by IEEE p1451.0 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);
```

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 clauses 7 and 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 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 w/o generating a timeout error.
- Note a timeout 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); 

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 ArgumentArray.

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 p1451.0 layer on the NCAP will perform simple conversions between all the 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 down-converted 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 TEDS for 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 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 before performing the read w/o generating a timeout error.
   Note a timeout 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 timeout 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
timeout error. Note a timeout 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 “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 timeout 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 timeout error. Note a timeout 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 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.
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 IEEE p1451.0 and 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock( in Args::UInt16 transCommId, in Args::TimeDuration timeout);</td>
<td>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, nsec==0 implies no wait and can be used to test for an existing lock. A value of secs==0, nsec == -1 implies wait forever. Using a value of “wait forever” is extremely dangerous as it can create deadlocks. Return result: Error code.</td>
</tr>
<tr>
<td>unlock( in Args::UInt16 transCommId);</td>
<td></td>
</tr>
<tr>
<td>reportLocks( out Args::UInt16Array transCommIds);</td>
<td></td>
</tr>
<tr>
<td>breakLock( in Args::UInt16 transCommId);</td>
<td></td>
</tr>
<tr>
<td>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);</td>
<td></td>
</tr>
<tr>
<td>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);</td>
<td></td>
</tr>
<tr>
<td>configureAttributes( in Args::UInt16 transCommId, in Args::StringArray attributeNames);</td>
<td></td>
</tr>
<tr>
<td>trigger(in Args::UInt16 transCommId, in Args::TimeInstance triggerTime, in Args::TimeDuration timeout, in Args::UInt16 SamplingMode);</td>
<td></td>
</tr>
<tr>
<td>startTrigger( in Args::UInt16 transCommId, in Args::TimeInstance triggerTime, in Args::TimeDuration timeout, in Args::UInt16 SamplingMode, in AppCallback callback, out Args::UInt16 operationId);</td>
<td></td>
</tr>
<tr>
<td>clear( in Args::UInt16 transCommId, in Args::TimeDuration timeout, in Args::UInt8 clearMode);</td>
<td></td>
</tr>
<tr>
<td>registerStatusChange( in Args::UInt16 transCommId, in Args::TimeDuration timeout, in AppCallback callback, out Args::UInt16 operationId);</td>
<td></td>
</tr>
<tr>
<td>unregisterStatusChange( in Args::UInt16 transCommId);</td>
<td></td>
</tr>
</tbody>
</table>
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 alternate calling thread to invoke unlock( ). This 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.

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,
);
1 in Args::UInt8 cmdClassId,
2 in Args::UInt8 cmdFunctionId,
3 in Args::ArgumentArray inArgs,
4 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 timeout 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,
  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 timeout 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,
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 timeout 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 timeout 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 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.

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 clauses 7 and 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 timeout 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 timeout 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.
The “triggerTime” parameter specifies when to begin the operation. A value specified in the past will result
in an immediate timeout 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 timeout 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 timeout error. A value of secs==0, nsecs==-1 means
“wait forever.”
The “clearMode” specifies the clear mode as shown in Table 87.

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Clear All</td>
</tr>
<tr>
<td>2</td>
<td>Clear Communications channel</td>
</tr>
<tr>
<td>3</td>
<td>Clear buffers</td>
</tr>
<tr>
<td>4</td>
<td>Reset TIM State Machine</td>
</tr>
<tr>
<td>5</td>
<td>Clear TEDS Cache</td>
</tr>
<tr>
<td>6-127</td>
<td>Reserved</td>
</tr>
<tr>
<td>128-255</td>
<td>Open to Manufacturers</td>
</tr>
</tbody>
</table>

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
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.
The “timeout” is the maximum time to wait before a timeout 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 IEEE p1451.0 and called by the application to provide access to the TEDS. The methods in this interface are listed in Table 88.

Table 88—TedsAccess methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Args::UInt16 readTeds( in Args::UInt16 transCommId, in Args::TimeDuration timeout, in Args::UInt8 tedsType, out Args::ArgumentArray teds);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 writeTeds( in Args::UInt16 transCommId, in Args::TimeDuration timeout, in Args::UInt8 tedsType, in Args::ArgumentArray teds);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 readRawTeds( in Args::UInt16 transCommId, in Args::TimeDuration timeout, in Args::UInt8 tedsType, out Args::OctetArray rawTeds);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 writeRawTeds( in Args::UInt16 transCommId, in Args::TimeDuration timeout, in Args::UInt8 tedsType, in Args::OctetArray rawTeds);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 updateTedsCache( in Args::UInt16 transCommId, in Args::TimeDuration timeout, in Args::UInt8 tedsType);</td>
<td></td>
</tr>
</tbody>
</table>
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.

Parameters:
The “transCommId” specifies the desired transducer communication session.
The “timeout” specifies the duration to wait before returning a timeout error if no response is received. A
value of secs==0, nsec==0 implies no wait and may be used to only read from the cache. A value of
secs==0, nsec == -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 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 timeout error if no response is received. A
value of secs==0, nsec==0 implies no wait and may be used to only read from the cache. A value of
secs==0, nsec == -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,

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
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.

Parameters:

- The “transCommId” specifies the desired transducer communication session.
- The “timeout” specifies the duration to wait before returning a timeout error if no response is received. A value of secs==0, nsec==0 implies no wait and may be used to only read from the cache. A value of secs==0, nsec == -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:

```cpp
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 require 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 timeout error if no response is received. A value of secs==0, nsec == -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:

```cpp
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.
- The “timeout” specifies the duration to wait before returning a timeout error if no response is received. A value of secs==0, nsec == -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:**
```cpp
interface CommManager { }
```

The CommManager interface is provided by IEEE p1451.0 and 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 0.

#### Table 89—CommManager

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE1451dot0::TransducerServices::CommManager</td>
</tr>
<tr>
<td>Args::UInt16 getCommModule( in Args::UInt8 moduleId, out ModuleCommunication::Comm commObject, out Args::UInt8 type, out Args::UInt8 technologyId);</td>
</tr>
</tbody>
</table>

### 10.5.1 IEEE1451Dot0::TransducerServices::CommManager::getCommModule

**IDL:**
```cpp
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 p1451.0 processing and interact directly with the underlying communications object. By consulting the “type” parameter, the application can safely down-cast to either the “P2PComm” or “NetComm” object.

Applications must use extreme caution when accessing the underlying “Comm” objects as incorrect usage may compromise the IEEE p1451.0 layer. This method is provided to allow expansion beyond the IEEE p1451.0 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 down-cast. Valid values are represented in the [out] “technologyId” 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**

<table>
<thead>
<tr>
<th>Enumerations</th>
<th>Type Code Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>P2P_TYPE</td>
<td>Specifies a P2PComm object</td>
</tr>
<tr>
<td>1</td>
<td>NET_COMM_TYPE</td>
<td>Specifies a NetComm object</td>
</tr>
<tr>
<td>2-255</td>
<td>reserved</td>
<td></td>
</tr>
</tbody>
</table>

Return result: Error code.
10.6 IEEE1451Dot0::TransducerServices::AppCallback

IDL: interface AppCallback { }

The AppCallback interface is provided by applications and called by IEEE p1451.0 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

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Args::UInt16 measurementUpdate( in Args::UInt16 operationId, in Args::ArgumentArray measValues, in Args::UInt16 status);</td>
</tr>
<tr>
<td>Args::UInt16 actuationComplete ( in Args::UInt16 operationId, in Args::UInt16 status);</td>
</tr>
<tr>
<td>Args::UInt16 statusChange( in Args::UInt16 operationId, in Args::UInt16 status);</td>
</tr>
<tr>
<td>Args::UInt16 commandComplete( in Args::UInt16 operationId, in Args::ArgumentArray outArgs, in Args::UInt16 status);</td>
</tr>
<tr>
<td>Args::UInt16 triggerComplete( in Args::UInt16 operationId, in Args::UInt16 status);</td>
</tr>
</tbody>
</table>

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 is 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 IEEE p1451.0. 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.

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 IEEE p1451.0. 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 IEEE p1451.0. 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 IEEE p1451.0. 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.

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 IEEE p1451.0. See 9.3.1.2 for error codes.

11 Module Communications Applications Program Interface (API)

IDL: module ModuleCommunication { };
The Module Communications API provides the interface between the functions defined by IEEE p1451.0 and the communications functions defined by another member of the IEEE 1451 family of standards.

All IEEE p1451.0 interfaces are inside the IDL module “IEEE1451Dot0”.

NOTE—Throughout this section the nomenclature IEEE p1451.0 or 1451.0 refers to a device or part of a device that is in compliance with IEEE p1451.0. The terms 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 p1451.5, IEEE p1451.6 or other similar standard. IEEE Std 1451.1 and IEEE Std 1451.4 are excluded from this list.

Table 92—Transducer Services API classes and interfaces

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comm</td>
<td>The Comm abstract interface provides the mechanisms to control the “life cycle” of an IEEE 1451.X instance.</td>
</tr>
<tr>
<td>P2PComm</td>
<td>The P2PComm interface is provided to perform point-to-point communication operations.</td>
</tr>
<tr>
<td>NetComm</td>
<td>The NetComm interface is provided to perform network communication operations.</td>
</tr>
<tr>
<td>Registration</td>
<td>The Registration interface provides methods to register an IEEE 1451.X module with IEEE p1451.0.</td>
</tr>
<tr>
<td>P2PRegistration</td>
<td>The P2PRegistration interface provides methods to register specific TIMs with IEEE p1451.0.</td>
</tr>
<tr>
<td>NetRegistration</td>
<td>The NetRegistration interface provides methods to register specific TIMs and groups of TIMs with IEEE p1451.0.</td>
</tr>
<tr>
<td>Receive</td>
<td>The abstract Receive interface does not define any generic methods. It is provided for future expansion.</td>
</tr>
<tr>
<td>P2PReceive</td>
<td>The P2PReceive interface provides methods for IEEE 1451.X Point-to-point communications module to notify IEEE p1451.0 that a message has been received. It also provides the method for aborting an operation.</td>
</tr>
<tr>
<td>NetReceive</td>
<td>The NetReceive interface provides methods for IEEE 1451.X network communications module to notify IEEE p1451.0 that a message has been received. It also provides the method for aborting an operation.</td>
</tr>
</tbody>
</table>

11.1 IEEE1451Dot0::ModuleCommunication::Comm

IDL: interface Comm {

The Comm abstract interface is provided by IEEE 1451.X and called by IEEE p1451.0 to provide a common mechanism to control the “life cycle” of an IEEE 1451.X instance. The Comm methods are listed in Table 93.

Table 93—Comm methods

| 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); |
11.1.1 IEEE1451Dot0::ModuleCommunication::Comm::init

IDL: Args::UInt16 init( );

This method will be called at startup and when IEEE p1451.0 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 1451.0.

Parameters:
None.

Return result: Error code.

11.1.2 IEEE1451Dot0::ModuleCommunication::Comm::shutdown

IDL: Args::UInt16 shutdown( );

This 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 1451.0.

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 IEEE p1451.0 to put this IEEE 1451.X instance into the sleep (i.e. low power) state. This will be when IEEE p1451.0 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.

11.1.4 IEEE1451Dot0::ModuleCommunication::Comm::wakeup

IDL: Args::UInt16 wakeup();

This method is called by IEEE p1451.0 to wakeup this IEEE 1451.X instance from the sleep (i.e. low power) state. This will be when IEEE p1451.0 has previously put this IEEE 1451.X interface to sleep and now needs to resume communication.

Parameters:
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 is a low-level mechanism to send an arbitrary command to the local IEEE 1451.X. 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.

Parameters:
The “operationId” specifies the desired command class code. IEEE 1451.X will define legal operations.
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 “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 IEEE p1451.0 to configure this local instance of the IEEE 1451.X interface. The contents of the “params” ArgumentArray is IEEE 1451.X specific.

Parameters:
The “params” parameter is an ArgumentArray of configuration state variables. Each state variable of interest to IEEE 1451.X should be retrieved from this array to setup the IEEE 1451.X instance.

Return result: Error code.

11.1.7 IEEE1451Dot0::ModuleCommunication::Comm::getLocalConfiguration(delete)

IDL: Args::UInt16 getLocalConfiguration (outout Args::ArgumentArray params );

This method is called by IEEE p1451.0 to retrieve the configuration for local instance of the IEEE 1451.X interface. The contents of the “params” ArgumentArray is IEEE 1451.X specific.

Parameters:
The [out] “params” parameter is an ArgumentArray of configuration state variables. Each state variable of interest to IEEE 1451.X should be returned in this array to allow setup 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 is a low-level mechanism to send an arbitrary command to the local IEEE 1451.X. 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.

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 is a generic mechanism to describe 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.

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Logical Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Point to Point (P2P)</td>
</tr>
<tr>
<td>2</td>
<td>Network (Net)</td>
</tr>
<tr>
<td>3-127</td>
<td>Reserved</td>
</tr>
<tr>
<td>128-255</td>
<td>Open to manufacturers</td>
</tr>
</tbody>
</table>

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 IEEE 1451.X and called by IEEE p1451.0 to perform communication operations. The point-to-point Comm methods are listed in Table 95.

### Table 95—Point-to-point Comm methods

<table>
<thead>
<tr>
<th>Method Description</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Args::UInt16 read( in Args::TimeDuration timeout, inout Args::UInt32 len, out Args::OctetArray payload, out Args::Boolean last);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 write( in Args::TimeDuration timeout, in Args::OctetArray payload, in Args::Boolean last);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 flush( );</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 readSize( out Args::UInt32 cacheSize);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 setPayloadSize( in Args::UInt32 size);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 abort( );</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 commStatus( out Args::UInt16 statusCode );</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 setRemoteConfiguration( in Args::TimeDuration timeout, in Args::ArgumentArray params);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 getRemoteConfiguration( in Args::TimeDuration timeout, out Args::ArgumentArray params);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 sendRemoteCommand( in Args::TimeDuration timeout, in Args::UInt8 cmdClassId, in Args::UInt8 cmdFunctionId, in Args::ArgumentArray inArgs, out Args::ArgumentArray outArgs);</td>
<td></td>
</tr>
</tbody>
</table>

#### 11.2.1 IEEE1451Dot0::ModuleCommunication::P2Comm::read

**IDL:** 
```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 IEEE p1451.0 to retrieve information on the current communication operation. This method is always called by IEEE p1451.0 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 IEEE p1451.0 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:**
- 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 “maxLen” parameter indicates the maximum number of octets to be transferred by IEEE 1451.X.
- The [out] “payload” parameter is the OctetArray provided by IEEE p1451.0 to IEEE 1451.X. IEEE 1451.X will transfer available data into this array.
- The [out] “last” parameter indicates if additional calls to read( ) should be performed by 1451.0. A false value indicates that IEEE 1451.X has more octets to communicate to IEEE p1451.0 and IEEE p1451.0 must make additional read( ) calls. A true value indicates that the complete payload has been transferred from IEEE 1451.X to IEEE p1451.0.

**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 IEEE p1451.0 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 timeout 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 IEEE p1451.0 to provide additional portions of the payload. A false value indicates that IEEE p1451.0 has more octets to send and will call write( ) additional times. A true value indicates that the complete payload has been transferred from IEEE p1451.0 to IEEE 1451.X.

Return result: Error code.

11.2.3 IEEE1451Dot0::ModuleCommunication::P2PComm::flush

IDL: Args::UInt16 flush();

This method is called by IEEE p1451.0 to flush any cached information to the remote side. In most cases, this call is not used because IEEE 1451.X will always perform the transfer when the write( ) call is invoked with “last” set to true.

Parameters:
none.

Return result: Error code.

11.2.4 IEEE1451Dot0::ModuleCommunication::P2PComm::readSize

IDL: Args::UInt16 readSize( out Args::UInt32 cacheSize);

This method is called by IEEE p1451.0 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 IEEE 1451.X 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 IEEE p1451.0 to set the total number of octets available in the full payload. This method should be called when IEEE p1451.0 expects to make multiple calls to write( ) to pass the payload to IEEE 1451.X 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 IEEE 1451.X 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 IEEE p1451.0 on the initiating node to abort or reset a communication channel. If IEEE 1451.X has already initiated communication to the remote node, IEEE 1451.X 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 IEEE p1451.0 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.

Return result: Error code.

Table 96—State machine status codes

<table>
<thead>
<tr>
<th>Status Enumeration</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Idle</td>
</tr>
<tr>
<td>2</td>
<td>InitiatorWriting</td>
</tr>
<tr>
<td>3</td>
<td>InitiatorClosing</td>
</tr>
<tr>
<td>4</td>
<td>InitiatorReceivePending</td>
</tr>
<tr>
<td>5</td>
<td>InitiatorReading</td>
</tr>
<tr>
<td>6</td>
<td>InitiatorAborting</td>
</tr>
<tr>
<td>7</td>
<td>ReceiverIncomingMsg</td>
</tr>
<tr>
<td>8</td>
<td>ReceiverReading</td>
</tr>
<tr>
<td>9</td>
<td>ReceiverWriting</td>
</tr>
<tr>
<td>10</td>
<td>ReceiverOutgoingMsg</td>
</tr>
<tr>
<td>11</td>
<td>ReceiverAborting</td>
</tr>
<tr>
<td>12-127</td>
<td>Reserved</td>
</tr>
<tr>
<td>128-255</td>
<td>Open to manufacturers</td>
</tr>
</tbody>
</table>

Copyright © 2007 IEEE. All rights reserved. This is an unapproved IEEE Standards Draft, subject to change.
11.2.8 IEEE1451Dot0::ModuleCommunication::P2PComm::setRemoteConfiguration

IDL: Args::UInt16 setRemoteConfiguration(
  in Args::TimeDuration timeout,
  in Args::ArgumentArray params);

This method is called by IEEE p1451.0 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 timeout 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 IEEE 1451.X should be retrieved from this array to setup 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 IEEE p1451.0 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 timeout error. A value of secs==0, nsecs==-1 means “wait forever.”
The [out] “params” parameter is an ArgumentArray of configuration state variables. Each state variable of interest to IEEE 1451.X 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 is a low-level mechanism to send an arbitrary command to the remote IEEE 1451.X. It is recommended that the IEEE 1451.X 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 are 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 timeout 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 IEEE 1451.X and called by IEEE p1451.0 to perform communication operations. The methods that implement this interface are listed in Table 97.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Args::UInt16 open( in Args::UInt16 destId, in Args::Boolean twoWay, out Args::UInt16 maxPayloadLen, out Args::UInt16 commId);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 openQoS( in Args::UInt16 destId, in Args::Boolean twoWay, out Args::UInt16 maxPayloadLen, out Args::UInt16 commId, inout Args::QoSParams qosParams);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 close( in Args::UInt16 commId);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 readMsg( in Args::UInt16 commId, in Args::TimeDuration timeout, inout Args::UInt32 len, out Args::OctetArray payload, out Args::Boolean last);</td>
<td></td>
</tr>
<tr>
<td>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);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 writeMsg(in Args::UInt16 commId, in Args::TimeDuration timeout, in Args::OctetArray payload, in Args::Boolean last, in Args::UInt16 msgId);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 writeRsp(in Args::UInt16 commId, in Args::TimeDuration timeout, in Args::OctetArray payload, in Args::Boolean last);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 flush( in Args::UInt16 commId);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 readSize( in Args::UInt16 commId, out Args::UInt32 cacheSize);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 setPayloadSize( in Args::UInt16 commId, in Args::UInt32 size);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 abort(in Args::UInt16 commId);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 commStatus(in Args::UInt16 commId, in Args::UInt16 msgId, out Args::UInt16 statusCode );</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 discoverDestinations( );</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 joinGroup( in Args::UInt16 groupId, in Args::UInt16 destId);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 leaveGroup( in Args::UInt16 groupId, in Args::UInt16 destId);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 lookupDestId( in Args::UInt16 commId, out Args::UInt16 destId);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 setRemoteConfiguration( in Args::UInt16 commId, in Args::TimeDuration timeout, in Args::ArgumentArray params);</td>
<td></td>
</tr>
<tr>
<td>Args::UInt16 getRemoteConfiguration( in Args::UInt16 commId, in Args::TimeDuration timeout, out Args::ArgumentArray params);</td>
<td></td>
</tr>
<tr>
<td>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);</td>
<td></td>
</tr>
</tbody>
</table>
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 IEEE p1451.0 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.

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 IEEE p1451.0 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 IEEE 1451.X 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 IEEE p1451.0 to close a communication channel. This is called by IEEE p1451.0 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.
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 IEEE p1451.0 to retrieve information on the current communication operation. This method is always called by IEEE p1451.0 on the receiving node for both one-way and two-way communication sequences.

In cases where IEEE p1451.0 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 timeout 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 IEEE 1451.X. In the reply it indicates the number of octets transferred.
The [out] “payload” parameter is the OctetArray provided by IEEE p1451.0 to IEEE 1451.X. IEEE 1451.X 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 1451.0. A false value indicates that IEEE 1451.X has more octets to communicate to IEEE p1451.0 and IEEE p1451.0 must make additional readMsg( ) calls. A true value indicates that the complete payload has been transferred from IEEE 1451.X to IEEE p1451.0.

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 IEEE p1451.0 to retrieve the response information on the current communication operation. This method is always called by IEEE p1451.0 on the initiating node for two-way communication sequences.

In cases where IEEE p1451.0 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 timeout 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 IEEE 1451.X.

The [out] "payload" parameter is the OctetArray provided by IEEE p1451.0 to IEEE 1451.X. IEEE 1451.X will transfer available data into this array. Note the length of the OctetArray may be less than the maxLen parameter.

The [out] "last" parameter indicates if additional calls to readRsp() should be performed by 1451.0. A false value indicates that IEEE 1451.X has more octets to communicate to IEEE p1451.0 and IEEE p1451.0 must make additional readRsp() calls. A true value indicates that the complete payload has been transferred from IEEE 1451.X to IEEE p1451.0.

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 IEEE p1451.0 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 timeout 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 IEEE p1451.0 to provide additional portions of the payload. A false value indicates that IEEE p1451.0 has more octets to send and will call writeMsg() additional times. A true value indicates that the complete payload has been transferred from IEEE p1451.0 to IEEE 1451.X.

The "msgId" parameter defines the "message ID" for this transaction. In two-way transactions, IEEE 1451.X 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 IEEE p1451.0 to return a response to a command.

Parameters:

The "commId" parameter specifies the communication channel.
1. 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.”
2. The “payload” parameter is the OctetArray to be communicated.
3. The “last” parameter indicates if additional calls to writeRsp() will be made by IEEE p1451.0 to provide additional portions of the payload. A false value indicates that IEEE p1451.0 has more octets to send and will call writeRsp() additional times. A true value indicates that the complete payload has been transferred from IEEE p1451.0 to IEEE 1451.X.

Return result: Error code.

11.3.8 IEEE1451Dot0::ModuleCommunication::NetComm::flush

IDL: Args::UInt16 flush(in Args::UInt16 commId);

This method is called by IEEE p1451.0 to flush any cached information to the remote side. In most cases, this call is not used because IEEE 1451.X 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 IEEE p1451.0 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 IEEE 1451.X 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 IEEE p1451.0 to set the total number of octets available in the full payload. This method should be called when IEEE p1451.0 expects to make multiple calls to writeMsg() or writeRsp() to pass the payload to IEEE 1451.X in pieces. In cases where a single writeMsg() or writeRsp() call is made (i.e. last is true), this call is not needed as IEEE 1451.X shall internally call this method to record the payload size.

Parameters:
The “commId” parameter specifies the communication channel.
The “size” parameter specifies the full size of the payload.
11.3.11 IEEE1451Dot0::ModuleCommunication::NetComm::abort

IDL: Args::UInt16 abort(in Args::UInt16 commId);

This method is called by IEEE p1451.0 on the initiating node to abort or reset a communication channel. If
IEEE 1451.X has already initiated communication to the remote node, IEEE 1451.X 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 IEEE p1451.0 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 IEEE p1451.0 to force a discovery and registration of all TIMs accessible through
this IEEE 1451.X instance. In response, IEEE 1451.X will invoke the Register::registerDest( ) method for
each destination.

In normal situations, IEEE 1451.X should automatically call registerDest( ) at initialization time and during
a hot-swap event.

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 IEEE p1451.0 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 IEEE p1451.0 to remove a destination from a multicast group. A call to close() for a groupld 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 IEEE p1451.0 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.

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 IEEE p1451.0 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
timeout 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 IEEE p1451.0 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
timeout 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.

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 is a low-level mechanism to send an arbitrary command to the remote IEEE 1451.X. It is
recommended that the IEEE 1451.X 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 are 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 timeout 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.
11.4 IEEE1451Dot0::ModuleCommunication::Registration

IDL: interface Registration { }

The abstract Registration Interface is a collection of methods provided by IEEE p1451.0 and called by IEEE 1451.X. The methods are listed in Table 98. See the interfaces P2PRegistration and NetRegistration for additional methods.

<table>
<thead>
<tr>
<th>Table 98—Registration interface methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE1451Dot0::ModuleCommunication::Registration</td>
</tr>
<tr>
<td>Args::UInt16 unregisterModule (in Args::UInt8 moduleId);</td>
</tr>
<tr>
<td>Args::UInt16 reportModules (in Args::UInt8 maxLen, in Args::UInt8 offset, out Args::UInt8Array moduleIds);</td>
</tr>
<tr>
<td>Args::UInt16 getCommModule (in Args::UInt8 moduleId, out Comm commObject, out Args::UInt8 type, out Args::UInt8 technologyId);</td>
</tr>
</tbody>
</table>

11.4.1 IEEE1451Dot0::ModuleCommunication::Registration::registerModule

There are two methods used to register an IEEE 1451.X interface with 1451.0. 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.

<table>
<thead>
<tr>
<th>Table 99—Communication module identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumeration</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9-254</td>
</tr>
<tr>
<td>255</td>
</tr>
</tbody>
</table>

11.4.2 IEEE1451Dot0::ModuleCommunication::Registration::unRegisterModule

IDL: Args::UInt16 unregisterModule (in Args::UInt8 moduleId);

This method is provided by IEEE p1451.0 and called by IEEE 1451.X to de-register an interface. Its purpose is to inform IEEE p1451.0 that this instance of the IEEE 1451.X interface is no longer available.

Parameters:
The “moduleId” parameter specifies which interface to de-register. IEEE p1451.0 will remove all knowledge of this interface from its cache.
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 IEEE p1451.0 and called by IEEE 1451.X 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.

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 down-cast 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 p1451.0 and IEEE 1451.X layers. This method is provided to allow expansion beyond the IEEE p1451.0 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 down-cast. 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 IEEE p1451.0 and called by IEEE 1451.X for the P2P case.
The methods that make up this interface are listed in Table 100.
Table 100—P2PRegistration interface methods

<table>
<thead>
<tr>
<th>Method Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE1451Dot0::ModuleCommunication::P2PRegistration::registerModule</td>
</tr>
<tr>
<td>Args::UInt16 registerModule( in P2PComm commInterface, in Args::UInt8 technologyId, in Args::String name, out Args::UInt8 moduleId)</td>
</tr>
<tr>
<td>IEEE1451Dot0::ModuleCommunication::P2PRegistration::registerDestination</td>
</tr>
<tr>
<td>Args::UInt16 registerDestination( in Args::UInt8 moduleId, in Args::UInt16 maxPayloadSize)</td>
</tr>
<tr>
<td>IEEE1451Dot0::ModuleCommunication::P2PRegistration::unregisterDestination</td>
</tr>
<tr>
<td>Args::UInt16 unregisterDestination( in Args::UInt8 moduleId)</td>
</tr>
</tbody>
</table>

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);

This method is provided by IEEE p1451.0 and called by IEEE 1451.X during configuration for the “P2P” case. Its purpose is to inform IEEE p1451.0 that IEEE 1451.X is available.

Parameters:
The “commInterface” parameter specifies the P2PComm interface. See Table 94 for possible values. IEEE p1451.0 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 IEEE p1451.0 and called by IEEE 1451.X during configuration for the “P2P” case. Its purpose is to inform IEEE p1451.0 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 IEEE p1451.0 and called by IEEE 1451.X during configuration for the “P2P” case. Its purpose is to inform IEEE p1451.0 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.

11.6 IEEE1451Dot0::ModuleCommunication::NetRegistration

IDL: interface NetRegistration { }

The NetRegistration Interface is provided by IEEE p1451.0 and called by IEEE 1451.X for the network case. The methods required by this interface are listed in Table 101.

<table>
<thead>
<tr>
<th>Table 101—NetRegistration interface methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE1451Dot0::ModuleCommunication::NetRegistration</td>
</tr>
<tr>
<td>Args::UInt16 registerModule( in NetComm commInterface, in Args::UInt8 technologyId, in Args:: String name, out Args::UInt8 moduleId);</td>
</tr>
<tr>
<td>Args::UInt16 registerDestination ( in Args::UInt8 moduleId, out Args::UInt16 destId );</td>
</tr>
<tr>
<td>Args::UInt16 unregisterDestination ( in Args::UInt8 moduleId, in Args::UInt16 destId);</td>
</tr>
<tr>
<td>Args::UInt16 reportDestinations ( in Args::UInt8 moduleId, in Args::UInt16 maxLen, in Args::UInt16 offset, out Args::UInt16Array destinations);</td>
</tr>
<tr>
<td>Args::UInt16 reportGroups ( in Args::UInt8 moduleId, in Args::UInt16 maxLen, in Args::UInt16 offset, out Args::UInt16Array groups);</td>
</tr>
<tr>
<td>Args::UInt16 reportGroupMembers ( in Args::UInt8 moduleId, in Args::UInt16 groupId, in Args::UInt16 maxLen, in Args::UInt16 offset, out Args::UInt16Array groups);</td>
</tr>
</tbody>
</table>

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 IEEE p1451.0 and called by IEEE 1451.X during configuration for the “Net” case. Its purpose is to inform IEEE p1451.0 that IEEE 1451.X is available.

Parameters:
The “commInterface” parameter specifies the NetComm interface. See Table 94 for possible values. IEEE p1451.0 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};

This method is provided by IEEE p1451.0 and called by IEEE 1451.X to register a new destination. Its purpose is to inform IEEE p1451.0 that the new destination is available. It is the responsibility of IEEE 1451.X to detect if a new TIM appears in the system and to register it with IEEE p1451.0.

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 1451.0. IEEE 1451.X will need to cache appropriate communication endpoint information for this “destId.” The “destId” parameter will be passed back to IEEE p1451X. 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 IEEE p1451.0 and called by IEEE 1451.X to unregister a destination. Its purpose is to inform IEEE p1451.0 that the destination is no longer available. It is the responsibility of IEEE 1451.X to detect if a TIM disappears from the system and unRegister that TIM with IEEE p1451.0.

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 IEEE p1451.0 and called by IEEE 1451.X 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.
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 IEEE p1451.0 and called by IEEE 1451.X 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 IEEE p1451.0 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 IEEE p1451.0 and called by IEEE 1451.X 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 IEEE p1451.0 to use to return the values. These are the “destId” values for each known group destination.

Return result: Error code.

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 IEEE p1451.0 and called by IEEE 1451.X. The methods used to implement this interface are listed in Table 102.

### Table 102—P2PReceive interface methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE1451Dot0::ModuleCommunication::P2PReceive::abort</td>
<td>Args::UInt16 abort( in Args::UInt16 status );</td>
</tr>
<tr>
<td></td>
<td>Args::UInt16 notifyMsg( in Args::Boolean twoWay, in Args::UInt32 payloadLen, in Args::UInt32 cacheLen, in Args::UInt16 status);</td>
</tr>
<tr>
<td></td>
<td>Args::UInt16 notifyRsp( in Args::UInt32 payloadLen, in Args::UInt32 cacheLen, in Args::UInt16 status);</td>
</tr>
</tbody>
</table>

#### 11.8.1 IEEE1451Dot0::ModuleCommunication::P2PReceive::abort

IDL: Args::UInt16 abort( in Args::UInt16 status );

This method is called by IEEE 1451.X to abort the current operation. This will be called by the receiving node’s IEEE 1451.X layer when the IEEE p1451.0 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 IEEE 1451.X. Typically, this return value will be ignored by IEEE 1451.X.

#### 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 IEEE p1451.0 and called by IEEE 1451.X 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.
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 IEEE 1451.X. Typically, this return value will be ignored by IEEE 1451.X.

#### 11.8.3 IEEE1451Dot0::ModuleCommunication::P2PReceive::notifyRsp

IDL: Args::UInt16 notifyRsp(}
This method is provided by IEEE p1451.0 and called by IEEE 1451.X 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 IEEE 1451.X. Typically, this return value will be ignored by IEEE 1451.X.

11.9 IEEE1451Dot0::ModuleCommunication::NetReceive

IDL: interface NetReceive {

The NetReceive interface is a collection of methods provided by IEEE p1451.0 and called by IEEE 1451.X. The methods used to implement this interface are listed in Table 103.

Table 103—NetReceive interface methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Args::UInt16 abort( in Args::UInt16 commId, in Args::UInt16 status);</td>
<td>This method is called by IEEE 1451.X to abort the current operation. This will be called by the receiving node’s IEEE 1451.X layer when the IEEE p1451.0 command processing has already been initiated and must be terminated.</td>
</tr>
<tr>
<td>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);</td>
<td></td>
</tr>
<tr>
<td>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);</td>
<td></td>
</tr>
</tbody>
</table>

11.9.1 IEEE1451Dot0::ModuleCommunication::NetReceive::abort

IDL: Args::UInt16 abort(
  in Args::UInt16 commId,
  in Args::UInt16 status);

This method is called by IEEE 1451.X to abort the current operation. This will be called by the receiving node’s IEEE 1451.X layer when the IEEE p1451.0 command processing has already been initiated and must be terminated.

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 IEEE 1451.X. Typically, this return value will be ignored by IEEE 1451.X.
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 IEEE p1451.0 and called by IEEE 1451.X 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 p1451.0 layer does not call open( ) or close( ) on this “rcvCommId.” This “rcvCommId” is
managed by IEEE 1451.X.

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 IEEE 1451.X. Typically, this return value will be ignored by
IEEE 1451.X

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 IEEE p1451.0 and called by IEEE 1451.X when an incoming response
message is available. This is invoked on the initiating node for two-way communications.

Note that the receiving node’s IEEE p1451.0 layer does not call open( ), so a “rcvCommId” is automatically provided
in this call. This “rcvCommId” is managed by IEEE 1451.X.

For “two way” communications, the receiving IEEE p1451.0 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 p1451.0 layer does not call open( ) or close( ) on this “rcvCommId.” This “rcvCommId” is
managed by IEEE 1451.X.

The “msgId” is passed to the IEEE p1451.0 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.
12 HTTP protocol

Hypertext Transfer Protocol (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, used to define HTTP, is
analogous to the IEEE p1451.0 “NCAP-End User.” The NCAP can be compared to the “server” as it
serves data to the attached network, and the End-User can be compared to 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 also 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 section
of this standard 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 the user
holds accountable for the results of the interaction. This method can be used to take a command, or
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 Transducer Service Interface (TSI) is an NCAP-only API used by measurement and control
applications to access the IEEE p1451.0 system. 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 Service Interface of IEEE 1451.0 contains five interfaces: TransducerAccess,
TransducerManager, TimDiscovery, TEDSManager, and AppCallback. The first four interfaces are
implemented by IEEE p1451.0 and are called by the measurement applications. If the application desires
advanced optional features, it will need to implement the “AppCallback” interface, which IEEE p1451.0
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 IEEE p1451.0 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. Similarly the “A” means “Actuator.” Users can send a HTTP request to the HTTP server on the NCAP and get HTTP response from the HTTP server. The request–response process can be described as follows:

1) A user or client sends a HTTP request to the HTTP server on the IEEE 1451.0 NCAP.
2) The HTTP server on the NCAP receives a HTTP request, processes it, and then calls the corresponding IEEE 1451.0 API.
3) The IEEE 1451.0 API calls on the IEEE 1451.x API to communicate with the IEEE 1451.x TIM and get the results from the TIM.
4) The HTTP server on the NCAP gets the results from the IEEE 1451.0 API, and then returns the HTTP response to the user.

![Figure 23—HTTP access to an IEEE 1451.0 NCAP](image)

**12.1.1 HTTP message format**

In this clause, we use 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. 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

<table>
<thead>
<tr>
<th>Field</th>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;host&gt;</td>
<td>The host portion of this statement will include the target IEEE 1451 node domain name.</td>
<td>&lt;host&gt; = “192.168.1.91”</td>
</tr>
<tr>
<td>&lt;port&gt;</td>
<td>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 down side 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.</td>
<td>&lt;port&gt;=&quot;80&quot;</td>
</tr>
<tr>
<td>&lt;path&gt;</td>
<td>The path indicates the 1451 path including the command itself.</td>
<td>&lt;path&gt;=&quot;1451/TransducerAccess/ReadData&quot;</td>
</tr>
<tr>
<td>&lt;parameters&gt;</td>
<td>The parameters associated with the command.</td>
<td>&lt;parameters&gt;=&quot;timId=1&amp;channelId=2&amp;timeout=14&amp;samplingMode=continuous&amp;format=text&quot;</td>
</tr>
</tbody>
</table>

Table 105—Example parameters

<table>
<thead>
<tr>
<th>Field</th>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>timId</td>
<td>The identifier of selected TIM See 10.1.2</td>
<td>1</td>
</tr>
<tr>
<td>channelId</td>
<td>The identifier of the selected TransducerChannel of the TIM See 10.2.1</td>
<td>2</td>
</tr>
<tr>
<td>timeout</td>
<td>This argument specifies how long to wait to perform the reading before generating a timeout error.</td>
<td>14</td>
</tr>
<tr>
<td>samplingMode</td>
<td>This argument specifies the sampling mechanism. See 5.10.1 and 7.1.2.4</td>
<td>Continuous</td>
</tr>
<tr>
<td>format</td>
<td>“format” is the desired return format. The options are “text,” “HTML” or “xml.” The contents of the string are case insensitive.</td>
<td>text</td>
</tr>
</tbody>
</table>

Table 106—HTTP API

<table>
<thead>
<tr>
<th>API Type</th>
<th>name</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery API</td>
<td>TIMDiscovery</td>
<td>1451/Discovery/TIMDiscovery</td>
</tr>
<tr>
<td></td>
<td>TransducerDiscovery</td>
<td>1451/Discovery/TransducerDiscovery</td>
</tr>
<tr>
<td>Transducer Access API</td>
<td>ReadData</td>
<td>1451/TransducerAccess/ReadData</td>
</tr>
<tr>
<td></td>
<td>StartReadData</td>
<td>1451/TransducerAccess/StartReadData</td>
</tr>
<tr>
<td></td>
<td>MeasurementUpdate</td>
<td>1451/TransducerAccess/MeasurementUpdate</td>
</tr>
<tr>
<td></td>
<td>WriteData</td>
<td>1451/TransducerAccess/WriteData</td>
</tr>
<tr>
<td></td>
<td>StartWriteData</td>
<td>1451/TransducerAccess/StartWriteData</td>
</tr>
<tr>
<td>TEDS Manager API</td>
<td>ReadTeds</td>
<td>1451/TEDSManger/ReadTeds</td>
</tr>
<tr>
<td></td>
<td>ReadRawTeds</td>
<td>1451/TEDSManger/ReadRawTeds</td>
</tr>
<tr>
<td></td>
<td>WriteTeds</td>
<td>1451/TEDSManger/WriteTeds</td>
</tr>
<tr>
<td></td>
<td>WriteRawTeds</td>
<td>1451/TEDSManger/WriteRawTeds</td>
</tr>
<tr>
<td></td>
<td>UpdateTedsCache</td>
<td>1451/TEDSManger/UpdateTedsCache</td>
</tr>
<tr>
<td>Transducer Manager API</td>
<td>SendCommand</td>
<td>1451/TransducerManager/SendCommand</td>
</tr>
<tr>
<td></td>
<td>StartCommand</td>
<td>1451/TransducerManager/StartCommand</td>
</tr>
<tr>
<td></td>
<td>CommandComplete</td>
<td>1451/TransducerManager/CommandComplete</td>
</tr>
<tr>
<td></td>
<td>Trigger</td>
<td>1451/TransducerManager/Trigger</td>
</tr>
<tr>
<td></td>
<td>StartTrigger</td>
<td>1451/TransducerManager/StartTrigger</td>
</tr>
</tbody>
</table>
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 webpage in HTTP 1.1 format. The webpage 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.

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 rightmost ordinal being indexed first. A CR/LF sequence shall be returned after each complete set of rightmost 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 \texttt{Args::UInt16 reportTims()} as described in clause 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 or ASCII format.

12.2.1.1 Input parameters

The following parameters shall be supplied with this API call.

\texttt{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.

\texttt{UInt16 errorCode:} error information as defined in 9.3.1.2.

\texttt{UInt16Array timeIds:} timIds of all TIMs available of NCAP

12.2.1.3 TIMDiscovery XML response schema

If the response format is “XML” the following schema shall be used for the response.

```xml
<?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="timeIds" 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 \texttt{Args::UInt16 reportChannels()} as described in clause 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.
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.

```xml
<?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 ReadData 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 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 timeout error if no response is received. A value of secs==0, nsec == -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.

12.3.1.1.2 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 response schema

If the response format is “XML” the following schema shall be used for the response.

```xml
<?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 is 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.

TimeDuration timeout: This argument specifies how long to wait to perform the reading without generating a timeout error if no response is received. A value of secs==0, nsec == -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 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
<?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
- 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.
1.2.3.1.3 Measurement Update XML response schema

If the response format is "XML" the following schema shall be used for the response.

```xml
<?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).

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
timeout error if the return is not received. A value of secs==0, nsec == -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.
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
<?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 (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 the specified TransducerChannel
- 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 timeout error if no response is received. A value of secs==0, nsec==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
<?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://www.ieee.org/1451/"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <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>
```

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 the 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 timeout error if no response is received. A value of secs==0, nsec == -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 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 TEDS Access Code.
- **ArgumentArray teds**: This array contains the data read from the specified TEDS.
12.4.1.3 ReadTEDS XML response schema

If the response format is “XML” the following schema shall be used for the response.

```xml
<?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 clause 6.8 of RFC 2045.

Path: 1451/TEDSManager/ReadRawTeds

GET: Retrieving 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 timeout error if no response is received. A value of `secs==0, nsec == -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 TEDS type as listed in Table 17 where it is called TEDS Access Code.
- `_String responseFormat`: specifies the response format.

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
<?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="ReadRawTEDSHTTPResponse"/>
  <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.
- **TimeDuration timeout**: This argument specifies how long to wait after writing the TEDS before generating a timeout error if no response is received. A value of `secs==0, nsec == -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 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
<?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 clause 6.8 of RFC 2045.

Path: 1451/TEDSManager/WriteRawTeds

POST: Write 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
  timeout error if no response is received. A value of secs==0, nsec == -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 TEDS Access Code.
- **OctetArray rawTEDS**: Raw TEDS data of specified TransducerChannel or of specified TIM.
- **CTSTR 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 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
<?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>
```

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: update 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 timeout error.
- UInt8  tedsType: specifies the tedsType as listed in Table 17 where it is called TEDS Access Code.

12.4.5.2 UpdateTedsCache HTTP response

The response to this API call shall contain the following parameters.

- UInt16 errorCod: 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 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
<?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
```
12.5 Transducer Manager API

This API contains four aaaaAPIs. 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: send 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 timeout error if no response is received. A value of secs==0, nsec == -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.

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"?>
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: send 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 timeout 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.

- 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
<?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:sequence>
</xs:complexType>
```
12.5.3 CommandComplete

This method completes a non-blocking StartCommand 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 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
- 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
<?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 timeout 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
- **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
<?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**: start a trigger to transducer available in 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 timeout error.
- **UInt16 SamplingMode**: specifies the sampling mode. See 5.11 for details.
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.
- 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
<?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>
```
(informative)

Bibliography

This annex provides references useful in understanding the body of this standard. Also included are references to commercial, trade, or governmental non-balloted standards controlled by specific organizations. The notation [Ax] within this standard is a reference to the document in this annex denoted by the same notation.


[A2]: IEEE 100 The Authoritative Dictionary of IEEE Standards Terms


[A7]: IEEE P1451.6, Draft Standard for a Smart Transducer Interface for Sensors and Actuators - A High-speed CAN open-based Transducer Network Interface for Intrinsically Safe (IS) and Nonintrinsically Safe Applications


---

6 Available from Aligent Technical Publications Department, 1501 Page Mill Road, Mail Stop 2L, Palo Alto, California, 94304, (415) 857-4573
Annex A (informative)

Guidance to transducer services interface

Measurement and control applications interact with the IEEE p1451.0 system 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 p1451.0 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.

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 while 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.

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 timeout 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, there are a large number of commands that are defined in the IEEE p1451.0 system. 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.

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.
B.3 IEEE p1451.0 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 IEEE p1451.0 are not part of the IEEE p1451.0 standard, this diagram is only an aid to explain one way that the IEEE p1451.0 processing could be structured.

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 p1451.0 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:

1. 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.

2. 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 ten Octets long.
   - TransducerChannel identifier of 0x0001 as a UInt16. This is the TransducerChannel number to be read.
   - Command Class of 0x02 as a UInt8 that indicates an “operational” command class
   - Command Function code as a UInt8. The command code is “Read TransducerChannel data-set segment” (value 1) in this case.
   - Length of variable portion of 0x0004 as a UInt16.
   - The DataSetOffset is the only argument of the command and it contains 0x00000000 as a UInt32 since the data-set contains a single value.
   - 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.
   - When the TIM-side processing has completed, IEEE 1451.X will transfer a result OctetArray back to the NCAP. This will result in the notify() method being invoked.
   - The Notify processing wakes up the blocked application thread.
   - The read() method is invoked to retrieve the response OctetArray.
   - The Codec’s decode() method is used to convert the OctetArray back to an ArgumentArray.
   - The CorrectionEngine’s convert() method is used to generate corrected values (i.e. conversion from TIM-side to NCAP-side units).
   - The ArgumentArray is further formatted to return the desired measurement attributes. This may involve extracting select information from cached TEDS (for example, units).
   - The ArgumentArray is returned as an “out” parameter back to the application.
Figure B.2—Non-blocking Read Example

MeasApp:AppCallback

:err:=open(tid, channelId, transducerCommId)

open: return

:err:=startRead(transducerCommId, triggerTime, timeout, triggerMode, callback, operationId)

read: return

:err:=measurementUpdate(operationId, value, status)

:procRes():sults()

measurementUpdate: return
Figure B.3—SendCommand sequence diagram
Figure B.4—Details of the read operation sequence diagram
Annex B (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. There are four aspects of this API to be aware of:

1. 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.

2. 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.

3. 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.

4. 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 IEEE 1451.X and called by IEEE p1451.0 to control communications between the NCAP and TIMs. The “Registration” interface is provided by IEEE p1451.0 and called by IEEE 1451.X to register itself and to register known destinations. The “Receive” interface is provided by IEEE p1451.0 and called by IEEE 1451.X when incoming communication messages are received by IEEE 1451.X 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 communicating 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.
Figure C.1—Communications Methods

- <<interface>> Comm
  - Common Methods
  - <<interface>> P2PComm
    - Point-to-point Interface Methods
  - <<interface>> NetComm
    - Network Interface Methods

- <<interface>> Registration
  - Common Methods
  - <<interface>> P2PRegistration
    - Point-to-point Methods
  - <<interface>> NetRegistration
    - Network Methods
C.2 Symmetric APIs

These APIs are symmetric and supports 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 if the device needs to initiate communication to many devices, or if 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 if 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 p1451.0 implementation. This negates the need to use some methods of the Registration interface.

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. IEEE p1451.0 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 IEEE p1451.0 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, IEEE p1451.5 Clause 7). 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 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 p1451.0 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 p1451.0 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 IEEE p1451.0 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 p1451.5-802.11, each node has a unique IP address and network ports allocated for IEEE p1451.0 communications. Each IEEE 1451.X technology could have different set of required parameters.

Although the nature of these parameters is specific to each IEEE 1451.X technology, IEEE p1451.0 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.
C.6 The destination identifier “destId” parameter

In IEEE p1451.0, 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 IEEE p1451.0 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 p1451.0 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 p1451.0 layer by calling the registerDest( ) method. IEEE p1451.0 will assign the unique “destId” and IEEE 1451.X shall cache appropriate network “node” information and associate it with this “destId.” For example, a IEEE p1451.5-802.11 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 p1451.0 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 p1451.0 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 p1451.0 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 p1451.0’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 p1451.0 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 IEEE p1451.0 is finished, it will invoke close( ) to notify IEEE 1451.X 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 p1451.0 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 IEEE p1451.0 and provide a unique “commId.” In this case, open( ) is not called on the receiving side. IEEE p1451.0 will use that “commId” to read the incoming network data via the readMsg( ) call. Optionally, IEEE p1451.0 will use that “commId” to send a response back to the initiating node. IEEE p1451.0 will not call close( ) on this commId because this is handled internally by IEEE 1451.X.

C.8 The message transaction identifier “msgId” parameter

When initiating communications after successfully calling the open( ) or openQoS( ) calls, the IEEE p1451.0 layer will begin communications by calling the writeMsg( ) method. IEEE p1451.0 will provide
the appropriate “commId” so that IEEE 1451.X will know what communication session to use. IEEE p1451.0 will also specify a unique “msgId” so that IEEE 1451.X 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 IEEE p1451.0 when the response has been received.

Optionally, IEEE 1451.X 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 IEEE 1451.X reaches memory or network limits, it should fail on subsequent writeMsg() calls. IEEE p1451.0 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 IEEE p1451.0 invokes the methods that read or write the OctetArray from/to the IEEE 1451.X layer (for example, readMsg(), readRsp(), writeMsg(), and writeRsp()), IEEE p1451.0 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 IEEE 1451.X 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 p1451.0 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 IEEE p1451.0 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.
Figure C.4—Communications state machine for the initiating node
Figure C.5—Communications state machine for the receiving node

Enhancements:
- add shutdown() from states other than idle. This would most likely flow through the aborting state.
- add flow control where the message is received and sent in segments
C.11 Communication sequence

The sequence diagram\footnotemark[1] in Figure C.6 illustrates the typical call sequence between the initiating and receiving node for a “one-way” operation. Information flows from IEEE p1451.0 on the initiating node to IEEE p1451.0 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 IEEE 1451.X.

On the initiating node, IEEE p1451.0 invokes openQoS(), writeMsg(), and close(). On the receiving node, IEEE 1451.X invokes the receiveMsg() method to inform IEEE p1451.0 that a new communication operation has started. IEEE p1451.0 calls back into IEEE 1451.X 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 IEEE p1451.0. IEEE 1451.X 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 IEEE p1451.0 on the initiating node to IEEE p1451.0 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.

\footnotetext[1]{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.}
Figure C.6—Simple communications sequence
Figure C.7—Two way communications sequence
Annex C (Normative)

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:

- 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 (Extensible Markup Language) 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 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 the Calibration Identification TEDS are provided and are based on the information in the corresponding TEDS defined in IEEE 1451.2-1997. IEEE p1451.0 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 Commands TEDS is provided. The schema for the Geographic Location TEDS 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
<?xml version="1.0" encoding="UTF-8"?>
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:schema>
```
<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>
  <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:annotation source="Array of UInt32"/>
</xs:simpleType>

<xs:simpleType name="Float32Array">
<xs:annotation source="Array of Float32"/>
<xs:list itemType="stml:Float32"/>
</xs:simpleType>

<xs:simpleType name="Float64Array">
<xs:annotation source="Array of Float64"/>
<xs:list itemType="stml:Float64"/>
</xs:simpleType>

<xs:simpleType name="OctetArray">
<xs:annotation source="Array of Octet"/>
<xs:list itemType="stml:Octet"/>
</xs:simpleType>

<xs:simpleType name="BooleanArray">
<xs:annotation source="Array of _Boolean"/>
<xs:list itemType="stml:_Boolean"/>
</xs:simpleType>

<xs:simpleType name="StringArray">
<xs:annotation source="Array of _String"/>
<xs:list itemType="stml:_String"/>
</xs:simpleType>

<xs:simpleType name="ErrorCode" final="restriction">
<xs:restriction base="xs:string">
<xs:enumeration value="NO_ERROR"/>
<xs:enumeration value="INVALID_COMMID"/>
<xs:enumeration value="UNKNOWN_DESTID"/>
</xs:restriction>
</xs:simpleType>
<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:simpleType>
</xs:complexType>
<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:restriction>
</xs:simpleType>
<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">
<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="accessLatency" 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>
      <xs:element name="ValueError" type="stml:Boolean" minOccurs="0"/>
      <xs:element name="valueInt8" type="stml:UInt8" minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
<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="valueUint32" 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:documentation>
</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"/>
<xs:element name="seconds" type="stml:UInt8"/>
<xs:element name="amperes" type="stml:UInt8"/>
<xs:element name="kelvins" type="stml:UInt8"/>
</xs:sequence>
</xs:complexType>
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:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
<xs:element name="Value" type="stml:UInt8Array"/>

</xs:sequence>
</xs:complexType>

<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"/>
      <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:sequence>
    </xs:complexType>
  </xs:element>
</xs:element>
<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 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>
<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>
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 Identification TEDS, Calibration-Identification TEDS, Commands TEDS and the Location and Title TEDS and the Geographic Location TEDS.

D.5 Meta-IdentificationTEDS.xsd

```xml
<?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:complexType name="Meta-IdentificationTEDSType">
            <xs:complexContent>
                <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>
    </xs:element>
</xs:schema>
```
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="ManufacturerID" 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>
<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="CalibrationDescription" 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:complexType>
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
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>
          <xs:choice>
            <xs:element name="TagValuePair" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:choice>
                  <xs:element name="TagValuePair" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType mixed="true">
                      <xs:choice>
                        <xs:element name="CommandTEDSDataBlockType">
                          <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>
                              <xs:choice>
                                <xs:element name="TagValuePair" minOccurs="0" maxOccurs="unbounded">
                                  <xs:complexType>
                                    <xs:choice>
                                      <xs:element name="CommandTEDSDataBlockType">
                                        <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>
                                            <xs:choice>
                                              <xs:element name="TagValuePair" minOccurs="0" maxOccurs="unbounded">
                                                <xs:complexType>
                                                  <xs:choice>
                                                    <xs:element name="CommandTEDSDataBlockType">
                                                      <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>
                                                          <xs:choice>
                                                            <xs:element name="TagValuePair" minOccurs="0" maxOccurs="unbounded">
                                                              <xs:complexType>
                                                                <xs:choice>
                                                                  <xs:element name="CommandTEDSDataBlockType">
                                                                    <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>
                                                                        <xs:choice>
                                                                          <xs:element name="TagValuePair" minOccurs="0" maxOccurs="unbounded">
                                                                            <xs:complexType>
                                                                              <xs:choice>
                                                                                <xs:element name="CommandTEDSDataBlockType">
                                                                                  <xs:element name="Name" type="stml:_String"/>
<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"/>
      <xs:element name="timeInstanceArrayValue" type="stml:TimeInstanceArrayType" minOccurs="0"/>
      <xs:element name="timeDurationArrayValue" type="stml:TimeDurationArrayType" minOccurs="0"/>
      <xs:element name="timeValue" minOccurs="0"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
<xs:element name="reveal" minOccurs="0" maxOccurs="65535">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="argName" type="xs:string" use="required"/>
      <xs:element name="argNumber" type="xs:unsignedByte" use="required"/>
      <xs:element name="DataModel" type="stml:DataModelTypes" use="required"/>
      <xs:element name="desc" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
<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:element name="timeValue" type="xs:dateTime"/>
<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 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:complexType>
</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>
  </xs:complexType>
</xs:element>
</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:restriction>
</xs:simpleType>
D.9 LocationAndTitleTEDS.xsd

```xml
<?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">
                <xs:sequence>
                    <xs:element name="LocationAndTitle" maxOccurs="255">
                        <xs:restriction>
                            <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:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:element>
</xs:schema>
```
<?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>
1  </xs:schema>
2
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
<?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>
```

E.3 Example instance document

```xml
<?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/MetaIdentificationTEDSDataBlock.xsd">
  <manufacturerId>Acme Corp.</manufacturerId>
  <ModelNo>1036Z</ModelNo>
</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.
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 F 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
<?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>
```

F.3 Example instance document

```xml
<?xml version="1.0" encoding="UTF-8"?>
<TransducerChannelIdDataBlock
xmlns:stml="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI"
xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://grouper.ieee.org/groups/1451/0/1451HTTPAPI/TransducerChannelIdDataBlock.xsd">
<manufacturerId>Jones Pressure Products</manufacturerId>
<ModelNo>1500AB</ModelNo>
<ChannelDescription>Pressure channel</ChannelDescription>
```
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.
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
<?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="CalibrationDescription" 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>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

G.3 Example instance document

```xml
<?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"/>
```
<CalLabNo>Acurate Calibrations Inc</CalLabNo>
<CalTech>C.P. Lee</CalTech>
<CalibrationDescription>I had difficulty setting the gain on IA 3. Watch for excessive drift on next cal cycle</CalibrationDescription>
<CalDate>08/01/2002</CalDate>
<CalProc>Calibrated per the procedures in D6-46289</CalProc>
<CalStd>Ektron 8200, SN 24698</CalStd>
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 trouble-shooting, the vendor wants to define the following commands:

SetGain   Set gain, PreAmp
AlternateSetGain  Set gain, PreAmp
SetOffset1  Set offset, PreAmp

This is a contrived example with 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 trouble-shooting, calibration or commissioning purposes.

H.3 Schema for the data block
```xml
<?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">
    <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/1451/0/1451HTTPAPI/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>
</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/1451/0/1451HTTPAPI/IEEE1451CommandsTEDSSchema.xsd">

<cmd><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 +5V 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></cmd>

It should be apparent that a generic software tool could be developed to display the set of commands implemented, 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, read and properly display the result(s).
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. Further 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">
              <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
<?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/LocationAndTitleTEDS.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.
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
<?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
<?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/1451/0/1451HTTPAPIUnitsExtensionDataBlock.xsd"
><UnitsExtensionText>of Chlorine</UnitsExtensionText>
</UnitsExtensionDataBlock>
```
Annex K
(informative)

Examples Physical Units

The IEEE 1451.2 Standard 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. IEEE p1451.0 has taken that and expanded on it. This becomes even more necessary if the transducer must plug into any system and be usable 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 or 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 1451.2 these two functions were combined into the units fields in the Channel TEDS. IEEE p1451.0 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 standard 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 upon 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),” edited by B. N. Taylor, National Institute of Standards and Technology, Special Publication 330, 1991 Edition (U.S. Government Printing Office, Washington, DC, August 1991).
Table K.1—SI base units

<table>
<thead>
<tr>
<th>Base quantity</th>
<th>Name</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>meter</td>
<td>m</td>
</tr>
<tr>
<td>mass</td>
<td>kilogram</td>
<td>kg</td>
</tr>
<tr>
<td>time</td>
<td>second</td>
<td>s</td>
</tr>
<tr>
<td>electric current</td>
<td>ampere</td>
<td>A</td>
</tr>
<tr>
<td>thermodynamic temperature</td>
<td>kelvin</td>
<td>K</td>
</tr>
<tr>
<td>amount of substance</td>
<td>mole</td>
<td>mol</td>
</tr>
<tr>
<td>luminous intensity</td>
<td>candela</td>
<td>cd</td>
</tr>
</tbody>
</table>

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 J2 contains examples of derived units expressed in terms of SI base units.
### Table K.2—Units derived from the SI base units

<table>
<thead>
<tr>
<th>Derived quantity</th>
<th>Special name</th>
<th>Special symbol</th>
<th>Expression in terms of other SI units</th>
<th>Expression in terms of SI Base units</th>
</tr>
</thead>
<tbody>
<tr>
<td>plane angle</td>
<td>radian</td>
<td>rad</td>
<td>m m^{-1}=1</td>
<td></td>
</tr>
<tr>
<td>solid angle</td>
<td>steradian</td>
<td>sr</td>
<td>m2 m^{-2}=1</td>
<td></td>
</tr>
<tr>
<td>frequency</td>
<td>hertz</td>
<td>Hz</td>
<td>s^{-1}</td>
<td></td>
</tr>
<tr>
<td>area (square meter)</td>
<td></td>
<td></td>
<td>m2</td>
<td></td>
</tr>
<tr>
<td>volume (cubic meter)</td>
<td></td>
<td></td>
<td>m3</td>
<td></td>
</tr>
<tr>
<td>acceleration (meter per second squared)</td>
<td></td>
<td></td>
<td>m/s^2</td>
<td></td>
</tr>
<tr>
<td>wave number (reciprocal meter)</td>
<td></td>
<td></td>
<td>m^{-1}</td>
<td></td>
</tr>
<tr>
<td>mass density (density) (kilogram per cubic meter)</td>
<td></td>
<td></td>
<td>kg/m^3</td>
<td></td>
</tr>
<tr>
<td>specific volume (cubic meter per kilogram)</td>
<td></td>
<td></td>
<td>m^3/kg</td>
<td></td>
</tr>
<tr>
<td>current density (ampere per square meter)</td>
<td></td>
<td></td>
<td>A/m^2</td>
<td></td>
</tr>
<tr>
<td>magnetic field strength (ampere per meter)</td>
<td></td>
<td></td>
<td>A/m</td>
<td></td>
</tr>
<tr>
<td>amount-of-substance concentration</td>
<td></td>
<td></td>
<td>Mol/m^3</td>
<td></td>
</tr>
<tr>
<td>luminance (candela per square meter)</td>
<td></td>
<td></td>
<td>cd/m^2</td>
<td></td>
</tr>
<tr>
<td>force</td>
<td>newton</td>
<td>N</td>
<td>m kg s^{-2}</td>
<td></td>
</tr>
<tr>
<td>pressure, stress</td>
<td>pascal</td>
<td>Pa</td>
<td>N/m^2</td>
<td>m^{-1}kg s^{-2}</td>
</tr>
<tr>
<td>energy, work, quantity of heat</td>
<td>joule</td>
<td>J</td>
<td>N m</td>
<td>M2 kg s^{-2}</td>
</tr>
<tr>
<td>power, radiant flux</td>
<td>watt</td>
<td>W</td>
<td>J/s</td>
<td>m^2 kg s^{-3}</td>
</tr>
<tr>
<td>electric charge, potential difference,</td>
<td>coulomb</td>
<td>C</td>
<td>s A</td>
<td></td>
</tr>
<tr>
<td>electric conductance</td>
<td>volt</td>
<td>V</td>
<td>W/A</td>
<td>m^2kg s^{-3}A^{-1}</td>
</tr>
<tr>
<td>capacitance</td>
<td>farad</td>
<td>F</td>
<td>C/V</td>
<td>m^{-2} kg^{-1}S^4 A^2</td>
</tr>
<tr>
<td>electric resistance</td>
<td>ohm</td>
<td>Ω</td>
<td>V/A</td>
<td>m^2 kg s^{-3}A^{-2}</td>
</tr>
<tr>
<td>electric conductance</td>
<td>siemens</td>
<td>S</td>
<td>A/V</td>
<td>m^{-2} kg^{-1}A^3</td>
</tr>
<tr>
<td>magnetic flux</td>
<td>weber</td>
<td>Wb</td>
<td>V/s</td>
<td>m^2 kg s^{-2}A^{-1}</td>
</tr>
<tr>
<td>magnetic flux density</td>
<td>tesla</td>
<td>T</td>
<td>Wb/m^2</td>
<td>kg s^{-2}A^{-1}</td>
</tr>
<tr>
<td>inductance</td>
<td>henry</td>
<td>H</td>
<td>Wb/A</td>
<td>m^2 kg s^{-2}A^{-2}</td>
</tr>
<tr>
<td>Celsius temperature</td>
<td>degree Celsius</td>
<td>°C</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>luminous flux</td>
<td>lumen</td>
<td>lm</td>
<td>cd sr</td>
<td></td>
</tr>
<tr>
<td>illuminance</td>
<td>lux</td>
<td>lx</td>
<td>lm/m^2</td>
<td>m^{-2} cd sr</td>
</tr>
</tbody>
</table>

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 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. There are however some devices that 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\(^1\) m\(^{-1}\) or m\(^2\).

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. There are also quantities that 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.

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 ±four 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 1451.2 standard is the octet. The other is that there was no way defined in the IEEE 1451.2 standard 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 1/2 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 1451.2 and IEEE 1451.3, IEEE p1451.0 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 non-zero.

<table>
<thead>
<tr>
<th>Enum</th>
<th>rad</th>
<th>sr</th>
<th>m</th>
<th>kg</th>
<th>s</th>
<th>A</th>
<th>K</th>
<th>mol</th>
<th>cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponent</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>decimal</td>
<td>128</td>
<td>128</td>
<td>130</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

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\(^2\) m the exponent is two.

<table>
<thead>
<tr>
<th>Enum</th>
<th>rad</th>
<th>sr</th>
<th>m</th>
<th>kg</th>
<th>s</th>
<th>A</th>
<th>K</th>
<th>mol</th>
<th>cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponent</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>decimal</td>
<td>128</td>
<td>128</td>
<td>132</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>
Table K.5 shows the table for the measurement of pressure. The derived units for pressure are pascals. This involves three of the base units, the meter, the kilogram and the second.

<table>
<thead>
<tr>
<th>Enum</th>
<th>rad</th>
<th>sr</th>
<th>m</th>
<th>kg</th>
<th>s</th>
<th>A</th>
<th>K</th>
<th>mol</th>
<th>cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponent</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>decimal</td>
<td>128</td>
<td>128</td>
<td>126</td>
<td>130</td>
<td>124</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

The table for the representation of electrical resistance is given in Table K.6. As shown in the table this unit requires four of the base units.

<table>
<thead>
<tr>
<th>Enum</th>
<th>rad</th>
<th>sr</th>
<th>m</th>
<th>kg</th>
<th>s</th>
<th>A</th>
<th>K</th>
<th>mol</th>
<th>cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponent</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>-3</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>decimal</td>
<td>128</td>
<td>128</td>
<td>132</td>
<td>130</td>
<td>122</td>
<td>124</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

As shown in Table K.7 Noise Spectral Density also involves four of the 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>
<thead>
<tr>
<th>Enum</th>
<th>rad</th>
<th>sr</th>
<th>m</th>
<th>kg</th>
<th>s</th>
<th>A</th>
<th>K</th>
<th>mol</th>
<th>cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponent</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>-5/2</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>decimal</td>
<td>128</td>
<td>128</td>
<td>132</td>
<td>130</td>
<td>123</td>
<td>126</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Enum</th>
<th>rad</th>
<th>sr</th>
<th>m</th>
<th>kg</th>
<th>s</th>
<th>A</th>
<th>K</th>
<th>mol</th>
<th>cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponent</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>decimal</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>130</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>Enum</th>
<th>rad</th>
<th>sr</th>
<th>m</th>
<th>kg</th>
<th>s</th>
<th>A</th>
<th>K</th>
<th>mol</th>
<th>cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponent</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>decimal</td>
<td>128</td>
<td>128</td>
<td>130</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

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) \( W = m^2 \text{ kg s}^{-3} \)

<table>
<thead>
<tr>
<th>Enum</th>
<th>rad</th>
<th>sr</th>
<th>m</th>
<th>kg</th>
<th>s</th>
<th>A</th>
<th>K</th>
<th>mol</th>
<th>cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponent</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>decimal</td>
<td>128</td>
<td>128</td>
<td>132</td>
<td>130</td>
<td>122</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

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 to be used to identify the units. The exponents would all be set to 0. Note that for this case some special software will be required to display the appropriate units.

### Table K.11—Counts

<table>
<thead>
<tr>
<th>Enum</th>
<th>rad</th>
<th>sr</th>
<th>m</th>
<th>kg</th>
<th>s</th>
<th>A</th>
<th>K</th>
<th>mol</th>
<th>cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponent</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>decimal</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Enum</th>
<th>rad</th>
<th>sr</th>
<th>m</th>
<th>kg</th>
<th>s</th>
<th>A</th>
<th>K</th>
<th>mol</th>
<th>cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponent</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>decimal</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

### K.2 System Considerations

The discussion above describes how the physical quantity being measured or output can be identified but does not in many cases give the units that the operator wants to see displayed. IEEE p1451.0 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 example consider possible units for pressure as listed in Table K.13. There are many different sets of units 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 and 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>
<thead>
<tr>
<th>Units</th>
<th>Conversion slope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pascals</td>
<td>1</td>
</tr>
<tr>
<td>Kilopascals</td>
<td>0.001</td>
</tr>
<tr>
<td>Pounds per square inch (PSI)</td>
<td>6894.757</td>
</tr>
<tr>
<td>BAR</td>
<td>100,000</td>
</tr>
<tr>
<td>MILLIBAR</td>
<td>100</td>
</tr>
<tr>
<td>Inches of water</td>
<td>249.082</td>
</tr>
<tr>
<td>Millimeters of water</td>
<td>9.80665</td>
</tr>
<tr>
<td>Inches of Mercury</td>
<td>3386.38</td>
</tr>
<tr>
<td>Millimeters of mercury</td>
<td>133.3224</td>
</tr>
<tr>
<td>Atomspheres</td>
<td>101.325</td>
</tr>
<tr>
<td>Kilograms per square centimeter</td>
<td>98066.5</td>
</tr>
<tr>
<td>Torr</td>
<td>133.3224</td>
</tr>
</tbody>
</table>

NOTE—Some of these conversion constants are temperature sensitive and will only be correct at a given temperature.

As we noted above 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>
<thead>
<tr>
<th>Units</th>
<th>Conversion Slope</th>
<th>Conversion Intercept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kelvin</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Celsius</td>
<td>1</td>
<td>273.15</td>
</tr>
<tr>
<td>Fahrenheit</td>
<td>0.5555556</td>
<td>255.3722</td>
</tr>
<tr>
<td>Rankine</td>
<td>0.5555556</td>
<td>0</td>
</tr>
</tbody>
</table>

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 the way that the calibration
coefficients are represented. This means that a transducer that is build 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 Agilent Technical Publications Department [A1]
Annex L

(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 subclauses describe processes that are 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 provides the following information:

- whether or not it is allowable to overwrite an existing TEDS,
- whether or not this TEDS is supported,
- the maximum size of the TEDS, and
- Whether this is a Text-based or binary TEDS.
- whether this TEDS is resident or virtual,
- whether or not the current contents of this TEDS are valid,
- the current size of the TEDS
- the TEDS Checksum
- Whether or not 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.
- After the TEDS data has been transmitted, issue an Update TEDS command to signal that the TEDS data has 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 non-volatile memory before replying to the Update TEDS command.

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 is requested beyond the end of the TEDS.
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 is shown in Figure M.1. This is an optional feature, 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](image)

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.

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, $D_i$ 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 ($t_{pdi}$) 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 $i$ with a time delay, $T_{Di}$, such that $(P_{Di} + t_{pdi} + T_{Di})$ 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 TimInstance sensor**

When the nominal trigger model does not provide the required timing precision, the sensor may be augmented with an embedded TimInstance sensor, as shown below:
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.

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.
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, suggesting that the inter-sample 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:

\[ T_1 = T_{\text{tm}} + t_{\text{pd}} + t_{\text{is}} \]

where

- \( T_{\text{tm}} \) is the time of day at which the trigger message was sent by the NCAP.
- \( t_{\text{pd}} \) is the value of the Incoming propagation delay through the data transport logic field found in the TransducerChannel TEDS.
- \( t_{\text{is}} \) 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.

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

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
TransducerChannel, the manufacturer has chosen to provide a time interval value for each sample in the data-set, suggesting that the inter-sample spacing has some variation. The following calculation may be used:

\[ T(i) = T_{tm} + t_{pd} + t_{is}(i) \]

where

- \( T_{tm} \) is the time of day at which the trigger message was sent by the NCAP.
- \( t_{pd} \) is the value of the Incoming propagation delay through the data transport logic field found in the TransducerChannel TEDS.
- \( t_{is}(i) \) 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.
Annex N(informative)

Notation Summary for IDL

The IEEE p1451.0 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 web site 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 p1451.0 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 p1451.0 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 IEEE p1451.0 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 IEEE p1451.0 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 clause 4.9 of the IEEE p1451.0 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.
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 and 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.
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,
 in Args::UInt32 maxLen,
 out Args::OctetArray payload,
 out Args::_Boolean last );” 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 integer, typically indicating 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.
Annex O (Informative)

TEDS implementation of a simple sensor.

The clauses N.1 through clause N.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 will return a reading whenever queried. The sensor will output in degrees C after corrections have been applied and the operating range is -40 to 80 degrees C ± 2 degrees.

Clause 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 1451.0 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 twelve 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 clause 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 duplicated here).

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>03</td>
<td>Type field for the TEDS Identifier</td>
</tr>
<tr>
<td>Length</td>
<td>04</td>
<td>This field is always set to 04 indicating that the value field contains four octets</td>
</tr>
<tr>
<td>Family</td>
<td>00</td>
<td>This field identifies the member of the IEEE 1451 family of standards that defines this TEDS.</td>
</tr>
<tr>
<td>Class</td>
<td>01</td>
<td>This field identifies the TEDS being accessed. The value (found in Table 17, clause 7.1.1.1) is 01 for the Meta-TEDS.</td>
</tr>
<tr>
<td>Version</td>
<td>01</td>
<td>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 IEEE p1451.0</td>
</tr>
<tr>
<td>Tuple Length</td>
<td>01</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 clause 4.12 and Table O.2. (Table O.2 is a duplicate of Table 4)
Table O.2—Universal unique identification data type structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Number of bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>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 TIM manufacture. 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 meters. Thus the range represented by each 20-bit field is 0 to 1 048 575 arc seconds or 0 to 291 degrees which is sufficient to represent latitude and longitude on the earth’s surface.</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>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 AD. The beginning of the year shall be deemed to start on January 1, 00:00:00, TAI.</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>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 ten-second intervals. In this case, time values greater than one year1 are reserved. The beginning of the year shall be deemed to start on January 1, 00:00:00, TAI. 1There are approximately 31 536 000 seconds per year.</td>
<td>22</td>
</tr>
</tbody>
</table>

Our TIM was produced on Aug 14, 2005 and was the 120th unit produced on that day. The TIM was produced at a manufacturing site located at the geospatial co-ordinates 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 arc seconds.

There is only one manufacturer at this site, so the manufacturer’s field is 0.

1There are approximately 31 536 000 seconds per year.
The Year field value is 2005.

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 = 2240120.

Next, convert the fields to binary and combine them as shown in Table O.3.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Binary Value</th>
<th>Field width (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/S</td>
<td>N</td>
<td>00000000000000</td>
<td>1</td>
</tr>
<tr>
<td>Latitude</td>
<td>145367</td>
<td>000000111000000001111</td>
<td>20</td>
</tr>
<tr>
<td>E/W</td>
<td>W</td>
<td>010111010001000100010</td>
<td>20</td>
</tr>
<tr>
<td>Longitude</td>
<td>381218</td>
<td>010111010001000100010</td>
<td>20</td>
</tr>
<tr>
<td>Manufacturer</td>
<td>0</td>
<td>00000000000000000000000000000000</td>
<td>4</td>
</tr>
<tr>
<td>Year</td>
<td>2005</td>
<td>010111010101010101010</td>
<td>12</td>
</tr>
<tr>
<td>Date/Sequence</td>
<td>2240120</td>
<td>1000100010111001111000</td>
<td>22</td>
</tr>
</tbody>
</table>

Realigning and converting to octets yields the UUID format shown in Table O.4 (shown in hex).

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000001</td>
<td>81</td>
</tr>
<tr>
<td>11000000</td>
<td>C0</td>
</tr>
<tr>
<td>11111001</td>
<td>F9</td>
</tr>
<tr>
<td>01110100</td>
<td>74</td>
</tr>
<tr>
<td>01001000</td>
<td>48</td>
</tr>
<tr>
<td>10000001</td>
<td>81</td>
</tr>
<tr>
<td>11110101</td>
<td>F5</td>
</tr>
<tr>
<td>01100010</td>
<td>62</td>
</tr>
<tr>
<td>00101110</td>
<td>2E</td>
</tr>
<tr>
<td>01110100</td>
<td>78</td>
</tr>
</tbody>
</table>

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:

04 0A 81 C0 F9 74 48 81 F5 62 2E 78

**O.1.3 Operational Timeout**

The operational timeout for this TIM is 0.5 seconds. This is the timeout of normal operations. The number 0.5 will be represented as a F32 floating point number. Converting this into binary yields (see IEEE-754 Std for details):

0 01111110 00000000000000000000000000000000 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:

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
0A 04 3F 00 00 00

O.1.4 Slow access timeout

The slow access timeout is the same as the normal timeout, as there are no slow operations other than self test (covered below). Therefore this optional field is not included.

O.1.5 Self test timeout

The self test timeout for this TIM is 5.0 seconds. 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:

110000000000000000000000

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 clause 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 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 1’s complement. This is described in clause 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

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
<th>Field Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Total Length</td>
</tr>
<tr>
<td>00</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>04</td>
<td>Header</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>03 04 00 01 01 01</td>
</tr>
<tr>
<td>04</td>
<td>0A</td>
<td>UUID</td>
</tr>
<tr>
<td>81</td>
<td>C0</td>
<td>04 0A 81 C0 F9 74 48 81 F5 62 2E 78</td>
</tr>
<tr>
<td>4F</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>2E</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>0A</td>
<td>04</td>
<td>Operational Timeout</td>
</tr>
<tr>
<td>3F</td>
<td>00</td>
<td>0A 04 3F 00 00 00</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>0C</td>
<td>04</td>
<td>Self Test Timeout</td>
</tr>
<tr>
<td>C0</td>
<td>A0</td>
<td>0C 04 C0 A0 00 00</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>0D</td>
<td>02</td>
<td>Number of Channels</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>0D 02 00 01</td>
</tr>
<tr>
<td>F8</td>
<td>82</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

### O.2 TransducerChannel TEDS

The format for the TransducerChannel TEDS is described in clause 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

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>03</td>
<td>Type field for the TEDS Identifier</td>
</tr>
<tr>
<td>Length</td>
<td>04</td>
<td>This field is always set to 04 indicating that the value field contains four octets</td>
</tr>
<tr>
<td>Family</td>
<td>00</td>
<td>This field identifies the member of the IEEE 1451 family of standards that defines this TEDS.</td>
</tr>
<tr>
<td>Class</td>
<td>03</td>
<td>This field identifies the TEDS being accessed. The value, (found in Table 17, clause 7.1.1.1) is 03 for the TransducerChannel TEDS.</td>
</tr>
<tr>
<td>Version</td>
<td>01</td>
<td>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 1451.0</td>
</tr>
<tr>
<td>Tuple</td>
<td>01</td>
<td>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.</td>
</tr>
</tbody>
</table>

Converting this to TLV Octet format yields the following octet pattern
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 C after corrections are applied. The description of units is shown in clause 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 C is handled in the Calibration TEDS. Thus the unit descriptor has the form listed in Table O.7.

### Table O.7—Physical units

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>TLV Field Type</th>
<th>Value</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Physical units interpretation -- see Table 4</td>
<td>50 (32Hex)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$(2 \times \text{&lt;exponent of radians&gt;}) + 128$</td>
<td>51 (33Hex)</td>
<td>128</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>$(2 \times \text{&lt;exponent of steradians&gt;}) + 128$</td>
<td>52 (34Hex)</td>
<td>128</td>
<td>80</td>
</tr>
<tr>
<td>4</td>
<td>$(2 \times \text{&lt;exponent of meters&gt;}) + 128$</td>
<td>53 (35Hex)</td>
<td>128</td>
<td>80</td>
</tr>
<tr>
<td>5</td>
<td>$(2 \times \text{&lt;exponent of kilograms&gt;}) + 128$</td>
<td>54 (36Hex)</td>
<td>128</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>$(2 \times \text{&lt;exponent of seconds&gt;}) + 128$</td>
<td>55 (37Hex)</td>
<td>128</td>
<td>80</td>
</tr>
<tr>
<td>7</td>
<td>$(2 \times \text{&lt;exponent of amperes&gt;}) + 128$</td>
<td>56 (38Hex)</td>
<td>128</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>$(2 \times \text{&lt;exponent of kelvins&gt;}) + 128$</td>
<td>57 (39Hex)</td>
<td>130</td>
<td>82</td>
</tr>
<tr>
<td>9</td>
<td>$(2 \times \text{&lt;exponent of moles&gt;}) + 128$</td>
<td>58 (3AHex)</td>
<td>128</td>
<td>80</td>
</tr>
<tr>
<td>10</td>
<td>$(2 \times \text{&lt;exponent of candelas&gt;}) + 128$</td>
<td>59 (3BHex)</td>
<td>128</td>
<td>80</td>
</tr>
</tbody>
</table>

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 degrees C. 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-754 Std for details):

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
01000110 110100100000000000000000 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 degrees C. 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-754 Std for details):

0 10000111 0110000100000000 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 following 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 degrees C. 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 000000000000000000000000 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

**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 multi-range capability, so this field is omitted.

**O.2.10 Sample definition**

Sample definition consists of 3 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 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/second. 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, yielding a final octet pattern of
14 04 3D CC CC CD

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, yielding 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 seconds. The field is F32 and the field code is 23, yielding a final octet pattern of
17 04 3D CC CC CD

O.2.17 TransducerChannel warm-up time
This transducer will take 30 seconds to warm up to specs. The field code is 24 and the parameter type is F32, yielding a final octet pattern of
18 04 41 F0 00 00

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
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, yielding 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 seconds. The field code is 26 and the format is F32, yielding 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 un-triggered, 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 un-triggered 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 = 48 and the format is U8E, yielding a final octet pattern of

30 01 02

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 1 buffer of 1 reading in length. Thus the value of this field is 0 and may be omitted.

O.2.24 End of dataset operation

This field is only applicable to actuators and is not included in this case.

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
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:

- ID: 03 04 00 03 01 01
- Cal key: 0A 01 01
- Type key: 0B 01 00
- Units: 0C 06 32 01 00 39 01 82
- Lower limit: 0D 04 43 69 00 00
- Upper limit: 0E 04 43 B0 80 00
- Uncertainty: 0F 04 40 00 00 00
- Self test: 10 01 01
- Sample definition: 12 09 28 01 00 29 01 02 30 01 0C
- Update time: 14 04 3D CC CC CD
- Read setup time: 16 04 37 D1 B7 17
- Sampling period: 17 04 3D CC CC CD
- Warmup time: 18 04 41 F0 00 00
- Read delay: 19 04 37 D1 B7 17
- Self test time: 1A 04 40 A0 00 00
- Sampling: 1F 03 31 01 02

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.

O.3 Calibration TEDS

The TIM contains a single thermistor that is providing a temperature reading. This thermistor is placed into a voltage divider circuit and fed a 10 volt signal into a 10 volt 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 volt range, or 409.6 counts/volt. 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 C, so it must be converted to Kelvins to show proper SI units. Since we wish to output in degrees C, we will adjust this in the SI units clause.

We can use the simplified form of calibration shown in clause 8.6.3.10.

The Calibration TEDS for this TIM will contain the following fields:
O.3.1 Calibration TEDS Identification

Table O.8 lists the contents for the identification field. (from Table 41)

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>03</td>
<td>Type field for the TEDS Identifier</td>
</tr>
<tr>
<td>Length</td>
<td>04</td>
<td>This field is always set to 04 indicating that the value field contains four octets</td>
</tr>
<tr>
<td>Family</td>
<td>00</td>
<td>This field identifies the member of the IEEE 1451 family of standards that defines this TEDS.</td>
</tr>
<tr>
<td>Class</td>
<td>05</td>
<td>This field identifies the TEDS being accessed. The value, (found in Table 17, clause 7.1.1.1) is 03 for the Calibration TEDS.</td>
</tr>
<tr>
<td>Version</td>
<td>01</td>
<td>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 1451.0</td>
</tr>
<tr>
<td>Tuple</td>
<td>01</td>
<td>This field gives the number of octets in the length field of all tuples in the TEDS except this tuple.</td>
</tr>
</tbody>
</table>

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 Aug 14, 2005 and the time was 14:00:00. Placing this into the Time format described in clause 4.15. yields a total number of seconds since Jan 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, yielding a final octet pattern of:

0A 04 43 00 9F E0 00 00 00 00

O.3.3 Calibration interval

This TIM has a calibration interval of 1 year. Converting to seconds yields 365*24*60*60 = 31536000 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, yielding 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 is going to output in degrees C, 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 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:

Copyright © 2007 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
1. **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

   0C 06 1F 04 43 88 93 33

2. **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.

3. **O.3.6 Pre and post conversion operations**

   No pre or post conversion operations are required. Therefore these fields are omitted.

4. **O.3.7 Linear conversion method**

   This TEDS will be using the linear conversion method, that 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 clause 8.6.3.20, but will omit both the data channel and channel key as there is only 1 channel that only returns raw a/d counts.

5. **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 clause 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.

   The field type is 51 and the combined length is 8, yielding a final octet pattern of

   33 08 43 9C 28 F6 44 7D 5B 85

6. **O.3.7.2 Final linear conversion octet format**

   The field code is 20 and the total length of the subfield is 10 yielding a final octet format of

   14 0A 33 08 43 9C 28 F6 44 7D 5B 85

7. **O.3.8 Final Calibration TEDS format**

   None of the other fields apply to a simple linear regression calibration, so they are omitted. The fields included are:

   03 04 00 05 01 01 (header)
   0A 08 43 00 0F 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 00 03 04 00 05 01 01
   0A 08 43 00 0F 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
**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 is used (from Table 41)

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>03</td>
<td>Type field for the TEDS Identifier</td>
</tr>
<tr>
<td>Length</td>
<td>04</td>
<td>This field is always set to 04 indicating that the value field contains four octets</td>
</tr>
<tr>
<td>Family</td>
<td>00</td>
<td>This field identifies the member of the IEEE 1451 family of standards that defines this TEDS.</td>
</tr>
<tr>
<td>Class</td>
<td>12</td>
<td>This field identifies the TEDS being accessed. The value, (found in Table 17, clause 7.1.1.1) is 12 for the User’s Transducer Name TEDS.</td>
</tr>
<tr>
<td>Version</td>
<td>01</td>
<td>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 1451.0</td>
</tr>
<tr>
<td>Tuple</td>
<td>01</td>
<td>This field gives the number of octets in the length field of all tuples in the TEDS except this tuple.</td>
</tr>
</tbody>
</table>

Converting this to TLV Octet format yields the following octet pattern

03 04 00 0C 01 01

**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, 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:

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, yielding a final octet pattern for this TEDS of:

00 00 00 13 03 04 00 0C 01 01
O.5 Required commands

This TIM will implement the minimum command set required to operate as a legal 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 of clause 7.1.1.

Table O.10 reproduces most of Table 16.

<table>
<thead>
<tr>
<th>Function</th>
<th>Command</th>
<th>State</th>
<th>TIM</th>
<th>Reply Expected</th>
<th>Required Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>Query TEDS</td>
<td>Any</td>
<td>Active</td>
<td>Yes</td>
<td>Required</td>
</tr>
<tr>
<td>2</td>
<td>Read TEDS segment</td>
<td>Any</td>
<td>Active</td>
<td>Yes</td>
<td>Required</td>
</tr>
<tr>
<td>3</td>
<td>Write TEDS segment</td>
<td>Any</td>
<td>Active</td>
<td>No</td>
<td>Required</td>
</tr>
<tr>
<td>4</td>
<td>Update TEDS</td>
<td>Any</td>
<td>Active</td>
<td>Yes</td>
<td>Required</td>
</tr>
<tr>
<td>5</td>
<td>Run self-test</td>
<td>Idle</td>
<td>Active</td>
<td>No</td>
<td>Required</td>
</tr>
<tr>
<td>6</td>
<td>Write service request mask</td>
<td>Any</td>
<td>Active</td>
<td>No</td>
<td>Required</td>
</tr>
<tr>
<td>7</td>
<td>Read service request mask</td>
<td>Any</td>
<td>Active</td>
<td>Yes</td>
<td>Required</td>
</tr>
<tr>
<td>8</td>
<td>Read status-event register</td>
<td>Any</td>
<td>Active</td>
<td>Yes</td>
<td>Required</td>
</tr>
<tr>
<td>9</td>
<td>Read status-condition register</td>
<td>Any</td>
<td>Active</td>
<td>Yes</td>
<td>Required</td>
</tr>
<tr>
<td>10</td>
<td>Clear status-event register</td>
<td>Any</td>
<td>Active</td>
<td>No</td>
<td>Required</td>
</tr>
<tr>
<td>11</td>
<td>Write status-event protocol state</td>
<td>Idle</td>
<td>Active</td>
<td>No</td>
<td>Required</td>
</tr>
<tr>
<td>12</td>
<td>Read status-event protocol state</td>
<td>Any</td>
<td>Active</td>
<td>Yes</td>
<td>Required</td>
</tr>
</tbody>
</table>

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 0’s 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 TEDSand User’s Transducer Name TEDS shall return attributes with the read only bit cleared. All other fields will be implemented as described in clause 7.1.1.1. Note that only channel 1 will be allowed when addressing the TransducerChannel TEDS and Calibration TEDS and 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 clauses 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 request mask.

These commands will read or set the service request mask for either the entire TIM or for 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 a read data is sent to Channel 1, an acquisition will take place before data is 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 Exhauusted, Not First Read of Data-set.

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 the idle state commands that are implemented in this TIM.

<table>
<thead>
<tr>
<th>Function</th>
<th>Command</th>
<th>Address Group definition</th>
<th>Address TransducerChannel</th>
<th>Proxy Group</th>
<th>Reply Expected</th>
<th>Required Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>AddressGroup definition</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Required</td>
</tr>
<tr>
<td>10</td>
<td>Calibrate TransducerChannel</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Command</th>
<th>Address</th>
<th>Proxy</th>
<th>Group</th>
<th>Reply</th>
<th>Expected</th>
<th>Required</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TransducerChannel</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>See note</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>data-set segment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Trigger command</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Required</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 return the data in response to the command.

O.5.3.2 Trigger

This command is required, but will be treated as a NOP.

O.5.4 Transducer idle or operating state commands

The commands listed in Table O.13 will be implemented as described in clause 7.1.4, except for the Read TransducerChannel trigger state.

Table O.13—Transducer idle or operating state commands

<table>
<thead>
<tr>
<th>Function</th>
<th>Command</th>
<th>Address</th>
<th>Proxy</th>
<th>Group</th>
<th>Reply Expected</th>
<th>Required</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TransducerChannel</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Operate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Idle</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Read TransducerChannel trigger state</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Read AddressGroup assignment</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Required</td>
<td></td>
</tr>
</tbody>
</table>
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 clause 7.1.5.1

Table O.14—TIM sleep state commands

<table>
<thead>
<tr>
<th>Function</th>
<th>Command</th>
<th>Address</th>
<th>Required</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Wake-up</td>
<td>Yes</td>
<td>No</td>
<td>Optional</td>
</tr>
</tbody>
</table>

O.5.6 TIM active state commands

The active state commands listed in Table O.15 will be implemented as described in clause 7.1.6

Table O.15—TIM active state commands

<table>
<thead>
<tr>
<th>Function</th>
<th>Command</th>
<th>Address</th>
<th>Required</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read TIM version</td>
<td>Yes</td>
<td>Yes</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>TIM Sleep</td>
<td>Yes</td>
<td>No</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Store operational Setup</td>
<td>Yes</td>
<td>No</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Recall operational Setup</td>
<td>Yes</td>
<td>No</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Read IEEE p1451.0 Version</td>
<td>Yes</td>
<td>Yes</td>
<td>Required</td>
</tr>
</tbody>
</table>

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.

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 p1451.0 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>
<thead>
<tr>
<th>Function</th>
<th>Command</th>
<th>Address</th>
<th>TIM</th>
<th>Global</th>
<th>Reply Expected</th>
<th>Required</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reset</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Optional</td>
<td></td>
</tr>
</tbody>
</table>

O.5.7.1  Reset

This command is implemented and causes the initialization sequence to occur.