TEST SPECIFICATION

HART®
COMMUNICATION PROTOCOL

Slave Common Practice Command Test Specification

HCF_TEST-004, Revision 4.0

Release Date: 12 March 2009

Release Date: 12 March 2009

Document Distribution / Maintenance Control / Document Approval

To obtain information concerning document distribution control, maintenance control, and document approval please contact the HART Communication Foundation (HCF) at the address shown below.

Trademark Information

HART® is a registered trademark of the HART Communication Foundation, Austin, Texas, USA. Any use of the term HART hereafter in this document, or in any document referenced by this document, implies the registered trademark. WirelessHART™ is a trademark of the HART Communication Foundation. All other trademarks used in this or referenced documents are trademarks of their respective companies. For more information contact the HCF Staff at the address below.

Attention: Foundation Director
HART Communication Foundation
9390 Research Boulevard
Suite I-350
Austin, TX 78759, USA
Voice: (512) 794-0369
FAX: (512) 794-3904

http://www.hartcomm.org

Intellectual Property Rights

The HCF does not knowingly use or incorporate any information or data into the HART Protocol Standards which the HCF does not own or have lawful rights to use. Should the HCF receive any notification regarding the existence of any conflicting Private IPR, the HCF will review the disclosure and either (a) determine there is no conflict; (b) resolve the conflict with the IPR owner; or (c) modify the standard to remove the conflicting requirement. In no case does the HCF encourage implementers to infringe on any individual's or organization's IPR.

Table of Contents

Index to Tables

Preface

This preface is included for informational purposes only.

This Test Specification is a companion to Revision 9.0 of the *Common Practice Command Specification*. The principal change to this version of the test specification is to provide support for HART 7. Changes to this document include:

- CAL000 Checks for Common Practice Commands: Added support for 16-bit command numbers. Added support (response codes and byte count) for commands introduced and modified by HART 7.

- CAL001 Verify Write Protect: Added support for HART 7 write commands 77, 87, 88, 89, 92, 97, 99, 102, 103, 104, 106, 108, 109, 116, 117, 118, and 513.

- CAL033 Read Device Variables: Added support for the device variable codes required in HART 7 and later devices.

- CAL042 Perform Device Reset: Added support for the HART 7 command 0 Byte Count.

- CAL051 Write Dynamic Variable Assignments: Added support for the device variable codes required in HART 7 and later devices.

- CAL054 Read Device Variable Information: Added support for "update time period" returned in command 54 responses from HART 7 and later devices.

- CAL071 Lock Device: Added support for wireless products and the additional lock codes introduced in HART 7.

- CAL074 Verify I/O System Commands. Added support for HART 7 commands 77, 84-88, and 94. The original test is now Test Case A. The HART 7 and later functionality was added by creating additional testcases. Test Case A verifies the basic I/O system and device functionality. Test Case B specifically tests features supported by HART 7 and later I/O systems. Test Case C verifies the changing of master address in HART 7 and later I/O systems. Test Case D is focused on the I/O System statistics in HART 7 and later products.

- CAL078 Command Aggregation: New test created to support HART 7 command 78, which allows a single request response transaction to include multiple commands.

- CAL091 Trending: New test to verify the configuration and execution of trends using HART 7 command 91, 92, and 93.

- CAL101 I/O Subsystem Burst Mode: New test to verify the optional behavior introduced in HART 7 that allows an I/O sub-system, multiplexor, or wireless adapter to support a burst message even if the sub-device does not support burst mode. HART 7 commands 101 and 102 are tested in addition to using the burst mode functionality verified in the DLL tests

- CAL115 Event Notification: New test cases to verify the optional behavior introduced in HART 7 that allows publication of status information independent of data publication. TestCase A and B are for all product types. Test Case C is for profiles that support sub-devices (I/O sub-systems, wireless adapters, and multiplexors). Test Case A tests the fundamental requirements of event notification. Test Case B tests the queueing of events for multiple events. TestCase C focuses on products that support sub-devices. HART 7 commands 115, 116, 117, 118, and 119 are tested in addition to using the burst mode functionality verified in the DLL tests.

- CAL512 Country Code: New test to verify operation of the optional HART 7 commands 512 and 513. The commands support the reading and writing of the country code used to determine the intended installation locale.

In addition, the entire document was reviewed for consistency with HART 7 requirements and updated accordingly. A number of minor modifications resulted from this review.

Introduction

Common Practice Commands, while optional, provide standardized commands applicable to a wide range of Field Devices. In fact, statistics indicate that over 12 Common Practice Commands are used by close to 90% of all HART compatible Field Devices (e.g., Commands 33-35, 38, 40-42, 44-46, 48, 59). HART 6 and 7 add several Common Practice Commands, and the number of Common Practice Commands supported by Field Devices and Host Applications have increased.

The HCF is committed to ensuring that the Field Devices and Host Applications can successfully utilize the capabilities found in Common Practice Commands. Supplying Test Specifications to developers supports this goal.

This test specification extends support for the HART Application Layer in the HCF QA Program to the Common Practice Commands. Clear testing requirements are provided for the requirements found in Protocol Specifications. These Test Specifications:

- Provide clear test requirements. The Test Specifications reduce the number of the Test Plans that must be developed by the manufacturer.
- Can be used early in the development effort to informally verify functionality as it is implemented.
- Must be completed along with the Test Report (HCF_PROC-12, and HCF_FRM-119) prior to product release and product registration with the HCF.
- Are useful parts of a regression testing program as the Field Device is maintained and enhanced.
- Clarify ambiguities in the Protocol. Since this specification is balloted and approved like all other HART Specifications, it is equally binding.

This document defines tests for HART Common Practice Command requirements. Common Practice Command Tests (generally) become useful later in the development life-cycle. Frequently, this will occur after Physical Layer, Data Link Layer, and Universal Command testing is complete.

The Common Practice Command tests can be classified as follows:

- **Data Link Layer Commands**. These commands support the establishment of a communication connection between the Master and the Field Device. For instance, command 59 modifies the FSK preamble length.
- **Primary Variable Range Commands**. These commands allow the relationship between the analog signal and the Primary Variable digital value to be defined.
- **Loop Current Support**. These are a series of tests supporting the forcing or calibration of the Loop Current.
- **Device Management Commands**. These commands support routine device management functions, like forcing a self-test or performing a device reset.

- **Transducer Trim Commands**. These commands allow the adjustment or "trim" of a Device Variable.
- **Mapping Process Variable Commands**. These commands allow the user to view and adjust the mapping of the connection between Device Variables and Dynamic Variables.
- **Primary Variable Commands**. These commands support the configuration of the Primary Variable.
- **Burst Mode Commands**. These commands are necessary for the publishing of cyclical process data using "Burst" messaging. In this mode, a device is instructed to publish the response to a command continuously without any further Master or Host action.
- **Event Notification Commands**. These commands configure a device to publish changes in the device's status, independently from data publication. The implementation of Event Notification requires Burst mode implementation.
- **Data Trending Commands**. Trending commands enable the collection of monotonically spaced data samples for a specified Device Variable to be acquired. The implementation of Trending
- **I/O System and Sub-Device Commands**. Theses commands enable communication to multiple devices via an intermediate Bridging Device or I/O System.
- **Synchronized Device Action Commands**. Synchronous Action commands are used to defer a device activity or action to a specified, future time. The actions could be measurements or other operations performed by multiple devices in a synchronized manner.
- Analog Channel Commands. These commands provide standardized support for Field Devices supporting more than one analog connection to the host system. Tests are performed on each channel for all Analog Channel commands.

Like the Common Practice Commands themselves, there is some coupling between the different tests. For example, any Device Variable damping writes (Command 55) are confirmed by reading the Device Variable information (Command 54). However, the tests are designed to be as independent as possible. Section 5.1 lists the recommended order of testing to further simplify the testing of Common Practice Commands.

For each command implemented, manufacturers of Field Devices are expected to execute and pass all of the corresponding conformance tests.

1. SCOPE

This document defines conformance tests for the HART Common Practice Commands. Developers of Field Devices must use these conformance tests to improve the probability of conformance for their device to the HART Common Practice Command Specification. Field Devices must successfully complete all applicable tests in this document. In other words, if a Common Practice Command is implemented in a Field Device, then its implementation must pass the corresponding tests in this Specification. Common Practice Commands that are not implemented are not tested.

1.1 Features Tested

All major Common Practice Command features in the DUT are tested. This includes:

- Tests to provoke every allowed Response Code;
- Verification of data written to the DUT;
- Proper application of "Busy" and Delayed Response;
- Support for both transmitters and actuators;
- Detailed tests for more than 60 Common Practice Commands.

1.2 Features Not Tested

Some features of the Common Practice Command Specification are not tested including:

- Commands that are not recommended (39, 57, 58, 61, 110)
- Commands tested in Data Link Layer Tests (59, 103-105, 107-109)
- Trim Commands are not fully tested (e.g., "Applied Process Too High", "Applied Process Too Low" in Command 82)
- Tests have not been developed for Commands 106, 113, 114
- Block Transfer Commands (111, 112)
- Proper conversion of DUT values are not confirmed when Engineering Unit Codes are modified.

Compliance with all Protocol Requirements is mandatory. The developers must confirm that their devices meet all Protocol requirements, including those features not tested by this specification.

2. REFERENCES

2.1 The HART-Field Communications Protocol Specifications

These documents published by the HART Communication Foundation are referenced throughout this specification:

Token-Passing Data Link Layer Specification, HCF_SPEC-81

Command Summary Specification, HCF_SPEC-99

Universal Command Specification, HCF_SPEC-127

Common Practice Command Specification, HCF_SPEC-151

Common Tables, HCF_SPEC-183

Note: While HART 7 specifications are referenced throughout this test specification, HART 5 and 6 Specifications applicable to the same commands are also supported.

2.2 Other HCF Documents

The following documents describe the procedure for demonstrating HART Compliance and register the device with the HCF. All devices claiming HART Compliance must be submitted to the HCF for compliance verification and registration.

HCF Quality Assurance Program. HCF_PROC-12

Device Registration Form. HCF_FRM-110

2.3 Related Documents

The following documents provide guidance and background information used in developing this Test Specification:

IEEE Standard for Software Test Documentation, ANSI/IEEE Std 829

IEEE Standard for Software Unit Testing, ANSI/IEEE Std 1008

3. DEFINITIONS

Definitions for terms can be found in the *HART Communications Protocol Specification.* Terms used in this document include: ASCII, Broadcast Address, Busy, Data Link Layer, Delayed Response, Delayed Response Mechanism, Device Reset, Device Variable, Dynamic Variable, Fixed Current Mode, Floating Point, ISO Latin-1, Master, Multi-drop, Not-A-Number, Packed ASCII, Preamble, Request Data Bytes, Response Data Bytes, Response Message, Slave, Slave Time-Out, Software Revision Level, Time Constant, Units Code.

Some other terms used only within the context of the *Common Practice Command, Test Specification* are:

BYTE_COUNT Refers to the value contained in the Byte Count Field of the DUT response.

COMMUNICATIONS_ ERROR Indicates that communication itself was unsuccessful. In other words, there was no response or the DUT detected a communications error (see the *Command Summary Specification*).

Primitive Test A Test designed to verify conformance with a narrowly focused set of requirements found in the HART Field Communications Protocol (see Test). Each Primitive Test consists of both Test Case(s) and the corresponding Test Procedure(s).

RESPONSE_CODE When communication is successful (from a Data Link Layer viewpoint) a slave indicates the correctness of the master response using this byte (see the *Command Summary Specification*).

Test A set of one or more Test Cases and Test Procedures.

Test Case A narrowly focused set of conditions, inputs and expected outputs designed to verify proper operation of the DUT.

Test Procedure A sequence of steps or actions designed to fully execute a Test Case.

4. SYMBOLS/ABBREVIATIONS

DUT	**D**evice **U**nder **T**est
HCF	**H**ART **C**ommunication **F**oundation
LEP	**L**ower **E**nd **P**oint
LRV	**L**ower **R**ange **V**alue
LTL	**L**ower **T**ransducer **L**imit
MRV	**M**ost **R**ecent **V**alue
SOM	**S**tart **O**f **M**essage
UEP	**U**pper **E**nd **P**oint
URV	**U**pper **R**ange **V**alue
UTL	**U**ppe**r** **T**ransducer **L**imit

5. APPROACH

This Test Specification uses a "black box" approach to testing compliance with Common Practice Command requirements. Testing is decomposed into a series of narrowly focused Tests, each containing one or more test cases and test procedures.

- Each test is described in a narrative form, in some cases, with the assistance of tables containing test vectors.
- The test procedures are described using pseudo code.
- Within each test procedure termination points are uniquely numbered. This allows cross-referencing should the DUT fail a test.

The tests should be performed in the sequence shown in Section 5.1.

Common Practice Commands are optional. As a result, the test procedures verify support for any optional requirements prior to the main body of the test. If the Field Device response is "Command Not Implemented" then the test procedure aborts the test. An abort of a test simply indicates the test is not applicable. However, if Common Practice Command is supported, then the Field Device must meet Protocol requirements exactly as specified. The test will proceed to assess compliance.

Note: Some tests depend on the DUT supporting a companion command to the Common Practice Command under test. For example, Command 56 (Write Device Variable Transducer Serial Number) requires Command 54 (Read Device Variable Information) to be supported (e.g., to read the value actually stored in the DUT).

5.1 Testing Sequence

Since each test verifies compliance with a specific Common Practice Command requirement, some tests depend on successful completion of other tests. As a result, these dependencies require the tests to be completed in a certain order. The following table shows the recommended order of testing.

Table 1 Test Execution Sequence

No.	Test
1	CAL000
2	CAL041
3	CAL042
4	CAL048
5	CAL034
6	CAL044
7	CAL049
8	CAL040
9	CAL035
10	CAL047
11	CAL033
12	CAL054
13	CAL053
14	CAL055
15	CAL056
16	CAL050
17	CAL051
18	CAL060
19	CAL063
20	CAL070
21	CAL071
22	CAL072
23	CAL064
24	CAL065
25	CAL066
26	CAL068
27	CAL069
28	CAL062
29	CAL074
30	CAL079
31	CAL080
32	CAL107
33	CAL036
34	CAL037
35	CAL038
36	CAL043
37	CAL045
38	CAL046
39	CAL052
40	CAL067
41	CAL073
42	CAL001
43	CAL078
44	CAL091
45	CAL114
46	CAL115
47	CAL512

5.2 Conventions

Throughout the Test Definitions, some conventions are used. The most common are references to the command status bytes (i.e., Communication Status, Field Device Status, and Response Codes). References to these data are explained in the following sections. In addition, angle brackets are often included in test vectors or the pseudo code. Text shown in italics between < > brackets is to be replaced by the corresponding data for the DUT.

5.2.1 Communication Errors

A "`COMMUNICATIONS_ERROR`" consists of one or more of the following error indications (see *Command Summary Specification*):

- No Response
- Vertical Parity Error (i.e., Parity Error)
- Longitudinal Parity Error (i.e., Bad Check Byte)
- Framing Error
- Overrun Error
- Buffer Overflow

5.2.2 Response Code

"RESPONSE_CODE" indicates whether a Slave Device considered the master request valid or not. Common Response Codes used in this document include (see *Command Response Code Specification*):

Table 2 Common Response Codes

Slave Indication	Code No.
"Success"	0
"Invalid Selection"	2
"Too Few Data Bytes Received"	5
"In Write Protect"	7
"Update Failure"	8
"Busy"	32
"Delayed Response Initiated"	33
"Delayed Response Running"	34
"Command Not Implemented"	64

5.2.3 Device Status

A "DEVICE_STATUS" consists of one or more of the following status indications (see *Command Summary Specification*):

- Device Malfunction (bit 7)
- Configuration Changed (bit 6)
- Cold Start (bit 5)
- More Status Available (bit 4)
- Loop Current Fixed (bit 3)
- Loop Current Saturated (bit 2)
- Non-PV Out of Limits (bit 1)
- PV Out of Limits (bit 0)

5.3 Comparing Floating-Point Numbers

Floating point numbers are widely used in Common Practice Commands. As a result, the values of floating point numbers are often compared. To ensure consistent results, actual testing must use a small "delta" when comparing floating point numbers. Delta must be chosen to be as large as the value of a few (3-6) least significant bits of the fractional part of the floating-point number. For equalities, this delta establishes a dead-band. For inequalities, the dead-band is a small offset to the benefit of the DUT.

6. DELIVERABLES

The Test Report included in ANNEX B shall be completed for each Field Device tested. This Test Report is a simple check list indicating:

- Who performed the tests;
- What Field Device was used for testing;
- When the testing was completed; and
- The completion status for each test.

In addition, some tests require additional data to be recorded (e.g., listing the Device Variables found during testing). This data must be attached to the Test Report.

The Test Report provides: a record of the testing; will satisfy most Quality Assurance Audits, and provides sufficient detail to allow the test results to be reproduced. The Test Report must be included with the registration of the manufacturer's Field Device with the HCF.

> Note: Registration of devices is not limited to delivering the completed test report. Other supporting materials and documentation must also be provided as indicated by HCF procedures and policies set forth in the *HCF Quality Assurance and Device Registration Procedure*

7. TEST DEFINITIONS

7.1 CAL000 Check for Common Practice Commands

This test scans the Common Practice Commands Numbers and verifies basic command operation as follows:

- All commands must be answered.
- Commands that are not implemented in the Field Device must be answered "Command not Implemented".
- Undefined Common Practice Commands must be answered "Command not Implemented".
- All Commands are dispatched with no data bytes. Commands requiring request data bytes must be answered "Too Few Data Bytes Received".
- All responses must contain the same command number as sent, valid response data, and the appropriate Response Code.

The "Set Commands" are not scanned to prevent accidentally creating grossly erroneous configurations. In other words, only Commands 33-35, 40, 44-47, 49-51, 53-56, 59, 60, 62-72, 74-82, and 84-109, 113-127 are scanned. See the test vectors in Table 3.

Table 3 Test Vectors for Scan of Common Practice Commands

Command (Cmd)	Response Codes (RC)	Byte Count (BC)
33	5, 64	2
34	5, 64	2
35	5, 64	2
40	5, 64	2
44	5, 64	2
45	5, 64	2
46	5, 64	2
47	5, 64	2
49	5, 64	2
50	0, 64	6, 2
51	5, 64	2
52	5, 64	2
53	5, 64	2
54	5, 64	2
55	5, 64	2
56	5, 64	2
59	5, 64	2
60	5, 64	2
61	0, 64	26, 2 (Note 1)
62	5, 64	2
63	5, 64	2
64	5, 64	2
65	5, 64	2
66	5, 64	2
67	5, 64	2
68	5, 64	2
69	5, 64	2
70	5, 64	2
(H5)	0, 64	2,12
71	5, 64	2
(H5)	64	2
72	0, 9, 64	2
(H5)	64	2
74	0, 64	10, 2
(H5)	64	2
75	5, 64	2
(H5)	64	2
76	0, 64	3, 2
(H5)	64	2
77	64	2
78	5, 64	2, 2
(H5,6)	64	2
79	5, 64	2
(H5)	64	2
80	5, 64	2
(H5)	64	2
81	5, 64	2
(H5)	64	2
82	5, 64	2
(H5)	64	2

Command (Cmd)	Response Codes (RC)	Byte Count (BC)
84	5,64	2
(H5,6)	64	2
85	5, 64	2
(H5,6)	64	2
86	5, 64	2
(H5,6)	64	2
87	5, 64	2
(H5,6)	64	2
88	5, 64	2
(H5,6)	64	2
89	5, 64	2
(H5,6)	64	2
90	0, 64	17, 2
(H5,6)	64	2
91	5, 64	2
(H5,6)	64	2
92	5, 64	2
(H5,6)	64	2
93	5, 64	2
(H5,6)	64	2
94	0, 64	18, 2
(H5,6)	64	2
95	0, 64	8, 2
(H5,6)	64	2
96	5, 64	2
(H5,6)	64	2
97	5, 64	2
(H5,6)	64	2
98	5, 64	2
(H5,6)	64	2
99	5, 64	2
(H5,6)	64	2
100	64	2
101	5, 64	2
(H5,6)	64	2
102	5, 64	2
(H5,6)	64	2

Command (Cmd)	Response Codes (RC)	Byte Count (BC)
103	5, 64	2
(H5,6)	64	2
104	5, 64	2
(H5,6)	64	2
105	0, 64	29, 2
(H6)	0, 64	8, 2
(H5)	64	2
106	0, 64	2
(H5)	64	2
107	5, 64	2
108	5, 64	2
109	5, 64	2
110	0, 64	22, 2
113	5, 64	2
114	5, 64	2
115	5, 64	2
(H5,6)	64	2
116	5, 64	2
(H5,6)	64	2
117	5, 64	2
(H5,6)	64	2
118	5, 64	2
(H5,6)	64	2
119	5, 64	2
(H5,6)	64	2
120	64	2
121	64	2
122	0, 5, 64	X,2,2(Note 2)
123	0, 5, 64	X,2,2(Note 2)
124	0, 5, 64	X,2,2(Note 2)
125	0, 5, 64	X,2,2(Note 2)
126	0, 5, 64	X,2,2(Note 2)
127	64	2
512	0, 64	6, 2
(H5,6)	64	2
513	5, 64	2
(H5,6)	64	2
514 – 767	64	2

Notes:

1 Commands that are "Not Recommended" for use should not be supported unless unless they are necessary to meet the requirements in Section 6 of the *Command Summary Specification* (i.e., to maintain backward compatibility with a previous revision of the device). Any "Not Recommended" command being supported will be noted and a warning printed.

2 Factory Only Commands should be protected. If the commands are not implemented, the response will be "Command Not Implemented". If the commands are read commands or command commands, the response is device specific. If the command is accessible and is a write command, the response will be "Too Few Data Bytes Received".

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	All

Test Procedure

Initialize response table. This table stores the correct Response Code and Byte Count for each Common Practice Command when messaged with zero data bytes.

```
      CALL IdentifyDevice
```

Sequentially send Commands in Table 3 with zero data bytes.

```
      FOR each TEST_CASE in Table 3
            SEND Cmd with zero data bytes
            IF there is no response
                  THEN Test result is FAIL                         (8000 + TEST_CASE)
            END IF
            CALL TestValidFrame()
            IF (RESPONSE_CODE is not in list)
                  THEN Test Result is FAIL                         (9000 + TEST_CASE)
            END IF
            IF (BYTE_COUNT is not in list)
                  THEN Test Result is FAIL                        (10000 + TEST_CASE)
            END IF
      FOR END

      END TEST
```

7.2 CAL001 Verify Write Protect

Verifies support for write protect. If device responds to Command 15 with Write Protect as "None"(251), then this test is aborted. Otherwise, the test verifies that the Field Device is in Write Protect and then tests every Set and Write command.

Note: Devices are implicitly expected to check for command implemented, then write protect.

As shown in the following three tables, the Write Protect Tests are divided into three categories :

- Simple commands (see Table 4) that require no special parameters and only affect a single property;
- Device Variable commands (see Table 6) that may not be supported for all Device Variables (e.g., not all Device Variables have a Transducer Serial Number property); and
- Analog Channel commands (see Table 5) that may not be applicable to every Analog Channel (e.g., Write Analog Channel Transfer Function may not apply all Analog Channels).

The Device Variable and Analog Channel commands require multiple indices to be identified and tested with each command. For each index, the Write or Set command may not be applicable for that command. In these cases, the DUT will answer with "Invalid Selection" (which is not considered a failure). If a Device Variable and Analog Channel command is implemented, it must be valid for at least one index.

The following tables summarize the test requirements. Each set of table entries are for the indicated Common Practice Command, and shows the request data and the valid Response Codes. Any special notes are included after the table.

References:

Specification	Rev.	Sections
Command Summary Specification	8.0	6, 7.4.2, 7.4.3
Universal Command Specification	6.0	6.15
Common Practice Command Specification	8.0	All

Table 4 Simple Write Protect Test Cases

	Read Command		Write Command	Input (Stimulus)		Output (Expected Result)
15	Read Device Information	34	Write PV Damping Value	Increment Value		RC=In Write Protect RC=Command Not Implemented
15	Read Device Information	35	Write PV Range Values	10% Change		RC=In Write Protect RC=Command Not Implemented
15	Read Device Information	36	Set PV Upper Range Value	---		RC=In Write Protect RC=Command Not Implemented

Read Command	Write Command	Input (Stimulus)		Output (Expected Result)

15 Read Device Information	37 Set PV Lower Range Value	---		RC=In Write Protect
				RC=Command Not Implemented

15 Read Device Information	47 Write PV Transfer Function	New Value		RC=In Write Protect
				RC=Command Not Implemented

n/a	38 Reset Configuration Changed Flag[1]	---		RC=Success
				RC=In Write Protect
				RC=Command Not Implemented

n/a	40 Enter/Exit Fixed Current Mode[2, 4]	4.25		RC=Success
				RC=In Write Protect
				RC=Command Not Implemented

n/a	41 Perform Self Test[3]	---		RC=Success
				RC=Access Restricted
				RC=Command Not Implemented

n/a	42 Perform Device Reset[3]	---		RC=Success
				RC=Access Restricted
				RC=Command Not Implemented

1 Read Primary Variable	43 Set PV Zero	---		RC=In Write Protect
				RC=Command Not Implemented

1 Read Primary Variable	44 Write PV Units	New Units Code		RC=In Write Protect
				RC=Command Not Implemented

2 Read Loop Current and Percent Range	45 Trim Loop Current Zero[4]	4.01mA		RC=In Write Protect
				RC=Command Not Implemented

2 Read Loop Current and Percent Range	46 Trim Loop Current Gain[4]	19.98		RC=In Write Protect
				RC=Command Not Implemented

				RC=Command Not Implemented[6]

14 Read Primary Variable Transducer Information	49 Write PV Transducer Serial Number	Increment Value		RC=In Write Protect
				RC=Command Not Implemented

50 Read Dynamic Variable Assignments	51 Write Dynamic Variable Assignments	Rotate Assignments		RC=In Write Protect
				RC=Command Not Implemented

0 Read Unique Identifier	59 Write Number Of Response Preambles	Increment Value		RC=In Write Protect
				RC=Command Not Implemented

76 Read Lock Device State	71 Lock Device[5]	Temporary and Permanent Lock		RC=Success
				RC=Cannot Lock Device
				RC=Command Not Implemented

n/a	77 Send Command to Sub-device	Embedded command 0		RC=Success
				RC=Command Not Implemented

	87 Write I/O System Master Mode	Mode 0		RC=In Write Protect
				RC=Command Not Implemented

	88 Write I/O System Retry Count	Count of 3		RC=In Write Protect
				RC=Command Not Implemented

Read Command	Write Command	Input (Stimulus)		Output (Expected Result)

90 Read Real-time Clock	89 Set Real-time Clock	Current time		RC=In Write Protect
				RC=Command Not Implemented

91 Read Trend Configuration	92 Write Trend Configuration	TrendControl 0		RC=In Write Protect
				RC=Command Not Implemented

96 Read Synchronous Action	97 Configure Synchronous Action	Action 0		RC=In Write Protect
				RC=Command Not Implemented

98 Read Command Action	99 Configure Command Action	Command 9		RC=In Write Protect
				RC=Command Not Implemented

101 Read sub-device to burst message mapping	102 Write sub-device to burst message mapping	Message 0 Sub-device 0		RC=In Write Protect
				RC=Command Not Implemented

n/a	103 Write Burst Period	Message 0 Period 2sec		RC=In Write Protect
				RC=Command Not Implemented

n/a	104 Write Burst Trigger	Message 0		RC=In Write Protect
				RC=Command Not Implemented

n/a	106 Flush Delayed Responses	---		RC=In Write Protect
				RC=Command Not Implemented

105 Read Burst Mode Configuration	108 Write Burst Mode Command Number	New Cmd Number		RC=In Write Protect
				RC=Command Not Implemented

n/a	109 Burst Mode Control	Message 0 Control 1		RC=In Write Protect
				RC=Command Not Implemented

115 Read Event Notification Summary	116 Write Event Notification Bit Mask	Event 0 with Command48 Bits		RC=In Write Protect
				RC=Command Not Implemented

n/a	117 Write Event Notification Timing	Event Time and Event 0		RC=In Write Protect
				RC=Command Not Implemented

n/a	118 Event Notification Control	Event 0 Control 0		RC=In Write Protect
				RC=Command Not Implemented

513 Read Country Code	513 Write Country Code	Action 0		RC=In Write Protect
				RC=Command Not Implemented

Notes on Table 4

1. Reset Configuration Change Flag allows Write Protect and the two bits are non-volatile. However, allowing a host application to ensure that the Configuration Change Flag is reset even while Write Protected is reasonable and a recommended practice.
2. Enter/Exit Fixed current mode allows Write Protect. However, better implementations will not Write Protect this command. This allows smart I/O system and other Hosts to periodically perform loop tests for diagnostic purposes. The Device Status provides appropriate feedback when the loop current is fixed.
3. Self Test and Device Reset may be disabled by write protect because they can cause temporary loss of communication and may cause a loop current transient. Devices inhibiting Self Test and Device Reset must Write Protect them as a set and return "Access Restricted"
4. Commands 40, 45, 46 require the loop current to be active.
5. Since Write Protect serves a similar purpose to Lock Device, implementations may indicate either "Success" or "Cannot Lock Device"

Test Case A: Simple Write and Set Commands

Checks Commands 34-38, 40-47, 49, 51, 59, 71, 77, 87, 88, 89, 92, 97, 99, 102-104, 106. 108, 109, 116-118, 513. See Table 4.

```
CALL IdentifyDevice()
SEND Command 15
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF

CALL TestValidFrame
IF (write protect = 251, "None")
      THEN Test result is ABORT                                         (7330)
END IF

IF (DUT is NOT in "Write Protect") THEN
      Prompt user: "Place DUT into Write Protect"
END IF
```

Verify that we are in Write Protect

```
SEND Command 15 to read writeProtect, lrv0, urv0, u0, damping, and tfc0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
CALL TestValidFrame
IF (DUT is NOT in "Write Protect")
      THEN Test result is FAIL                                          (7331)
END IF
```

Checking Commands 34-37, 47. Attempting to Write PV properties

```
SEND Command 34 with (damping + 1)
CALL TestValidFrame
IF ( (RESPONSE_CODE != "Command Not Implemented") and
  (RESPONSE_CODE != "In Write Protect") )
      THEN Test result is FAIL                                       (7332) |
ELSE
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (7333) |
      END IF
END IF

SEND Command 36
CALL TestValidFrame
IF ( (RESPONSE_CODE != "Command Not Implemented") and
  (RESPONSE_CODE != "In Write Protect") )
      THEN Test result is FAIL                                       (7334) |
ELSE
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (7335) |
      END IF
END IF
SEND Command 37
CALL TestValidFrame
IF ( (RESPONSE_CODE != "Command Not Implemented") and
  (RESPONSE_CODE != "In Write Protect") )
      THEN Test result is FAIL                                       (7336) |
ELSE
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (7337) |
      END IF
END IF

SET span10pct = 0.1 * (urv0 - lrv0)
SEND Command 35 with u0, (lrv0 + span10pct), (urv0 - span10pct)
CALL TestValidFrame
IF ( (RESPONSE_CODE != "Command Not Implemented") and
  (RESPONSE_CODE != "In Write Protect") )
      THEN Test result is FAIL                                       (7338) |
ELSE
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (7339) |
      END IF
END IF

SEND Command 47 with (tfc0 + 1)
CALL TestValidFrame
IF ( (RESPONSE_CODE != "Command Not Implemented") and
  (RESPONSE_CODE != "In Write Protect") )
      THEN Test result is FAIL                                       (7340) |
ELSE
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (7341) |
      END IF
END IF
```

```
SEND Command 15 to read lrv1, urv1, damping1, and tfc1
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
CALL TestValidFrame

IF (lrv1 != lrv0)
      THEN Test result is FAIL                                   (7342)
END IF

IF (urv1 != urv0)
      THEN Test result is FAIL                                   (7343)
END IF

IF (damping1 != damping)
      THEN Test result is FAIL                                   (7344)
END IF

IF (tfc1 != tfc)
      THEN Test result is FAIL                                   (7345)
END IF
```

Can we perform a loop test?

```
SEND Command 40 with 4.25mA
CALL TestValidFrame
SWITCH on (RESPONSE_CODE)
      CASE "Command Not Implemented"
            IF (BYTE_COUNT != 2)
                  THEN Test result is FAIL                       (7349)
            END IF

      CASE "In Write Protect"
            PRINT "Warning, Command 40 should function correctly
                  independent of write protect mode. This allows
                  smart I/O system and other Hosts to periodically
                  perform loop tests for diagnostic purposes."      (7350)

            IF (BYTE_COUNT != 2)
                  THEN Test result is FAIL                       (7351)
            END IF
      CASE "Loop Current Not Active"
            Test result is FAIL                                  (7352)

      CASE "Success"
            IF (BYTE_COUNT != 6)
                  THEN Test result is FAIL                       (7353)
            END IF
            SEND Command 40 with 0.00mA
            CALL TestValidFrame
            CALL VerifyResponseAndByteCount(0, 6)

      CASE DEFAULT
            Test result is FAIL                                  (7354)
END SWITCH
```

Can we perform a self test or device reset? Remember that communications is sometimes lost after a device reset or a self test.

```
SEND Command 41
```

```
CALL TestValidFrame
IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                        (7355)
END IF
IF (RESPONSE_CODE == "Command Not Implemented")

ELSE IF (RESPONSE_CODE == "Success")
      DO
            SEND Command 0 (Primary)
      WHILE (COMMUNICATIONS_ERROR == "No Response")
      SEND Command 42
      CALL TestValidFrame
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                  (7356)
      END IF
      IF ( (RESPONSE_CODE != "Command Not Implemented") and
        (RESPONSE_CODE != "Success") )
            THEN Test result is FAIL                                  (7357)
      END IF

ELSE IF (RESPONSE_CODE == "Access Restricted")
      DO
            SEND Command 3
      WHILE (COMMUNICATIONS_ERROR == "No Response")
      SEND Command 42
      CALL TestValidFrame
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                  (7358)
      END IF
      IF ( (RESPONSE_CODE != "Command Not Implemented") and
        (RESPONSE_CODE != "Access Restricted") )
            THEN Test result is FAIL                                  (7359)
      END IF

ELSE
      Test result is FAIL                                             (7360)

END IF

DO
      SEND Command 3
WHILE (COMMUNICATIONS_ERROR == "No Response")
```

Can we Set PV Zero (Command 43)?

```
SEND Command 43
CALL TestValidFrame
IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                        (7361)
END IF
IF ( (RESPONSE_CODE != "Command Not Implemented") and
  (RESPONSE_CODE != "In Write Protect") )
      THEN Test result is FAIL                                        (7362)
END IF
```

Can we change the PV Units (Command 44)?

```
SEND Command 1 to read units0
CALL TestValidFrame
CALL VerifyResponseAndByteCount(0, 7)
SEND Command 44 with (units0 + 1)
IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                     (7363)
END IF
IF ( (RESPONSE_CODE != "Command Not Implemented") and
  (RESPONSE_CODE != "In Write Protect") )
      THEN Test result is FAIL                                     (7364)
END IF
SEND Command 1 to read units1
CALL TestValidFrame
CALL VerifyResponseAndByteCount(0, 7)
IF (units1 != units0)
      THEN Test result is FAIL                                     (7365)
END IF
```

Can we trim the loop current zero (Command 45) or gain (Command 46)?

```
SEND Command 45 with 4.01mA
CALL TestValidFrame
IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                     (7366)
END IF
IF ( (RESPONSE_CODE != "Command Not Implemented") and
  (RESPONSE_CODE != "In Write Protect") )
      THEN Test result is FAIL                                     (7367)
END IF
SEND Command 46 with 19.98mA
CALL TestValidFrame
IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                     (7368)
END IF
IF ( (RESPONSE_CODE != "Command Not Implemented") and
  (RESPONSE_CODE != "In Write Protect") )
      THEN Test result is FAIL                                     (7369)
END IF
```

Try Writing the PV Transducer Serial Number (Command 49)?

```
SEND Command 14 to read tsn0
CALL TestValidFrame
CALL VerifyResponseAndByteCount(0, 18)
SEND Command 49 with (tsn0 + 1)
IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                     (7170)
END IF
IF ( (RESPONSE_CODE != "Command Not Implemented") and
  (RESPONSE_CODE != "In Write Protect") )
      THEN Test result is FAIL                                     (7171)
END IF
SEND Command 14 to read tsn1
CALL TestValidFrame
CALL VerifyResponseAndByteCount(0, 18)
IF (tsn1 != tsn0)
      THEN Test result is FAIL                                     (7172)
END IF
```

Try Re-Mapping PV (Command 51)?

```
SEND Command 50 to read pv0, sv0, tv0, qv0
SWITCH on (BYTE_COUNT)
      CASE [4, 5, 6]
            SET pv2 = sv0
            IF (sv0 == 250)
                  THEN SET pv2 = pv0 + 1
            END IF
            SEND Command 51 mapping pv2 to PV
            CALL TestValidFrame
            IF (BYTE_COUNT != 2)
                  THEN Test result is FAIL                            (7173)
            END IF
            IF ( (RESPONSE_CODE != "Command Not Implemented") and
              (RESPONSE_CODE != "In Write Protect") )
                  THEN Test result is FAIL                            (7174)
            END IF

      CASE 3
            SET pv2 = pv0 + 1
            SEND Command 51 mapping pv2 to PV
            CALL TestValidFrame
            IF (BYTE_COUNT != 2)
                  THEN Test result is FAIL                            (7175)
            END IF
            IF ( (RESPONSE_CODE != "Command Not Implemented") and
              (RESPONSE_CODE != "In Write Protect") )
                  THEN Test result is FAIL                            (7176)
            END IF

      CASE DEFAULT
            Test result is FAIL                                       (7177)
END SWITCH
SEND Command 50 to read pv1, sv1, tv1, qv1
IF (pv1 != pv0)
      THEN Test result is FAIL                                        (7178)
END IF
```

Can we change the number of response preambles (Command 59)?

```
SEND Command 59 with response preambles = 5
CALL TestValidFrame
IF (RESPONSE_CODE == "Success") THEN
      SEND Command 59 with response preambles = 15
      CALL TestValidFrame
ENDIF
IF ( (RESPONSE_CODE != "Command Not Implemented") and
  (RESPONSE_CODE != "In Write Protect") )
      THEN Test result is FAIL                                        (7179)
END IF

IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                        (7180)
END IF
```

Can we send commands to sub-devices (Command 77)?

```
      SEND Command 77 with
            Card 0
            Channel 0
            Preambles 5
            Address shortframe polling 0
            Command 0
            Byte Count 0
            Data Field 0

      IF RESPONSE_CODE != "Command not Implemented" AND
        RESPONSE_CODE != "SUCCESS"
            THEN Test result is FAIL                                             (7185)
      END IF
```

Can we change I/O System Master Mode (Command 87)?

```
      SEND Command 87 with 1
      IF RESPONSE_CODE != "Command not Implemented" AND
            RESPONSE_CODE != "In Write Protect"
            THEN Test result is FAIL                                             (7186)
      END IF
```

Can we change I/O System Retry Count (Command 88)?

```
      SEND Command 88 with retry count 3
      IF RESPONSE_CODE != "Command not Implemented" AND
        RESPONSE_CODE != "In Write Protect"
            THEN Test result is FAIL                                             (7187)
      END IF
```

Can we set Real-Time Clock (Command 89)?

```
      SEND Command 89 with
            Set Code 1
            Current Date
            Current Time
            Two additional bytes of 00

      IF RESPONSE_CODE != "Command not Implemented" AND
        RESPONSE_CODE != "In Write Protect"
            THEN Test result is FAIL                                             (7370)
      END IF
```

Can we change trend configuration (Command 92)?

```
      SEND Command 92 with
            Trend Number = 0
            Trend Control = 0
            Device Variable Code = 0
            Trend Sample Rate = 0x0DBBA000

      IF RESPONSE_CODE != "Command not Implemented" AND
        RESPONSE_CODE != "In Write Protect"
            THEN Test result is FAIL                                             (7371)
      END IF
```

Can we configure a synchronous action (Command 97)?

```
SEND Command 97 with
      Action Number = 0
      Action Control = 16
      Device Variable = 0
      Command Number = 1
      Trigger Date = Current Date
      Trigger Time = Current Time

IF RESPONSE_CODE != "Command not Implemented" AND
  RESPONSE_CODE != "In Write Protect"
      THEN Test result is FAIL                                      (7372)
END IF
```

Can we configure a command action (Command 99)?

```
SEND Command 99 with
            Sample Trigger Action = Continuous
            Command Number = 1
            Byte Count = 7
            no request Data Bytes

IF RESPONSE_CODE != "Command not Implemented" AND
  RESPONSE_CODE != "In Write Protect"
      THEN Test result is FAIL                                      (7373)
END IF
```

Can we map a sub-device to a burst message (Command 102)?

```
SEND Command 102 with
  Burst Message = 0
      Sub-device = 0

IF RESPONSE_CODE != "Command not Implemented" AND
  RESPONSE_CODE != "In Write Protect"
      THEN Test result is FAIL                                      (7374)
END IF
```

Can we change the burst period (Command 103)?

```
SEND Command 103 with
      Burst Message = 0
      Update Period = 1 second
      Maximum Period = 1 second

IF RESPONSE_CODE != "Command not Implemented" AND
  RESPONSE_CODE != "In Write Protect"
      THEN Test result is FAIL                                      (7375)
END IF
```

Can we change the burst trigger (Command 104)?

```
SEND Command 104 with
      Burst Message = 0
      Burst Trigger Mode Selection Code =
      Device Variable = 0
      Units Code = 0x39
      Trigger Level =

IF RESPONSE_CODE != "Command not Implemented" AND
  RESPONSE_CODE != "In Write Protect"
      THEN Test result is FAIL                                      (7376)
END IF
```

Command 108, Write Burst Mode Command Number

```
SEND Command 108 with Command Number 3 and Burst Message 0
```

```
      IF RESPONSE_CODE != "Command not Implemented" AND
        RESPONSE_CODE != "In Write Protect"
            THEN Test result is FAIL                                     (7379)
      END IF
```

Command 109, Burst Mode Control

```
      SEND Command 109 with Burst Mode Control Code Enabled and Burst Message 0
      IF RESPONSE_CODE != "Command not Implemented" AND
        RESPONSE_CODE != "In Write Protect"
            THEN Test result is FAIL                                     (7380)
      END IF
```

Command 116, Write Event Notification Bit Mask

```
      SEND Command 116 with
            Event Number = 0
            BitMask = Extended Device Status of Command48 (Byte 6)
      IF RESPONSE_CODE != "Command not Implemented" AND
        RESPONSE_CODE != "In Write Protect"
            THEN Test result is FAIL                                     (7381)
      END IF
```

Command 117, Write Event Notification Timing

```
      SEND Command 117 with
            Event Number = 0
            Retry Time = 0.500s
            Maximum Update Time = 0.500s
            De-bounce Interval = 0.500s
      IF RESPONSE_CODE != "Command not Implemented" AND
        RESPONSE_CODE != "In Write Protect"
            THEN Test result is FAIL                                     (7382)
      END IF
```

Command 118, Event Notification Control

```
      SEND Command 118 with
            Event Number = 0
            Event Control = 0
      IF RESPONSE_CODE != "Command not Implemented" AND
        RESPONSE_CODE != "In Write Protect"
            THEN Test result is FAIL                                     (7383)
      END IF
```

Command 513, Write Country Code

```
      SEND Command 513 with
            Country Code="es"
            SI Unit Restriction = 0
      IF RESPONSE_CODE != "Command not Implemented" AND
        RESPONSE_CODE != "In Write Protect"
            THEN Test result is FAIL                                     (7384)
      END IF

      END TEST CASE
```

Table 5 Write Protect Test Cases for Device Variables

	Command	Input (Stimulus)	Output (Result)

52	Set Device Variable Zero	dVar Code	RC=In Write Protect
			RC=Command Not Implemented

53	Write Device Variable Units	dVar Code,
		New Units Code
			RC=In Write Protect
			RC=Command Not Implemented

55	Write Device Variable Damping Value	dVar Code,
		Increment Value
			RC=In Write Protect
			RC=Command Not Implemented

56	Write Device Variable Transducer Serial No.	dVar Code,
		New Value
			RC=In Write Protect
			RC=Command Not Implemented

79	Write Device Variable[1]	dVar Code,
		Same Units,
		(UTL / LSL)/2,
		"Manual" | "Constant"
			RC=In Write Protect
			RC=Command Not Implemented

82	Write Device Variable Trim Point[2]	dVar Code, Same Units,
		Trim Point Code,
		LTL or UTL
			RC=In Write Protect
			RC=Command Not Implemented

83	Reset Device Variable Trim	dVar Code	RC=In Write Protect
			RC=Command Not Implemented

107	Write Burst Device Variables	Cmd 50 Codes, or
		First dVar Number
			RC=In Write Protect
			RC=Command Not Implemented

113	Catch Device Variable	dVar Code	RC=In Write Protect
			RC=Command Not Implemented

Notes on Table 5

❑ Write Device Variable allows Write Protect. However, some implementations may not Write Protect this command. This allows the operation of Hosts and shutdown systems to be periodically maintained and tested. In addition, Field Devices incorporating PID or other control actions may use this command to write setpoints. Valves-Actuators may find it desirable to bump shut-off valves using this command as part of a predictive maintenance program.

❑ Either the Lower or Upper Transducer Limit is sent based on the response to Command 81, Read Device Variable Trim Guidelines.

Test Case B: Device Variable Write and Set Commands

Checks Commands 52, 53, 55, 56, 79, 82, 83, 107, and 113. See Table 5..

```
CALL IdentifyDevice()
SEND Command 15
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
CALL TestValidFrame
IF (write protect = 251, "None")
      THEN Test result is ABORT                                 (7190)
END IF
```

```
      Prompt user: "Place DUT into Write Protect"
```

Verify that we are in Write Protect

```
      SEND Command 15
      IF ( UNIV_REVISION >= 6 )
            CALL VerifyResponseAndByteCount(0, 20)
      ELSE
            THEN CALL VerifyResponseAndByteCount(0, 19)
      END IF
      CALL TestValidFrame
      IF (DUT is NOT in "Write Protect")
            THEN Test result is FAIL                                             (7191)
      END IF
```

Are any of the Device Variable commands supported?

```
      SET cmdSupported == FALSE
      SEND Command 52
      IF (RESPONSE_CODE != "Command not Implemented")
            THEN SET cmdSupported == TRUE
      END IF
      SEND Command 53
      IF (RESPONSE_CODE != "Command not Implemented")
            THEN SET cmdSupported == TRUE
      END IF
      SEND Command 55
      IF (RESPONSE_CODE != "Command not Implemented")
            THEN SET cmdSupported == TRUE
      END IF
      SEND Command 56
      IF (RESPONSE_CODE != "Command not Implemented")
            THEN SET cmdSupported == TRUE
      END IF
      SEND Command 79
      IF (RESPONSE_CODE != "Command not Implemented")
            THEN SET cmdSupported == TRUE
      END IF
      SEND Command 82
      IF (RESPONSE_CODE != "Command not Implemented")
            THEN SET cmdSupported == TRUE
      END IF
      SEND Command 83
      IF (RESPONSE_CODE != "Command not Implemented")
            THEN SET cmdSupported == TRUE
      END IF
      SEND Command 107
      IF (RESPONSE_CODE != "Command not Implemented")
            THEN SET cmdSupported == TRUE
      END IF
      SEND Command 113
      IF (RESPONSE_CODE != "Command not Implemented")
            THEN SET cmdSupported == TRUE
      END IF

      IF (! CmdSupported)
            THEN Abort Test                                                      (7192)
      END IF
```

Command 33 must be supported to find valid Device Variables

```
      SEND Command 33 with no data bytes
      IF (RESPONSE_CODE == "Command not Implemented")
            THEN Test result is FAIL                                             (7193)
```

```
END IF
```

Command 54 must be supported to read Device Variable Information

```
SEND Command 54 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                       (7194)
END IF
```

There must be at least one valid Device Variable

```
dVar = -1
IF (FindNextDeviceVariable(dVar) == "No More Device Variables")
      THEN Test result is FAIL                                       (7195)
END IF
```

Check the Device Variable write commands

```
DO
```

Command 52, Set Device Variable Zero

```
      SEND Command 52 with dVar
      IF ( (RESPONSE_CODE != "Command not Implemented") and
        (RESPONSE_CODE != "In Write Protect") and
        (RESPONSE_CODE != "Invalid Selection") )
            THEN Test result is FAIL                                 (7196)
      END IF
```

Command 53, Write Device Variable Units

```
      SEND Command 33 (with one byte = dVar) to read units0
      CALL TestValidFrame
      IF ( (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Update Failure") )
            THEN Test result is FAIL                                 (7197)
      ELSE IF (BYTE_COUNT != 8)
            THEN Test result is FAIL                                 (7198)
      END IF
      SEND Command 53 with dVar and (units + 1)
      IF ( (RESPONSE_CODE != "Command not Implemented") and
        (RESPONSE_CODE != "In Write Protect") and
        (RESPONSE_CODE != "Invalid Device Variable Code") )
            THEN Test result is FAIL                                 (7199)
      END IF

      SEND Command 33 (with one byte = dVar) to read units1
      CALL TestValidFrame
      IF ( (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Update Failure") )
            THEN Test result is FAIL                                 (7200)
      END IF
      IF (BYTE_COUNT != 8)
            THEN Test result is FAIL                                 (7201)
      END IF
      IF (units1 != units0)
            THEN Test result is FAIL                                 (7202)
      END IF
```

Command 55, Write Device Variable Damping Value

```
      SEND Command 54 to read damp0, transducerSN0, utl0, utl1
      CALL TestValidFrame
      CALL VerifyResponseAndByteCount(0, 25)

      SEND Command 55 with (damp0 +1)
      IF (RESPONSE_CODE != "Command not Implemented")
        (RESPONSE_CODE != "In Write Protect") and
        (RESPONSE_CODE != "Invalid Selection") )
            THEN Test result is FAIL                                 (7203)
      END IF
```

Command 56, Write Device Variable Transducer Serial No.

```
SEND Command 56 with (transducerSN0 +1)
IF (RESPONSE_CODE != "Command not Implemented")
  (RESPONSE_CODE != "In Write Protect") and
  (RESPONSE_CODE != "Invalid Selection") )
      THEN Test result is FAIL                                  (7204)
END IF

SEND Command 54 to read damp1, transducerSN1
CALL TestValidFrame
CALL VerifyResponseAndByteCount(0, 25)
IF (damp1 != damp0)
      THEN Test result is FAIL                                  (7205)
END IF
IF (transducerSN1 != transducerSN0)
      THEN Test result is FAIL                                  (7206)
END IF
```

Command 79, Write Device Variable

```
SEND Command 33 (with one byte = dVar) to read units0, val0
CALL TestValidFrame
IF ( (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Update Failure") )
      THEN Test result is FAIL                                  (7207)
END IF
IF (BYTE_COUNT != 8)
      THEN Test result is FAIL                                  (7208)
END IF

SEND Command 79 with dVar, units0, ((utl0 + ltl0)/ 2)
IF ( (RESPONSE_CODE != "Command not Implemented")
  (RESPONSE_CODE != "In Write Protect") and
  (RESPONSE_CODE != "Device Variable index not allowed ...") )
      THEN Test result is FAIL                                  (7209)
END IF

SEND Command 33 (with one byte = dVar) to read units0, val1
CALL TestValidFrame
IF ( (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Update Failure") )
      THEN Test result is FAIL                                  (7210)
END IF
IF (BYTE_COUNT != 8)
      THEN Test result is FAIL                                  (7211)
END IF
IF (val1 != val0)
      THEN Test result is FAIL                                  (7212)
END IF
```

Command 82, Write Device Variable Trim Point

```
SEND Command 80 with dVar to read units0, ltp0, utp0
CALL TestValidFrame
IF (RESPONSE_CODE != "Device Variable index not allowed ...") and
  (RESPONSE_CODE != "Command not Implemented") ) THEN

      CALL VerifyResponseAndByteCount(0, 12)
      SEND Command 81 with dVar to read nPts, minLTP, minUTP
      CALL TestValidFrame
      CALL VerifyResponseAndByteCount(0, 25)

      IF ( nPts > 0) THEN
            SEND Command 82 with dVar, 1, minLTP
            IF (RESPONSE_CODE != "In Write Protect")
```

```
                  THEN Test result is FAIL                     (7213)
            END IF
      END IF
      IF ( nPts > 1) THEN
            SEND Command 82 with dVar, 2, minUTP
            IF (RESPONSE_CODE != "In Write Protect")
                  THEN Test result is FAIL                     (7214)
            END IF
      END IF

      SEND Command 80 with dVar to read units1, ltp1, utp1
      CALL TestValidFrame
      CALL VerifyResponseAndByteCount(0, 12)
      IF (ltp1 != ltp0)
            THEN Test result is FAIL                           (7215)
      END IF
      IF (utp1 != utp0)
            THEN Test result is FAIL                           (7216)
      END IF
```

Command 83, Reset Device Variable Trim

```
      SEND Command 83
      IF (RESPONSE_CODE != "In Write Protect")
            THEN Test result is FAIL                           (7217)
      END IF

      SEND Command 80 with dVar to read units1, ltp1, utp1
      CALL TestValidFrame
      CALL VerifyResponseAndByteCount(0, 12)
      IF (ltp1 != ltp0)
            THEN Test result is FAIL                           (7218)
      END IF
      IF (utp1 != utp0)
            THEN Test result is FAIL                           (7219)
      END IF

END IF
```

Command 107, Write Burst Device Variables

```
SEND Command 107 with dVar  (one request data byte)
IF ( (RESPONSE_CODE != "Command not Implemented")
      (RESPONSE_CODE != "In Write Protect") and
      (RESPONSE_CODE != "Invalid Selection") )
      THEN Test result is FAIL                                 (7220)
END IF
```

Command 113, Catch Device Variable

```
            SEND Command 114 to read sourceCmdNo
            IF (RESPONSE_CODE != "Command not Implemented")
                  SET cmdNo = 3
                  IF (sourceCmdNo == 3)
                        THEN SET cmdNo = 1
                  END IF

                  SEND Command 113 with dVar, modeCode = 2, sourcAddress = 0,
                    cmdNo, slot = 1, shedTime = 60

                  WHILE (RESPONSE_CODE == "BUSY" or "DR Running")
                        RE-SEND Command 113
                  END WHILE

                  IF ( (RESPONSE_CODE != "In Write Protect") and
                    (RESPONSE_CODE != "Device Variable index not allowed ...") )
                        THEN Test result is FAIL                          (7221)
                  END IF

            END IF

      WHILE (FindNextDeviceVariable(dVar) == "Device Variable Found")

      END TEST CASE
```

Table 6 Write Protect Test Cases for Analog Channels

	Command	Input (Stimulus)	Output (Result)

64	Write Analog Channel Damping Value	Increment Value	RC=In Write Protect
			RC=Command Not Implemented

65	Write Analog Channel Range Values	10% change	RC=In Write Protect
			RC=Command Not Implemented

66	Enter/Exit Fixed Analog Channel Mode[1]	Lower Endpoint + 10% Span	RC=In Write Protect
			RC=Command Not Implemented

67	Trim Analog Channel Zero	Lower Endpoint + 10% Span	RC=In Write Protect
			RC=Command Not Implemented

68	Trim Analog Channel Gain	Upper Endpoint - 10% Span	RC=In Write Protect
			RC=Command Not Implemented

69	Write Analog Channel Transfer Function	New Value	RC=In Write Protect
			RC=Command Not Implemented

Notes on Table 6

1. Enter/Exit Fixed Analog Channel Mode allows Write Protect. However, better implementations will not Write Protect this command. This allows smart I/O system and other Hosts to periodically perform loop tests for diagnostic purposes. The status bits in Command 48 provides appropriate feedback when the Analog Channel value is fixed.

Test Case C: Analog Channel Write and Set Commands

Checks Commands 64-69. See Table 6..

```
CALL IdentifyDevice()
SEND Command 15
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
CALL TestValidFrame
IF (write protect = 251, "None")
      THEN Test result is ABORT                                         (7230)
END IF

Prompt user: "Place DUT into Write Protect"
```

Verify that we are in Write Protect

```
SEND Command 15
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
CALL TestValidFrame
IF (DUT is NOT in "Write Protect")
      THEN Test result is FAIL                                          (7231)
END IF
```

Are any of the Analog Channel commands supported?

```
SET cmdSupported == FALSE
```

```
SEND Command 64
IF (RESPONSE_CODE != "Command not Implemented")
      THEN SET cmdSupported == TRUE
END IF
SEND Command 65
IF (RESPONSE_CODE != "Command not Implemented")
      THEN SET cmdSupported == TRUE
END IF
SEND Command 66
IF (RESPONSE_CODE != "Command not Implemented")
      THEN SET cmdSupported == TRUE
END IF
SEND Command 67
IF (RESPONSE_CODE != "Command not Implemented")
      THEN SET cmdSupported == TRUE
END IF
SEND Command 68
IF (RESPONSE_CODE != "Command not Implemented")
      THEN SET cmdSupported == TRUE
END IF
SEND Command 69
IF (RESPONSE_CODE != "Command not Implemented")
      THEN SET cmdSupported == TRUE
END IF

IF (! CmdSupported)
      THEN Abort Test                                                 (7232)
END IF
```

Command 60 must be supported to find valid Analog Channels

```
SEND Command 60 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                        (7233)
END IF
```

Command 63 must be supported to read Analog Channel Information

```
SEND Command 63 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                        (7234)
END IF
```

Command 70 must be supported to read Analog Channel Endpoints

```
SEND Command 70 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                        (7235)
END IF
```

There must be at least one valid Analog Channel

```
aChan = -1
IF (FindNextAnalogChannel(aChan) == "No More Analog Channels")
      THEN Test result is FAIL                                        (7236)
END IF
```

Test each Analog Channel.

```
DO
      SEND Command 63 with aChan to read xferCode0, damp0,
        units0, lrv0, urv0
      CALL TestValidFrame
      CALL VerifyResponseAndByteCount(0, 19)
```

Command 64, Write Analog Channel Damping Value

```
      SEND Command 64 with aChan, (damp0 +1)
```

```
IF (RESPONSE_CODE != "Command not Implemented")
  (RESPONSE_CODE != "In Write Protect") and
  (RESPONSE_CODE != "Invalid Selection") )
      THEN Test result is FAIL                                   (7237)
END IF
```

Command 65, Write Analog Channel Range Values

```
SET span10pct = 0.1 * (urv0 - lrv0)
SEND Command 65 with aChan, units0, (lrv0 + span10pct),
  (urv0 - span10pct)
CALL TestValidFrame
IF ( (RESPONSE_CODE != "Command Not Implemented")
  (RESPONSE_CODE != "Invalid Analog Channel Code") and
  (RESPONSE_CODE != "In Write Protect") )
      THEN Test result is FAIL                                   (7238)
ELSE
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                             (7239)
      END IF
END IF
```

Command 69, Write Analog Channel Transfer Function

```
SEND Command 69 with aChan, (xferCode0 + 1)
CALL TestValidFrame
IF ( (RESPONSE_CODE != "Command Not Implemented")
  (RESPONSE_CODE != "Invalid Analog Channel Code") and
  (RESPONSE_CODE != "In Write Protect") )
      THEN Test result is FAIL                                   (7240)
ELSE
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                             (7241)
      END IF
END IF
SEND Command 63 with aChan to read xferCode1, damp1,
  units0, lrv1, urv1
CALL TestValidFrame
CALL VerifyResponseAndByteCount(0, 19)
IF ( xferCode1 != xferCode0 )
      THEN Test result is FAIL                                   (7242)
END IF
IF ( damp1 != damp0 )
      THEN Test result is FAIL                                   (7243)
END IF
IF ( lrv1 != lrv0 )
      THEN Test result is FAIL                                   (7244)
END IF
IF ( urv1 != urv0 )
      THEN Test result is FAIL                                   (7245)
END IF
```

Command 66, Enter/Exit Fixed Analog Channel Mode

```
SEND Command 70 with aChan to read epUnits0, lep0, uep0
CALL TestValidFrame
CALL VerifyResponseAndByteCount(0, 12)

SET ep1pct = 0.01 * (uep0 - lep0)
SET ep10pct = 0.10 * (uep0 - lep0)
SEND Command 66 with aChan, epUnits0, (lep0 + ep10Pct)
CALL TestValidFrame
SWITCH on (RESPONSE_CODE)
      CASE "Command Not Implemented"
            IF (BYTE_COUNT != 2)
```

```
                        THEN Test result is FAIL                     (7229)
                  END IF
            CASE "Invalid Analog Channel Code"
                  IF (BYTE_COUNT != 2)
                        THEN Test result is FAIL                     (7246)
                  END IF
            CASE "In Write Protect"
                  PRINT "Warning, Command 66 should function correctly
                        independent of write protect mode. This allows
                        smart I/O system and other Hosts to periodically
                        perform loop tests for diagnostic purposes." (7247)
                  IF (BYTE_COUNT != 2)
                        THEN Test result is FAIL                     (7248)
                  END IF
            CASE "In Multidrop Mode"
                  Test result is FAIL                                (7249)
            CASE "Success"
                  IF (BYTE_COUNT != 8)
                        THEN Test result is FAIL                     (7250)
                  END IF
                  SEND Command 66 with "0x&F, 0xA0, 0x00, 0x00"
                  CALL TestValidFrame
                  CALL VerifyResponseAndByteCount(0, 8)
            CASE DEFAULT
                  Test result is FAIL                                (7251)
      END SWITCH
```

Command 67, Trim Analog Channel Zero

```
      SEND Command 67 with aChan, lep0 + ep1pct
      CALL TestValidFrame
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (7252)
      END IF
      IF ( (RESPONSE_CODE != "Command Not Implemented")
        (RESPONSE_CODE != "Invalid Analog Channel Code") and
        (RESPONSE_CODE != "In Write Protect") )
            THEN Test result is FAIL                                 (7253)
      END IF
```

Command 68, Trim Analog Channel Gain

```
      SEND Command 68 with aChan, uep0 - ep1pct
      CALL TestValidFrame
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (7254)
      END IF
      IF ( (RESPONSE_CODE != "Command Not Implemented")
        (RESPONSE_CODE != "Invalid Analog Channel Code") and
        (RESPONSE_CODE != "In Write Protect") )
            THEN Test result is FAIL                                 (7255)
      END IF

WHILE (FindNextAnalogChannel (aChan) == "Analog Channel Found")

END TEST CASE
```

7.3 CAL033 Read Device Variables

Verifies that the DUT responds properly to Command 33. Checks Addresses, Command Number, Response Codes and Byte Counts for Command 33, Read Device Variables.

Since no prior knowledge of the Device Variable code assignments is assumed, there will always be some set(s) of Device Variable codes which are valid and some sets of Device Variable codes which are invalid. Devices are required to reply to requests having more than the required data bytes, so five data bytes are used in determining the set of assigned Device Variables.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

The message sequence used is as follows:

1. Command 33 is sent with one data byte starting with Device Variable 0 and incrementing the Device Variable 249 to find a supported Device Variable. This Device Variable is used to verify Command 33 responses to 1-5 Request Data Bytes.

2. Command 33 is sent with one data byte starting with "00" hex and incrementing the data byte to "FE" hex to find a Device Variable that is not supported. The response to "Invalid Selection" is verified.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.8, 7.1

Test Procedure

```
CALL IdentifyDevice
CALL CheckForRecommendedCommand(33)
```

Find a supported Device Variable

```
dVar = -1
IF (FindNextDeviceVariable(dVar) == "No More Device Variables")
      THEN Test result is FAIL                                       (6300)
END IF
```

For each device variable, do some basic sanity checks

```
DO
      SEND Command 33 with dVar
      IF (slot 0 units = [0, 250-255])
            THEN Test result is FAIL                                 (6301)
      ELSE IF (slot 0 units = [170-219])
            SEND Command 54 with dVar
            IF (RESPONSE_CODE != 0)
                  THEN Test result is FAIL                           (6302)
            ELSE IF (Device Variable Classification = [0-63])
                  THEN Test result is FAIL                           (6303)
            END IF
      END IF
```

Send Command 33 with varying number of data bytes. Check command response length

```
      FOR ( nBytes = [2 – 5] )
            SEND Command 33 with nBytes of dVar
            CALL TestValidFrame
            IF ( (RESPONSE_CODE != 0)
              AND (RESPONSE_CODE != "Update Failure") )
                  THEN Test result is FAIL                           (6304)
            ELSE
                  IF (nBytes == 5 ) THEN
                        IF (BYTE_COUNT != 26 )
                              THEN Test result is FAIL               (6305)
                        END IF
                  ELSE IF (BYTE_COUNT != (2 + 6*nBytes) )
                        THEN Test result is FAIL                     (6306)
                  END IF
            END IF
      END FOR
WHILE (FindNextDeviceVariable(dVar) == "Device Variable Found")
```

For HART 7 and later devices, test the standardized device variable codes: 244, 245, 246, 247, 248, 249. For wireless products, also verify device variable code 243.

```
IF (UNIV_REVISION > 6) THEN

      SEND Command 33 with dVar = 246, 247, 248, 249
      CALL TestValidFrame
      IF ( (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Update Failure") )
            THEN Test result is FAIL                                 (6307)
      ELSE
            IF (BYTE_COUNT != 26 )
                  THEN Test result is FAIL                           (6308)
            END IF
      END IF
```

Use command 9 to get the primary variable value.

```
      SEND Command 9 with dVar = 246
      CALL TestValidFrame
      IF ( (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Update Failure") )
            THEN Test result is FAIL                                 (7457)
      ELSE
            IF (BYTE_COUNT != 15 )
                  THEN Test result is FAIL                           (7458)
            END IF
      END IF
```

Compare command 9 response for primary variable value to command 33.

```
      IF (Command33.DeviceVariableValue != Command9.DeviceVariableValue)
            THEN Test result is FAIL                                 (7459)
      END IF
```

Read the loop current value.

```
            SEND Command 33 with dVar = 245
            CALL TestValidFrame
            IF ( (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Update Failure") )
                  THEN Test result is FAIL                                   (7460)
            ELSE
                  IF (BYTE_COUNT != 8 )
                        THEN Test result is FAIL                             (7461)
                  END IF
            END IF
```

Read the percent Range.

```
            SEND Command 33 with dVar = 244
            CALL TestValidFrame
            IF ( (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Update Failure") )
                  THEN Test result is FAIL                                   (7462)
            ELSE
                  IF (BYTE_COUNT != 8 )
                        THEN Test result is FAIL                             (7463)
                  END IF
            END IF
```

For wireless products, verify the battery life.

```
            IF (PROFILE == WIRELESS)THEN
                  SEND Command 33 with dVar = 243
                  CALL TestValidFrame
                  IF ( (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Update
                  Failure") )
                        THEN Test result is FAIL                             (7464)
                  ELSE
                        IF (BYTE_COUNT != 8 )
                              THEN Test result is FAIL                       (7465)
                        END IF
                  END IF
            END IF

      END IF
```

Check "Invalid Selection" Response Code

```
      SEND Command 33 with 4 data bytes of 0xFF
      CALL VerifyResponseAndByteCount("Invalid Selection", 2)
      CALL TestValidFrame

      END TEST
```

7.4 CAL034 Write Primary Variable Damping Value

Verifies that the DUT responds properly to Command 34. Checks Addresses, Command Number, Response Code and Byte Count for Command 34, Write Primary Variable Damping Value.

This command allows some flexibility in Field Device implementation. For example, if an invalid Damping Value is received, an implementation can simply return an error (i.e., the Damping Value was not accepted). Alternatively, some implementations may choose to force the invalid Damping Value to a good value and return a warning. This test allows for either implementation.

A variety of Response Codes are available for this command. A sequence of Damping values are sent to simulate all of the possible conditions Table 7 summarizes the conditions tested and the legal DUT responses. To allow for different implementations, any one of the legal responses demonstrates compliance with the Command Specification. For example, Test Case 2 allows either of two valid responses.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

Table 7 Command 34: Damping Value Test Cases

	Input	**Output**			
	Damping	**Damping**	**BC**	**Result**	**Response Code**
1	damp0	damp0	6	*Success*	
2	1E30	U/C	2	*Error*	*Passed Parameter Too Large*
		NEW Damping	6	*Warning*	*Set to Nearest Possible Value*
3	-1.00	U/C	2	*Error*	*Passed Parameter Too Small*
		NEW Damping	6	*Warning*	*Set to Nearest Possible Value*
4	damp0 + .000001	NEW Damping	6	*Success*	
		NEW Damping	6	*Warning*	*Set to Nearest Possible Value*
5	damp0 + 1.00	NEW Damping	6	*Success*	
		U/C	2	*Error*	*Passed Parameter Too Large*
		NEW Damping	6	*Warning*	*Set to Nearest Possible Value*

Note 1:"U/C" indicates that the Damping Value in the DUT is unchanged by the command

Note 2:The term " damp0" indicates the last Damping Value read using Command 15

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.7, 7.2

Test Procedure

```
CALL IdentifyDevice
CALL CheckForRecommendedCommand(34)
CALL VerifyNotWriteProtected()
```

Determine initial damping value

```
SEND Command 15
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
SET damp0 to returned damping value
```

Valid message with an extra data byte.

```
SEND Command 34 with damp0 and an extra data byte
CALL VerifyResponseAndByteCount(0, 6)
CALL VerifyDampingTimeConstant(damp0, failurepoint)                  (6320)
```

Message with a very large damping time constant (1E30).

```
SEND Command 34 with 1E30
CALL TestValidFrame
IF (RESPONSE_CODE == "Passed Parameter Too Large")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (6321)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 6)
            THEN Test result is FAIL                                 (6322)
      END IF
ELSE
      Test result is FAIL                                            (6323)
END IF

IF (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET dlast to the returned damping time constant
ELSE
      SET dlast to damp0
END IF
CALL VerifyDampingTimeConstant(dlast, failurepoint)                  (6324)
```

Message with a negative damping time constant (-1).

```
SEND Command 34 with -1
CALL TestValidFrame
IF (RESPONSE_CODE == "Passed Parameter Too Small")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                (6325)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 6)
            THEN Test result is FAIL                                (6326)
      END IF
ELSE
      Test result is FAIL                                           (6327)
END IF

IF (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET dlast to the returned damping time constant
END IF
CALL VerifyDampingTimeConstant(dlast, failurepoint)                 (6328)
```

Message with high precision damping value (original damping value +0.000001 sec).

```
SEND Command 34 with (damp0 + 0.000001)
CALL TestValidFrame
IF (RESPONSE_CODE == "Success")
      IF (BYTE_COUNT != 6)
            THEN Test result is FAIL                                (6329)
      END IF
ELSE IF (RESPONSE_CODE == "Passed Parameter Too Large")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                (6330)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 6)
            THEN Test result is FAIL                                (6331)
      END IF
ELSE
      Test result is FAIL                                           (6332)
END IF

IF (RESPONSE_CODE == "SUCCESS")
   OR (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET dlast to the returned damping time constant
END IF
CALL VerifyDampingTimeConstant(dlast, failurepoint)                 (6333)
```

Message with original damping value + 1 second.

```
SEND Command 34 with damp0 + 1.0
CALL TestValidFrame
IF (RESPONSE_CODE == "SUCCESS")
      IF (BYTE_COUNT != 6)
            THEN Test result is FAIL                                    (6334)
      END IF
ELSE IF (RESPONSE_CODE == "Passed Parameter Too Large")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                    (6335)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 6)
            THEN Test result is FAIL                                    (6336)
      END IF
ELSE
      Test result is FAIL                                               (6337)
END IF
IF ((RESPONSE_CODE == "SUCCESS")
   OR (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET dlast to the returned damping time constant
ELSE
      SET dlast to damp0
END IF
CALL VerifyDampingTimeConstant(dlast, failurepoint)                     (6338)
```

Message with original damping value.

```
SEND Command 34 with damp0
CALL VerifyResponseAndByteCount(0, 6)
CALL VerifyDampingTimeConstant(damp0, failurepoint)                     (6339)
END TEST
```

VerifyDampingTimeConstant(d, failurepoint)

This procedure is unique to CAL034. It uses command 15 to verify that the damping time constant has the expected value. The procedure loops on the command 15 until the DUT returns a non-"Busy" response to the command.

```
PROCEDURE VerifyDampingTimeConstant(d, FAILUREPOINT)
DO
      SEND Command 1
      SEND Command 15
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                                    (6340)
      END IF
WHILE (RESPONSE_CODE == "Busy")
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF

IF damping time constant != d
      THEN Test result is FAIL                                  (FAILUREPOINT)
END IF
PROCEDURE END
```

7.5 CAL035 Write Primary Variable Range Values

Verifies that the DUT responds properly to Command 35. Checks Addresses, Command Number, Response Code and Byte Count for Command 35, Write Primary Variable Range Values. This test only checks the function of the command, not the entire device.

This command allows some flexibility in Field Device implementation. For example, if an invalid Range Value is received, an implementation can simply return an error (i.e., the Range Value was not accepted). Alternatively, some implementations may choose to force the invalid Range Value to a good value and return a warning. This test allows for either implementation.

A variety of Response Codes are available for this command. A sequence of range values is sent to simulate all of the possible conditions. Table 8 summarizes the conditions tested and the legal DUT responses. To allow for different implementations, any one of the legal responses demonstrates compliance with the Command Specification. For example, Test Case 2 allows either of two valid responses.

Features Tested (see Table 8):

- Data is sent to provoke all allowed Response Codes;
- Command 15 is used to verify writes of Range Values; and
- When "Busy" is detected, Command 1 is sent to confirm proper use of this Response Code.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

Feature NOT Tested:

- The Percent Range is not verified against the Range Values applied; and
- The Unit Code is not varied to confirm the PV Units are unaffected by Command 35

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.7, 7.3

Table 8 Command 35: Set Range Values Test Cases

	Input		**Output**				
	URV	**LRV**	**URV**	**LRV**	**BC**	**Result**	**Response Code**
1	urv0	lrv0	urv0	lrv0	11	*Success*	
2	1E30	lrv0	U/C	U/C	2	*Error*	*URV Too High*
			NEW URV	lrv0	11	*Warning*	*Set to Nearest Possible Value*
3	-1E30	lrv0	U/C	U/C	2	*Error*	*URV Too Low*
			NEW URV	lrv0	11	*Warning*	*Set to Nearest Possible Value*
4	urv0	1E30	U/C	U/C	2	*Error*	*LRV Too High*
			urv0	NEW LRV	11	*Warning*	*Set to Nearest Possible Value*
5	urv0	-1E30	U/C	U/C	2	*Error*	*LRV Too Low*
			urv0	NEW LRV	11	*Warning*	*Set to Nearest Possible Value*
6	1E30	-1E30	U/C	U/C	2	*Error*	*URV and LRV Out Of Limits*
			U/C	U/C	2	*Error*	*LRV Too Low*
			U/C	U/C	2	*Error*	*URV Too High*
			NEW URV	NEW LRV	11	*Warning*	*Set to Nearest Possible Value*
7	lrv0	lrv0	lrv0	lrv0	11	*Warning*	*Span Too Small*
			U/C	U/C	2	*Error*	*Invalid Span*
			NEW URV	NEW LRV	11	*Warning*	*Set to Nearest Possible Value*

Note 1:"U/C" indicates that the Range Value in the DUT is unchanged by the command
Note 2:The terms "urv0" and "lrv0" indicates the last Range Value(s) read using Command 15
Note 3:Contact the HCF if your device supports range values greater than 1E30

Test Procedure

```
CALL IdentifyDevice
CALL CheckForRecommendedCommand(35)
CALL VerifyNotWriteProtected()
PRINT "Warning: The Units Code sent By Command 35 must not change
      the PV Units Code.  Compliance with this requirement must be
      manually Verified"
```

Send a valid message with the original range values and units code as read by Command 15.

```
SEND Command 15 to read lrv0, urv0, u0
lrvOriginial = lrv0; urvOriginal = urv0; unitsOriginal = u0

SEND Command 35 to set LRV = lrv0, URV = urv0, units = u0
CALL VerifyResponseAndByteCount(0, 11)
CALL VerifyRange(lrv0, urv0, u0, failurepoint)                       (6350)
```

Send a valid message with an extra data byte.

```
SEND Command 35 to set LRV = lrv0, URV = urv0, units = u0, with one extra
      data byte
CALL VerifyResponseAndByteCount(0, 11)
CALL VerifyRange(lrv0, urv0, u0, failurepoint)                       (6355)
```

A message with a very large positive upper range value (1E30) and original lower range value.

```
SEND Command 35 to set LRV = lrv0, URV = 1E30, units = u0

CALL TestValidFrame
IF (RESPONSE_CODE == "SUCCESS")
      IF (BYTE_COUNT != 11)
            THEN Test result is FAIL                                 (6360)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 11)
            THEN Test result is FAIL                                 (6361)
      END IF
ELSE IF (RESPONSE_CODE == "Upper Range Value Too High")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (6362)
      END IF
ELSE
      Test result is FAIL                                            (6363)
END IF

IF (RESPONSE_CODE == "SUCCESS")
   OR (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET urv0 to the URV
END IF
CALL VerifyRange(lrv0, urv0, u0, failurepoint)                       (6365)
```

A message with a very large negative upper range value (-1E30) and original lower range value.

```
SEND Command 35 to set LRV = lrv, URV = -1E30, units = u0

IF (RESPONSE_CODE == "SUCCESS")
      IF (BYTE_COUNT != 11)
            THEN Test result is FAIL                                  (6370)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 11)
            THEN Test result is FAIL                                  (6371)
      END IF
ELSE IF (RESPONSE_CODE == "Upper Range Value Too Low")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                  (6372)
      END IF
ELSE
      Test result is FAIL                                             (6373)
END IF

IF (RESPONSE_CODE == "SUCCESS")
   OR (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET urv0 to the URV
END IF
CALL VerifyRange(lrv0, urv0, u0, failurepoint)                        (6375)
```

A message with a very large positive lower range value (1E30) and original upper range value

```
SEND Command 35 to set LRV = 1E30, URV = urv0, units = u0

CALL TestValidFrame
IF (RESPONSE_CODE == "SUCCESS")
      IF (BYTE_COUNT != 11)
            THEN Test result is FAIL                                  (6380)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 11)
            THEN Test result is FAIL                                  (6381)
      END IF
ELSE IF (RESPONSE_CODE == "Lower Range Value Too High")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                  (6382)
      END IF
ELSE
      Test result is FAIL                                             (6383)
END IF

IF (RESPONSE_CODE == "SUCCESS")
   OR (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET lrv0 to the LRV
END IF
CALL VerifyRange(lrv0, urv0, u0, failurepoint)                        (6385)
```

A message with a very large negative lower range value (-1E30) and original upper range value.

```
SEND Command 35 to set LRV = -1E30, URV = urv0, units = u0

CALL TestValidFrame
IF (RESPONSE_CODE == "SUCCESS")
      IF (BYTE_COUNT != 11)
            THEN Test result is FAIL                                   (6390)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 11)
            THEN Test result is FAIL                                   (6391)
      END IF
ELSE IF (RESPONSE_CODE == "Lower Range Value Too Low")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                   (6392)
      END IF
ELSE
      Test result is FAIL                                              (6393)
END IF

IF (RESPONSE_CODE == "SUCCESS")
   OR (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET lrv0 to the LRV
END IF
CALL VerifyRange(lrv0, urv0, u0, failurepoint)                         (6395)
```

A message with a very large positive upper range value (1E30) and a very large negative lower range value (-1E30)

```
SEND Command 35 to set LRV = -1E30, URV = 1E30, units = u0

CALL TestValidFrame
IF (RESPONSE_CODE == "SUCCESS")
      IF (BYTE_COUNT != 11)
            THEN Test result is FAIL                                   (6400)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 11)
            THEN Test result is FAIL                                   (6401)
      END IF
ELSE IF (RESPONSE_CODE == "Upper and Lower Range Values Out Of Limits")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                   (6402)
      END IF
ELSE IF (RESPONSE_CODE == "Lower Range Value Too Low")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                   (6394)
      END IF
ELSE IF (RESPONSE_CODE == "Upper Range Value Too High")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                   (6404)
      END IF
ELSE
      Test result is FAIL                                              (6403)
END IF
```

```
IF (RESPONSE_CODE == "SUCCESS")
   OR (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET lrv0 to the LRV
      SET urv0 to the URV
END IF
CALL VerifyRange(lrv0, urv0, u0, failurepoint)                    (6405)
```

A message with the upper range value equal to the lower range value.

```
SEND Command 35 to set LRV = lrv0, URV = lrv0, units = u0

CALL TestValidFrame
IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 11)
            THEN Test result is FAIL                              (6411)
      END IF
ELSE IF (RESPONSE_CODE == "Span Too Small")
      IF (BYTE_COUNT != 11)
            THEN Test result is FAIL                              (6412)
      END IF
ELSE IF (RESPONSE_CODE == "Invalid Span")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                              (6413)
      END IF
ELSE
      Test result is FAIL                                         (6414)
END IF

IF ( (RESPONSE_CODE == "Set To Nearest Possible Value")
   OR (RESPONSE_CODE == "Span Too Small") ) THEN
      SET lrv0 to the LRV
      SET urv0 to the URV
END IF
CALL VerifyRange(lrv0, urv0, u0, failurepoint)                    (6415)
```

A message with a units code of 0xFF.

```
SEND Command 35 to set LRV = lrv0, URV = lrv0 and units = 0xFF.
IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                    (6416)
END IF
IF ( (RESPONSE_CODE != "Invalid Selection") and
  (RESPONSE_CODE != "Invalid Units Code") )
      THEN Test result is FAIL                                    (6417)
END IF
CALL VerifyRange(lrv0, urv0, u0, failurepoint)                    (6420)
```

A message with the original range values and units code.

```
SEND Command 35 to set LRV = lrvOriginal, URV = lrvOriginal
      and units = uOriginal.
CALL VerifyResponseAndByteCount(0, 11)
CALL VerifyRange(lrvOriginal, lrvOriginal, uOriginal, failurepoint)  (6425)

END TEST
```

VerifyRange(lrv, urv, units, failurepoint)

This procedure is unique to CAL035. It uses command 15 to verify that the upper and lower range values are as expected. The procedure loops on the command 15 until the DUT returns a non-"Busy" response to the command.

```
PROCEDURE VerifyRange(lrv, urv, units, FAILUREPOINT)
DO
      SEND Command 1
      SEND Command 15 to read l, u, un
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                                (6428)
      END IF
WHILE (RESPONSE_CODE = "Busy")

IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF

IF lrv != l THEN
      Test result is FAIL                                 (FAILUREPOINT+0)
END IF
IF urv != u THEN
      Test result is FAIL                                 (FAILUREPOINT+1)
END IF
IF units != un THEN
      Test result is FAIL                                 (FAILUREPOINT+2)
END IF
PROCEDURE END
```

7.6 CAL036 Set Primary Variable Upper Range Value

Verifies that the DUT responds properly to Command 36. Checks Addresses, Command Number, Response Code and Byte Count for Command 36, Set Primary Variable Upper Range Value.

Table 9 summarizes the conditions used to test the DUT's response to Command 36

Note: If the LRV is not zero, the test will abort.

Table 9 Command 36 Test Cases

Case	Condition	Response Code	URV
1	PV near LRV	"Span Too Small"	URV == PV
		"Invalid Span"	URV Unchanged
		"Set To Nearest Possible Value"	URV Changed
2	PV > UTL	"Applied Process Too High"	URV Unchanged
3	PV < LTL	"Applied Process Too Low"	URV Unchanged
4	PV slightly less than UTL	Success	URV == PV

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.7, 7.4

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(36)
CALL VerifyNotWriteProtected()
```

LRV must be set to zero

```
SEND Command 15 to read lrv0, urv0, u0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
IF (lrv0 != 0.0)
      THEN Test Result is ABORT                                      (5000)
ENDIF
```

Check for "Span too Small" or "Invalid Span" by setting PV close to zero.

```
Prompt user: "Set the PV close to the Lower Range Value (lrv0)."
CALL ReadPV()
SEND Command 36
IF (RESPONSE_CODE = "Command not Implemented")
      THEN Test Result is ABORT                                        (5000)
END IF

CALL TestValidFrame
IF (RESPONSE_CODE == "Span Too Small")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                   (6451)
      ELSE
            CALL VerifyRangeAndPV(lrv0,PV,u0,PV,failurepoint)   (6452-6457)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                   (6458)
      END IF
ELSE IF (RESPONSE_CODE == "Invalid Span")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                   (6459)
      ELSE
            CALL VerifyRangeAndPV(lrv0,urv0,u0,PV,failurepoint) (6460-6465)
      END IF
ELSE
      Test result is FAIL                                              (6466)
END IF
```

Check for "Applied Process Too High" above the Upper Transducer Limit

```
Prompt user: "Set PV above the Upper Transducer Limit."
CALL ReadPV()
SEND Command 15 to read lrv0, urv0, u0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
SEND Command 36

CALL TestValidFrame
IF (RESPONSE_CODE == "Applied Process Too High")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                   (6468)
      ELSE
            CALL VerifyRangeAndPV(lrv0,urv0,u0,PV,failurepoint) (6470-6475)
      END IF
ELSE
      Test result is FAIL                                              (6469)
END IF
```

Check for "Applied Process Too Low" below the Lower Transducer Limit

```
Prompt user: "Set PV below the Lower Transducer Limit."
CALL ReadPV()
SEND Command 15 to read lrv0, urv0, u0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
SEND Command 36

CALL TestValidFrame
IF (RESPONSE_CODE == "Applied Process Too Low")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (6477)
      ELSE
            CALL VerifyRangeAndPV(lrv0,urv0,u0,PV,failurepoint) (6478-6483)
      END IF
ELSE
      Test result is FAIL                                            (6484)
END IF
```

Check for success just below the Upper Transducer Limit

```
Prompt user: "Set PV close to (but not greater than) the Upper Transducer
      Limit."
CALL ReadPV()
SEND Command 15 to read lrv0, urv0, u0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
SEND Command 36
CALL VerifyResponseAndByteCount(0, 2)
CALL VerifyRangeAndPV(lrv0, PV, u0, PV, failurepoint)           (6485-6490)

END TEST
```

7.7 CAL037 Set Primary Variable Lower Range Value

Verifies that the DUT responds properly to Command 37. Checks Addresses, Command Number, Response Code and Byte Count for Command 37, Set Primary Variable Lower Range Value. The span of the range values will stay the same unless the Lower Range Value pushes the Upper Range Value beyond a device limit.

Table 10 summarizes the conditions used to test the DUT's response to Command 37

Note: If the URV is not at the UTL the test will abort.

Table 10 Command 37 Test Cases

Case	Condition	Response Code	Range Values
1	PV Mid-Range	"New Lower Range Value Pushed"	LRV == PV URV == UTL
		"Applied Process Too High"	URV, LRV Unchanged
2	PV near UTL	"Invalid Span"	URV, LRV Unchanged
		"Applied Process Too High"	URV, LRV Unchanged
		"New Lower Range Value Pushed"	LRV == PV URV == UTL
3	PV > UTL	"Applied Process Too High"	URV, LRV Unchanged
4	PV < LTL	"Applied Process Too Low"	LRV, URV Unchanged
5	PV = 0.0	Success	LRV == PV

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.7, 7.5

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(37)
CALL VerifyNotWriteProtected()
```

URV must be set to UTL

```
Prompt user: "Set Upper Range Value to Upper Transducer Limit."
SEND Command 14 to read LTL, UTL
VerifyResponseAndByteCount(0,18)
SEND Command 15 to read lrv0, urv0, u0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
IF (lrv0 != UTL) THEN
      PRINT "URV must be set to Upper Transducer Limit"
      Test Result is FAIL                                             (6495)
END IF
```

Check PV near mid range

```
Prompt user: "Set PV to mid sensor range."
```

```
CALL ReadPV()
SEND Command 15 to read lrv0, urv0, u0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
SEND Command 37
IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                         (6496)
END IF
SWITCH on (RESPONSE_CODE)
      CASE "New Lower Range Value Pushed"
            CALL VerifyRangeAndPV(PV,UTL,u0,PV,failurepoint)    (6497-6502)
            SET span0 = UTL - PV

      CASE "Applied Process Too High"
            CALL VerifyRangeAndPV(lrv0,urv0,u0,PV,failurepoint) (6503-6508)
            SET span0 = urv0 – lrv0

      CASE DEFAULT
            Test result is FAIL                                         (6509)
END SWITCH
```

Check for PV near the Upper Transducer Limit

```
Prompt user: "Set PV close to (but not greater than) the Upper Transducer
      Limit."
CALL ReadPV()
SEND Command 15 to read lrv0, urv0, u0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
SEND Command 37
IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                         (6510)
END IF
SWITCH on (RESPONSE_CODE)
      CASE "Invalid Span"
            CALL VerifyRangeAndPV(lrv0,urv0,u0,PV,failurepoint) (6511-6516)

      CASE "Applied Process Too High"
            CALL VerifyRangeAndPV(lrv0,urv0,u0,PV,failurepoint) (6517-6522)

      CASE "New Lower Range Value Pushed"
            CALL VerifyRangeAndPV(PV,UTL,u0,PV,failurepoint)    (6523-6528)
      CASE DEFAULT
            Test result is FAIL                                         (6529)
END SWITCH
```

Check for "Applied Process Too High" above the Upper Transducer Limit

```
Prompt user: "Set PV above the Upper Transducer Limit."
CALL ReadPV()
SEND Command 15 to read lrv0, urv0, u0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
SEND Command 37
CALL VerifyResponseAndByteCount("Applied Process Too High", 2)
CALL VerifyRangeAndPV(lrv0, urv0, u0, PV, failurepoint)         (6530-6535)
```

Check for "Applied Process Too Low" below the Lower Transducer Limit

```
Prompt user: "Set PV below the Lower Transducer Limit."
CALL ReadPV()
SEND Command 15 to read lrv0, urv0, u0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
SEND Command 37
CALL VerifyResponseAndByteCount("Applied Process Too Low", 2)
CALL VerifyRangeAndPV(lrv0, urv0, u0, PV, failurepoint)         (6536-6541)
```

Check for success near 0.0

```
Prompt user: "Set PV to 0.0"
CALL ReadPV()
SEND Command 15 to read lrv0, urv0, u0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
SEND Command 37
CALL VerifyResponseAndByteCount(0, 2)
CALL VerifyRangeAndPV(PV, (span0+PV), u0, PV, failurepoint)     (6542-6547)

END TEST
```

7.8 CAL039 (Reserved)

Implementation of Common Practice Command 39 is not recommended. As a result, Field Device implementations are not tested.

7.9 CAL040 Enter/Exit Fixed Current Mode

Checks Addresses, Command Number, Response Code and Byte Count for Command 40, Enter/Exit Fixed Current Mode.

Command 40 allows a host to set the Primary Variable Current of a device to a value within the normal operating range. The host returns the Primary Variable Current to normal operation by sending a value of 0.0 to the device. When the device is in fixed current mode, the Device Status bit indicating Primary Variable Current Fixed must be set. Conversely, when the host returns the device Primary Variable Current to normal operation, this bit must be reset.

A sequence of commands is then sent to determine proper Response Code acknowledgment and Primary Variable Current Fixed Device Status bit state. A Command 2 is sent to read the starting current value. Then after each Command 40 is sent to the device, a Command 2 is sent to read the current value. Table 11 summarizes the conditions used to test the DUT's response to Command 40.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

Table 11 Command 40 Test Cases

Case	Command 40	Loop Current	Response Code	Loop Current Fixed
1	4.25mA	Enabled	Success	Yes
2	4.5 mA + Extra Data Byte	Enabled	Success	Yes
3	100.0 mA	Enabled	"Passed Parameter Too Large"	Yes
4	-4.0 mA	Enabled	"Passed Parameter Too Small"	Yes
5	0.0	Enabled	Success	No
6	4.25	Disabled	"Loop Current Not Active"	Yes

Note: Voltage mode (e.g., 1-5V) devices and devices operating over a part of the loop current range may perform this test manually with values appropriate to the range they support. To request a waiver provide detailed, written justification for the device's behavior to the HCF prior to submitting your device for registration. The HCF will assess the justification and, when appropriate, grant a waiver.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.3, 7.8

Test Procedure

```
CALL IdentifyDevice
CALL CheckForRecommendedCommand (40)
SEND Command 6 with one byte and poll address 0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 4)
      IF (DEVICE_STATUS == "Loop Current Fixed")
            THEN Test Result is FAIL                         (6567)
      END IF
      ELSE
            THEN CALL VerifyResponseAndByteCount(0, 3)
      END IF
```

Store original loop current value

```
      lcv = ReadLoopCurrentValue
```

Command 40 with valid message content (setting = 4.25).

```
      SEND Command 40 with loop current = 4.25
      CALL VerifyResponseAndByteCount(0, 6)
      IF (DEVICE_STATUS != "Loop Current Fixed")
            THEN Test Result is FAIL                                 (6568)
      END IF
      CALL VerifyLoopCurrent(4.25, failurepoint)                (6569-6570)
```

Command 40 with valid message (current value of 4.5) with an extra data byte.

```
      SEND Command 40 with loop current = 4.5 with extra data byte
      CALL VerifyResponseAndByteCount(0, 6)
      IF (DEVICE_STATUS != "Loop Current Fixed")
            THEN Test Result is FAIL                                 (6571)
      END IF
      CALL VerifyLoopCurrent(4.5, failurepoint)                 (6572-6573)
```

Command 40 message with a large current value (current value of 100.0).

```
      SEND Command 40 with current value = 100.0
      CALL VerifyResponseAndByteCount("Passed Parameter Too Large", 2)
      IF (DEVICE_STATUS != "Loop Current Fixed")
            THEN Test Result is FAIL                                 (6574)
      END IF
      CALL VerifyLoopCurrent(4.5, failurepoint)                 (6575-6576)
```

Command 40 message with a negative current value (current value of -4.0).

```
      SEND Command 40 with current value = -4.0
      CALL VerifyResponseAndByteCount("Passed Parameter Too Small", 2)
      IF (DEVICE_STATUS != "Loop Current Fixed")
            THEN Test Result is FAIL                                 (6577)
      END IF
      CALL VerifyLoopCurrent(4.5, failurepoint)                 (6578-6579)
```

Command 40 with valid message content (current value of 0.0).

```
      SEND Command 40 with current value = 0.0 mA
      CALL VerifyResponseAndByteCount(0, 6)
      IF (DEVICE_STATUS == "Loop Current Fixed")
            THEN Test Result is FAIL                                 (6580)
      END IF
      CALL VerifyLoopCurrent(lcv, failurepoint)                 (6581-6582)
```

Command 6 changing the poll address to 1.

```
      SEND Command 6 with one byte and polling address 1
      CALL VerifyResponseAndByteCount(0, 4)
```

```
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 4)
      IF (DEVICE_STATUS != "Loop Current Fixed")
            THEN Test Result is FAIL                                        (6583)
      END IF
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 3)
END IF
```

Command 40 with valid message content (current value of 16 mA).

```
SEND Command 40 with current value = 16.0 mA
CALL VerifyResponseAndByteCount("Loop Current Not Active", 2)
IF (DEVICE_STATUS != "Loop Current Fixed")
      THEN Test Result is FAIL                                              (6584)
END IF
```

Command 6 changing the poll address to 0.

```
SEND Command 6 with one byte and polling address 0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 4)
      IF (DEVICE_STATUS == "Loop Current Fixed")
            THEN Test Result is FAIL                                        (6585)
      END IF
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 3)
END IF

END TEST
```

ReadLoopCurrentValue()

This procedure is unique to CAL040. It uses command 2 to read the Loop Current. Command 2 does not allow a "Busy" response.

```
PROCEDURE ReadLoopCurrentValue()
DO
      SEND Command 2 to read loop current
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Update Failure") AND (BYTE_COUNT != 10) )
            THEN Test result is FAIL                                        (6586)
      END IF
WHILE (RESPONSE_CODE == "Update Failure")
VerifyResponseAndByteCount(0,10)                                            (6587)
RETURN loop current

PROCEDURE  END
```

7.10 CAL041 Perform Self Test

Verifies that the DUT responds properly to Command 41. Checks Addresses, Command Number, Response Code and Byte Count for Command 41, Perform Self Test.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.4, 7.9

Test Procedure

```
CALL IdentifyDevice
```

Initiate self test

```
SEND Command 41
IF (RESPONSE_CODE = "Command not Implemented")
      PRINT "Warning, Implementation of Command 41 is strongly
            recommended.  This command is implemented by most
            Field Devices and widely used in Host Applications."
      Abort Test                                                     (5002)
END IF
CALL VerifyResponseAndByteCount(0, 2)
CALL TestValidFrame

END TEST
```

7.11 CAL042 Perform Device Reset

Verifies that the DUT responds properly to Command 42. Checks Addresses, Command Number, Response Code and Byte Count for Command 42, Perform Device Reset.

This command requests the device to reset the microprocessor. Further communication with the device may or may not be possible for a limited amount of time. After the Command 42 is sent, a series of Command 0 request will be sent at timed intervals until the device responds to two Command 0 requests.

Note: Master (Host testing device) must not disconnect after command 42.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.4, 7.10

Test Procedure

```
      CALL IdentifyDevice
      IF ( UNIV_REVISION == 7) THEN
            byteCount = 24
      ELSE IF ( UNIV_REVISION == 6) THEN
            byteCount = 19
      ELSE IF ( UNIV_REVISION == 5) THEN
            byteCount = 16
      ELSE
            Abort Test                                                          (5002)
      END IF
```

Initiate Device Reset

```
      IF (UNIV_REVISION >= 6) THEN
            SEND Command 0 to read the = configuration changed counter (cfgCntr)
      END IF
      SEND Command 42
      IF (RESPONSE_CODE = "Command not Implemented")
            PRINT "Warning, Implementation of Command 42 is strongly
                  recommended.  This command is implemented by most
                  Field Devices and widely used in Host Applications."
            Abort Test                                                          (5002)
      END IF
      CALL VerifyResponseAndByteCount(0, 2)
      CALL TestValidFrame
```

First command after and Device Reset, Cold Start must be set

```
      SET Master = Secondary
      DO
            SEND Command 0
      WHILE ((COMMUNICATIONS_ERROR == "No Response")
            OR (RESPONSE CODE == "BUSY"))

      CALL VerifyResponseAndByteCount(0, byteCount)

      CALL TestValidFrame
      IF (DEVICE_STATUS != "Cold Start")
            THEN Test Result is FAIL                                            (6590)
      END IF
      IF ( UNIV_REVISION >= 6 )
```

```
            IF (configuration changed counter != cfgCntr)
                  THEN Test Result is FAIL                                (6591)
            END IF
      END IF

      SEND Command 0
      CALL VerifyResponseAndByteCount(0, byteCount)

      CALL TestValidFrame
      IF (DEVICE_STATUS == "Cold Start")
            THEN Test Result is FAIL                                      (6592)
      END IF
```

Make sure that the Primary's bit is still set.

```
      SET Master = Primary
      SEND Command 0
      CALL VerifyResponseAndByteCount(0, byteCount)

      CALL TestValidFrame
      IF (DEVICE_STATUS != "Cold Start")
            THEN Test Result is FAIL                                      (6593)
      END IF

      IF ( UNIV_REVISION >= 6 )
            IF (configuration changed counter != cfgCntr)
                  THEN Test Result is FAIL                                (6594)
            END IF
      END IF

      END TEST
```

7.12 CAL043 Set Primary Variable Zero

Verifies that the DUT responds properly to Command 43. Checks Addresses, Command Number, Response Code and Byte Count for Command 43, Set Primary Variable Zero.

A message is sent to the user to establish a sensor input close to zero measurement (the value must be close enough to the zero measurement to allow the device to re-zero its calibration, but greater than 1% of its normal measurement span from its present calibration). The user indicates when ready. Then Command 1 is sent to read the Primary Variable Value. Command 43 is then sent to re-zero the device followed by a Command 1.

A message is sent to the user to change the sensor input to a value that is too high for the device to re-zero its calibration. The user indicates when ready. Then Command 1 is sent to read the Primary Variable Value. Command 43 is then sent to attempt to re-zero the device followed by a sending Command 1 to verify proper device operation.

A message is sent to the user to change the sensor input to a value that is too low for the device to re-zero its calibration. The user indicates when ready. Then Command 1 is sent to read the Primary Variable Value. Command 43 is then sent to attempt to re-zero the device followed by a sending Command 1 to verify proper device operation.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.7, 7.11

Test Procedure

```
CALL IdentifyDevice
CALL VerifyNotWriteProtected()
```

Perform and normal re-zero of PV

```
Prompt user: "Set the PV close to about 1% of range."
CALL ReadPV()
IF (PV == 0.0) THEN
      PRINT "PV must be near to 1% of range (i.e. not zero)"
      Test Result is FAIL                                              (6595)
END IF

SEND Command 43
CALL TestValidFrame
IF (RESPONSE_CODE = "Command not Implemented")
      THEN Abort Test                                                  (5000)
END IF
```

```
IF (RESPONSE_CODE != "Success")
      THEN Test Result is FAIL                                       (6588)
END IF
IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                       (6589)
END IF

CALL ReadPV
IF (PV != 0.0) THEN
      THEN Test Result is FAIL                                       (6596)
END IF
```

Too high:

```
Prompt user: "Set the PV too high to re-zero."
CALL ReadPV
SET PV0 = PV
SEND Command 43
CALL TestValidFrame
CALL VerifyResponseAndByteCount("Applied Process Too High", 2)

CALL ReadPV()
IF (PV0 != PV)
      THEN Test Result is FAIL                                       (6597)
END IF
```

Too low:

```
Prompt user: "Set the PV too low to re-zero."
CALL ReadPV
SET PV0 = PV
SEND Command 43
CALL TestValidFrame
CALL VerifyResponseAndByteCount("Applied Process Too Low", 2)
CALL ReadPV()
IF (PV0 != PV)
      THEN Test Result is FAIL                                       (6598)
END IF

END TEST
```

7.13 CAL044 Write Primary Variable Units

Verifies that the DUT responds properly to Command 44. Checks Addresses, Command Number, Response Code and Byte Count for Command 44, Write Primary Variable Units.

A Command 1 is sent to determine the original Primary Variable units code of the device. Then a sequence of Command 44 is sent using all possible byte values from 0 to 0xFE. If a "Busy" Response Code (32) is detected during this sequence, the Command 44 will be repeated until the "Busy" Response Code is not detected. This is then followed by a sequence of a Command 44 followed by Commands 1, 14 and 15. If a "Busy" Response Code (32) is detected, Command 1 will be repeated until the Response Code is 0. The sequence of commands used is as follows:

1. A valid message is sent containing the original unit code.
2. A valid message is sent with an extra data byte (original unit code).
3. An invalid message is sent which is one data byte short.
4. A valid message is sent containing the original unit code.

Note 1: "Too Few Data Bytes Received" Response Code is verified in CAL000.

Note 2: In some devices, transducer units or range units may not be the same as PV units. When this occurs, the HCF may grant waiver on Failure Point 7126, 7135, 7145, 7155. To request a waiver provide detailed, written justification for the device's behavior to the HCF prior to submitting your device for registration. The HCF will assess the justification and, when appropriate, grant a waiver.

Note 3: Some devices may support PV units of "Special" (code 253). When this occurs, the HCF may grant waiver on Failure Point 7125, 7134, 7144, 7154. To request a waiver provide detailed, written justification for the device's behavior to the HCF prior to submitting your device for registration. The HCF will assess the justification and, when appropriate, grant a waiver.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.7, 7.12

Test Procedure

```
CALL IdentifyDevice
CALL CheckForRecommendedCommand (44)
CALL VerifyNotWriteProtected()
```

Read original Primary Variable Units code.

```
SEND Command 1 to read PVu0
IF (RESPONSE_CODE == "Update Failure")
      CALL VerifyResponseAndByteCount("Update Failure", 7)
ELSE
      CALL VerifyResponseAndByteCount(0, 7)
ENDIF
CALL TestValidFrame
```

Test all possibilities for units code.

```
SET PVuLast = PVu0
FOR ( n = [0 – 255] )
      SEND Command 44 with code = n
      IF (RESPONSE_CODE == "SUCCESS") THEN
            CALL VerifyResponseAndByteCount(0, 3)
            PVuLast = n
            PRINT "PV Unit Code PVuLast supported"
      ELSE IF (RESPONSE_CODE == "Invalid Selection") THEN
            CALL VerifyResponseAndByteCount("Invalid Selection", 2)
      ELSE
            Test result is FAIL                                         (7120)
      END IF
      CALL VerifyUnits(PVuLast, failurepoint)                      (7121-7126)
FOR END
```

A valid message is sent with an extra data byte (original unit code).

```
SEND Command 44 with code = PVu0 and one extra data byte
CALL VerifyResponseAndByteCount(0, 3)
CALL TestValidFrame
CALL VerifyUnits(PVu0, failurepoint)                               (7130-7135)
```

A valid message is sent containing the original unit code.

```
SEND Command 44 with code = PVu0
CALL VerifyResponseAndByteCount(0, 3)
CALL TestValidFrame
CALL VerifyUnits(PVu0, failurepoint)                               (7140-7145)
```

Verify that Command 35 does not affect PV Units.

```
SEND Command 35
IF (RESPONSE_CODE == "Too Few Data Bytes")
      SEND Command 44 with code = PVuLast
      CALL VerifyResponseAndByteCount(0, 3)
      CALL TestValidFrame

      SEND Command 15 to read urv0, lrv0

      SEND Command 44 with code = PVu0
      CALL VerifyResponseAndByteCount(0, 3)
      CALL TestValidFrame

      SEND Command 35 with urv0, lrv0, PVuLast
      CALL VerifyResponseAndByteCount(0, 11)
      CALL TestValidFrame
      CALL VerifyUnits(PVu0, failurepoint)                         (7150-7155)
END IF

END TEST
```

VerifyUnits(u, failurepoint)

This procedure is unique to CAL044.

```
PROCEDURE VerifyUnits(u, FAILUREPOINT)
SEND Command 1 to read PVu
CALL VerifyResponseAndByteCount(0, 7)
CALL TestValidFrame
IF PVu != u
      THEN Test result is FAIL                             (FAILUREPOINT+0)
END IF

IF ( UNIV_REVISION >= 6 )

      SEND Command 8 to read PV classification
      IF (PV classification == [1-63] or 240-255)
            THEN Test result is FAIL                       (FAILUREPOINT+1)
      END IF
      IF ((PV classification == 0)
        AND (PVUnits > 169)
        AND (PVUnits < 220))
            THEN Test result is FAIL                       (FAILUREPOINT+2)
      END IF
      IF (PVUnits == [250-255] )
            THEN Test result is FAIL                       (FAILUREPOINT+3)
      END IF

END IF

SEND Command 14 to read transducer units
IF (analog channel flag indicates loop current is output)
      IF (transducer units != 250) THEN
            IF (pvUnits != transducer units)
                  THEN Test result is FAIL                 (FAILUREPOINT+5)
            END IF
      END IF
END IF
CALL VerifyResponseAndByteCount(0, 18)
CALL TestValidFrame
IF transducer limits units != u
      THEN Test result is FAIL                             (FAILUREPOINT+4)
END IF

PROCEDURE END
```

7.14 CAL045 Trim Loop Current Zero

Verifies that the DUT responds properly to Command 45. Checks Addresses, Command Number, Response Code and Byte Count for Command 45, Trim Loop Current Zero.

A variety of Response Codes are available for this command. A sequence of current values is used to simulate all of the possible conditions. To create error responses, invalid current values will be set to 100.0 and -4.0 for maximum and minimum extremes respectively. A sequence of Command 45 to create the Response Code values followed by Command 2 to read the current will be sent. Table 12 summarizes the conditions used to test the DUT's response to Command 45.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

Table 12 Command 45 Test Cases

Case	Loop Current	Cmd 45	Multi-Drop	Response Code
1	4.0 mA	4.0 mA	No	Success
2	4.0 mA	4.0 mA and Extra Data Byte	No	Success
3	4.0 mA	100.0 mA	No	"Passed Parameter Too Large"
4	4.0 mA	-4.0 mA	No	"Passed Parameter Too Small"
5	20.0 mA	4.0 mA	No	"Incorrect Loop Current Mode or Value"
6	(Don't Care)	4.0 mA	Yes	"Loop Current Not Active"

Note: Voltage mode (e.g., 1-5V) devices and devices operating over a part of the loop current range may perform this test manually with values appropriate to the range they support. To request a waiver provide detailed, written justification for the device's behavior to the HCF prior to submitting your device for registration. The HCF will assess the justification and, when appropriate, grant a waiver.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.3, 7.13

Test Procedure

```
CALL IdentifyDevice
CALL CheckForRecommendedCommand (45)
CALL VerifyNotWriteProtected()
SEND Command 6 with one byte and poll address 0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 4)
      IF (DEVICE_STATUS == "Loop Current Fixed")
            THEN Test Result is FAIL                                 (6600)
      END IF
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 3)
END IF
```

Loop Current is set to 4.00 mA.

```
IF (UNIV_REVISION >= 6) THEN
      SEND Command 15 to read PV Analog Channel Flags
ELSE
      Prompt user to set Analog Channel Flags
END IF
IF ("Analog Input Channel")
      Prompt user: "Set the Loop Current to 4.0 mA."
ELSE
      SEND Command 40 with Loop Current = 4.00 mA
      CALL VerifyResponseAndByteCount(0, 6)
END IF
CALL VerifyLoopCurrent(4.0, failurepoint)                       (6601-6602)
```

A valid Command 45 message is sent with a current value of 4.00 mA .

```
SEND Command 45 with current value = 4.00 mA
CALL VerifyResponseAndByteCount(0, 6)
CALL VerifyLoopCurrent(4.0, failurepoint)                       (6603-6604)
```

Check Command 45 with an extra data byte.

```
SEND Command 45 with Loop Current = 4.00 mA with extra byte
CALL VerifyResponseAndByteCount(0, 6)
CALL VerifyLoopCurrent(4.0, failurepoint)                       (6605-6606)
```

Check "Passed Parameter Too Large" with a large current value (100 mA).

```
SEND Command 45 with Loop Current = 100 mA
CALL VerifyResponseAndByteCount("Passed Parameter Too Large", 2)
CALL VerifyLoopCurrent(4.0, failurepoint)                       (6607-6608)
```

Check "Passed Parameter Too Small" with a negative current value (-4.00 mA).

```
SEND Command 45 with Loop Current = -4 mA
CALL VerifyResponseAndByteCount("Passed Parameter Too Small", 2)
CALL VerifyLoopCurrent(4.0, failurepoint)                       (6609-6610)
```

Loop Current is set to 20.00 mA.

```
IF ("Analog Input Channel")
      Prompt user: "Set the Loop Current to 20.0 mA."
ELSE
      SEND Command 40 with Loop Current = 20.00 mA
      CALL VerifyResponseAndByteCount(0, 6)
END IF
CALL VerifyLoopCurrent(20.0, failurepoint)                      (6611-6612)
SEND Command 45 with current value = 4.00 mA
CALL VerifyResponseAndByteCount("Incorrect Loop Current Mode or Value", 2)
CALL VerifyLoopCurrent(20.0, failurepoint)                      (6613-6614)
```

Remove DUT from fixed current mode (if necessary).

```
IF (!"Analog Input Channel")
      SEND Command 40 with Loop Current = 0.00 mA
      CALL VerifyResponseAndByteCount(0, 6)
END IF
```

Command 6 is sent changing the poll address to 1 (multi-drop mode).

```
DO
      SEND Command 1
      SEND Command 6 with polling address 1 and
         loop current signaling disabled
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                                 (6615)
      END IF
WHILE (RESPONSE_CODE == "Busy")
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 4)
      IF (DEVICE_STATUS != "Loop Current Fixed")
            THEN Test Result is FAIL                                 (6616)
      END IF
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 3)
END IF
```

A Command 45 message is sent with a current value of 4.00 mA.

```
SEND Command 45 with current value = 4.00 mA
CALL VerifyResponseAndByteCount("Loop Current Not Active", 2)
```

Command 6 changing the poll address to 0.

```
SEND Command 6 with polling address 0 and loop current signaling enabled
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 4)
      IF (DEVICE_STATUS == "Loop Current Fixed")
            THEN Test Result is FAIL                                 (6617)
      END IF
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 3)
END IF

END TEST
```

7.15 CAL046 Trim Loop Current Gain

Verifies that the DUT responds properly to Command 46. Checks Addresses, Command Number, Response Code and Byte Count for Command 46, Trim Loop Current Gain.

A variety of Response Codes are available for this command. A sequence of current values are sent to simulate all of the possible conditions. To create error responses, invalid current values will be set to 100.0 and -4.0 for maximum and minimum extremes respectively. A sequence of Command 46 to provoke the Response Code values followed by Command 2 to read the current are sent. When the current value is to be changed, Command 40 will be sent to establish the new current (if appropriate). Table 13 summarizes the conditions used to test the DUT's response to Command 46.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

Table 13 Command 46 Test Cases

Case	Loop Current	Cmd 45	Multi-Drop	Response Code
1	20.0 mA	20.0 mA	No	Success
2	20.0 mA	20.0 mA and Extra Data Byte	No	Success
3	20.0 mA	100.0 mA	No	"Passed Parameter Too Large"
4	20.0 mA	-4.0 mA	No	"Passed Parameter Too Small"
5	4.0 mA	20.0 mA	No	"Incorrect Loop Current Mode or Value"
6	(Don't Care)	20.0 mA	Yes	"Loop Current Not Active"

Note: Voltage mode (e.g., 1-5V) devices and devices operating over a part of the loop current range may perform this test manually with values appropriate to the range they support. To request a waiver provide detailed, written justification for the device's behavior to the HCF prior to submitting your device for registration. The HCF will assess the justification and, when appropriate, grant a waiver.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.3, 7.14

Test Procedure

```
      CALL IdentifyDevice
      CALL CheckForRecommendedCommand (46)
      CALL VerifyNotWriteProtected()
```

Loop Current is set to 20.00 mA.

```
      IF (UNIV_REVISION >= 6) THEN
            SEND Command 15 to read PV Analog Channel Flags
      ELSE
            Prompt user to set Analog Channel Flags
      END IF
      IF ("Analog Input Channel")
            Prompt user: "Set the Loop Current to 20.0 mA."
      ELSE
            SEND Command 40 with Loop Current = 20.00 mA
            CALL VerifyResponseAndByteCount(0, 6)
      END IF
      CALL VerifyLoopCurrent(20.0, failurepoint)                      (6618-6619)
```

A valid Command 46 message is sent with a current value of 20.00 mA .

```
      SEND Command 46 with Loop Current = 20.00 mA
      CALL VerifyResponseAndByteCount(0, 6)
      CALL VerifyLoopCurrent(20.0, failurepoint)                      (6620-6621)
```

A valid Command 46 message is sent with a current value of 20.00 mA and an extra data byte.

```
      SEND Command 46 with Loop Current = 20.00 mA with extra byte
      CALL VerifyResponseAndByteCount(0, 6)
      CALL VerifyLoopCurrent(20.0, failurepoint)                      (6622-6623)
```

A Command 46 message is sent with a large current value (100 mA).

```
      SEND Command 46 with Loop Current = 100 mA
      CALL VerifyResponseAndByteCount("Passed Parameter Too Large", 2)
      CALL VerifyLoopCurrent(20.0, failurepoint)                      (6624-6625)
```

A Command 46 message is sent with a negative current value (-4.00 mA).

```
      SEND Command 46 with Loop Current = -4 mA
      CALL VerifyResponseAndByteCount("Passed Parameter Too Small", 2)
      CALL VerifyLoopCurrent(20.0, failurepoint)                      (6626-6627)
```

Loop Current is set to 4.00 mA.

```
      IF ("Analog Input Channel")
            Prompt user: "Set the Loop Current to 4.0 mA."
      ELSE
            SEND Command 40 with Loop Current = 4.00 mA
            CALL VerifyResponseAndByteCount(0, 6)
      END IF
      CALL VerifyLoopCurrent(4.0, failurepoint)                       (6628-6629)
      SEND Command 46 with current value = 20.00 mA
      CALL VerifyResponseAndByteCount("Incorrect Loop Current Mode or Value", 2)
      CALL VerifyLoopCurrent(4.0, failurepoint)                       (6630-6631)
```

Remove DUT from fixed current mode (if necessary).

```
IF (!"Analog Input Channel")
      SEND Command 40 with Loop Current = 0.00 mA
      CALL VerifyResponseAndByteCount(0, 6)
END IF
```

Command 6 is sent changing the poll address to 1 (multi-drop mode).

```
DO
      SEND Command 1
      SEND Command 6 with polling address 1 and
         loop current signaling disabled
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                                 (6632)
      END IF
WHILE (RESPONSE_CODE == "Busy")
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 4)
      IF (DEVICE_STATUS != "Loop Current Fixed")
            THEN Test Result is FAIL                                 (6633)
      END IF
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 3)
END IF
```

A Command 46 message is sent with a current value of 20.00 mA.

```
SEND Command 46 with current value = 20.00 mA
CALL VerifyResponseAndByteCount("Loop Current Not Active", 2)
```

Command 6 changing the poll address to 0.

```
SEND Command 6 with polling address 0 and loop current signaling enabled
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 4)
      IF (DEVICE_STATUS == "Loop Current Fixed")
            THEN Test Result is FAIL                                 (6634)
      END IF
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 3)
END IF

END TEST
```

7.16 CAL047 Write Primary Variable Transfer Function

Verifies that the DUT responds properly to Command 47. Checks Addresses, Command Number, Response Code and Byte Count for Command 47, Write Primary Variable Transfer Function.

A Command 15 is sent to determine the original Primary Variable transfer function of the device. Then a sequence of Command 47 is sent using all possible byte values from 0 to 0xFE to determine the transfer functions codes actually used. If a "Busy" Response Code (32) is detected during this sequence, the Command 47 will be repeated until the "Busy" Response Code is not detected. This is then followed by a sequence of a Command 47 followed by a Command 15. If a "Busy" Response Code (32) is detected, Command 15 will be repeated until the Response Code is 0. The sequence of commands used is as follows:

2. A valid message is sent containing the original transfer function.

3. A valid message is sent with an extra data byte (original transfer function).

4. An invalid message is sent which is one data byte short.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.7, 7.15

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(47)
CALL VerifyNotWriteProtected()
```

Read original Transfer Function code.

```
SEND Command 15 to read tf0
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
CALL TestValidFrame
```

Test all possibilities for function code.

```
SET tfLast = tf0
FOR ( n = [0 – 254] )
      SEND Command 47 with code = n
      IF (RESPONSE_CODE == "SUCCESS") THEN
            CALL VerifyResponseAndByteCount(0, 3)
            tfLast = n
            SWITCH on (n)
                  CASE 250-254
                        Test result is FAIL                             (6642)
                  CASE 231-233
                        PRINT "Warning: Transfer Function Code <n>
                              is not interoperable and should not be
                              supported in any Field Device"
                  CASE DEFAULT

            END SWITCH

      ELSE IF (RESPONSE_CODE == "Invalid Selection") THEN
            CALL VerifyResponseAndByteCount("Invalid Selection", 2)
      ELSE
            Test result is FAIL                                     (6635)
      END IF
      CALL VerifyFunction(tfLast, failurepoint)                (6636-6637)
FOR END
```

A valid message is sent containing the original transfer function code.

```
SEND Command 47 with code = tf0
CALL VerifyResponseAndByteCount(0, 3)
CALL TestValidFrame
CALL VerifyFunction(tf0, failurepoint)                         (6638-6639)
```

A valid message is sent with an extra data byte (original transfer function code).

```
SEND Command 47 with code = tf0 and one extra data byte
CALL VerifyResponseAndByteCount(0, 3)
CALL TestValidFrame
CALL VerifyFunction(tf0, failurepoint)                         (6640-6641)

END TEST
```

VerifyFunction(f, failurepoint)

This procedure is unique to CAL047.

```
PROCEDURE VerifyFunction(f, FAILUREPOINT)
DO
      SEND Command 1
      SEND Command 15 to read transfer function
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                        (FAILUREPOINT)
      END IF
WHILE (RESPONSE_CODE == "Busy")
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
CALL TestValidFrame
IF transfer function != f
      THEN Test result is FAIL                            (FAILUREPOINT+1)
END IF
PROCEDURE END
```

7.17 CAL049 Write Primary Variable Transducer Serial Number

Verifies that the DUT responds properly to Command 49. Checks Addresses, Command Number, Response Code and Byte Count for Command 49, Write Primary Variable Transducer Serial Number.

A Command 14 is sent to determine the original Primary Variable Transducer serial number. Then a sequence of Command 49 followed by Command 14 is sent.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.7, 7.17

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(49)
CALL VerifyNotWriteProtected()
```

Read original serial number.

```
SEND Command 14 to read sn0
CALL VerifyResponseAndByteCount(0, 18)
CALL TestValidFrame
```

A valid message is sent containing a three byte serial number.

```
SET testsn = 3 bytes of 0x00
SEND Command 49 with testsn
CALL VerifyResponseAndByteCount(0, 5)
CALL TestValidFrame
CALL VerifySerialNumber(testsn, failurepoint)                    (6655-6656)

SET testsn = 3 bytes of 0xF0
SEND Command 49 with testsn
CALL VerifyResponseAndByteCount(0, 5)
CALL TestValidFrame
CALL VerifySerialNumber(testsn, failurepoint)                    (6657-6658)
```

A valid message is sent with an extra data byte (original serial number).

```
SEND Command 49 with sn0 and one extra data byte
CALL VerifyResponseAndByteCount(0, 5)
CALL TestValidFrame
CALL VerifySerialNumber(sn0, failurepoint)                       (6659-6660)
```

A valid message is sent containing the original serial number.

```
SEND Command 49 with sn0
CALL VerifyResponseAndByteCount(0, 5)
CALL TestValidFrame
CALL VerifySerialNumber(sn0, failurepoint)                       (6661-6662)

END TEST
```

VerifySerialNumber(sn, failurepoint)

This procedure is unique to CAL049.

```
PROCEDURE VerifySerialNumber(sn, FAILUREPOINT)

DO
      SEND Command 1
      SEND Command 14 to read Primary Variable Transducer Serial Number
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                          (FAILUREPOINT)
      END IF
WHILE (RESPONSE_CODE == "Busy")
CALL VerifyResponseAndByteCount(0, 18)
CALL TestValidFrame
IF Transducer Serial Number != sn
      THEN Test result is FAIL                              (FAILUREPOINT+1)
END IF

PROCEDURE END
```

7.18 CAL050 Read Dynamic Variable Assignments

Verifies that the DUT responds properly to Command 50. Checks Addresses, Command Number, Response Code and Byte Count for Command 50, Read Dynamic Variable Assignments.

Command 50 is sent and the reply analyzed for proper content.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.6, 7.18

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(50)
```

Read variable assignments and check for invalid Device Variable codes

```
SEND Command 50 to get PV_Dvar, SV_Dvar, TV_Dvar, QV_Dvar
IF (UNIV_REVISION >= 6) THEN
      CALL VerifyResponseAndByteCount(0, 6)
      Cmd50BC = 6
ELSE
      IF (RESPONSE_CODE != 0)
            THEN Test result is FAIL                                 (5110)
      END IF
      IF (BYTE_COUNT != 3, 4, 5, or 6)
            THEN Test result is FAIL                                 (5111)
      END IF
      Cmd50BC = BYTE_COUNT
END IF
CALL TestValidFrame
SEND Command 3
IF ( (BYTE_COUNT < 24) AND (Cmd50BC ==6) AND (QV_Dvar != 250) )
      THEN Test result is FAIL                                       (6665)
ELSE IF ( (BYTE_COUNT < 19) AND (Cmd50BC >=5) AND (TV_Dvar != 250) )
      THEN Test result is FAIL                                       (6666)
ELSE IF ( (BYTE_COUNT < 14) AND (Cmd50BC >=4) AND (SV_Dvar != 250) )
      THEN Test result is FAIL                                       (6667)
END IF

IF ( PV_Dvar > 243) )
      THEN Test result is FAIL                                       (6668)
END IF
IF ( SV_Dvar > 243) )
      THEN Test result is FAIL                                       (6669)
END IF
IF ( TV_Dvar > 243) )
      THEN Test result is FAIL                                       (6670)
END IF
IF ( QV_Dvar > 243) )
      THEN Test result is FAIL                                       (6671)
END IF
END TEST
```

7.19 CAL051 Write Dynamic Variable Assignments

Verifies that the DUT responds properly to Command 51. Checks Addresses, Command Number, Response Code and Byte Count for Command 51, Write Dynamic Variable Assignments.

Since no prior knowledge of the Dynamic Variable code assignments is assumed, there will always be some set(s) of Device Variable codes which are valid and some sets of Device Variable codes which are invalid for each of the Dynamic Variable assignments. Further, this is a command which may be truncated from four assigned Dynamic Variables to a single Dynamic Variable.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.6, 7.19

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(51)
CALL VerifyNotWriteProtected()
```

Determine original Device Variable mapping. Any device implementing Command 51 must also implement Command 50, allowing hosts to read the Dynamic Variable assignments.

```
SEND Command 50 to read dVarMap[]
CALL VerifyResponseAndByteCount(0, 6)
```

Find the mappable Device Variables

```
CREATE empty pvList, svList, tvList, qvList
SET vMap = dVarMap

CALL FindMappableDevVar(0, vMap, pvList, failurepoint)          (7270-7276)
Set vMap[0] = dVarMap[0]

IF (dVarMap[1] != "Not Used" ) THEN
      CALL FindMappableDevVar(1, vMap, svList, failurepoint)    (7277-7283)
      Set vMap[1] = dVarMap[1]
ENDIF

IF (dVarMap[2] != "Not Used" ) THEN
      CALL FindMappableDevVar(2, vMap, tvList, failurepoint)    (7284-7290)
      Set vMap[2] = dVarMap[2]
ENDIF

IF (dVarMap[3] != "Not Used" ) THEN
      CALL FindMappableDevVar (3, vMap, qvList, failurepoint)   (7291-7297)
      Set vMap[3] = dVarMap[3]
ENDIF

PRINT pvList, svList, tvList, qvList
```

Verify backward compatibility

```
      FOR n = 1 to 4
            SET vMap = first n bytes from dVarMap
            IF (IssueCmd51(vMap, failurepoint) != Success")          (7298-7301)
                  THEN Test result is FAIL                                (7302)
            END IF
      FOR END
      CALL VerifyDeviceVariableMapping(dVarMap, (failurepoint))       (7304-7305)
```

Test Command 51 with an extra byte

```
      SET vMap = dVarMap + one byte
      IF (IssueCmd51(vMap, failurepoint) != Success")                 (7306-7309)
            THEN Test result is FAIL                                      (7310)
      END IF
      CALL VerifyDeviceVariableMapping(dVarMap, failurepoint)         (7311-7312)
```

Write invalid Device Variable codes

```
      SET vMap = with 4 bytes of 0xFF
      IF (IssueCmd51(vMap, failurepoint) == Success")                 (7313-7316)
            THEN Test result is FAIL                                      (7317)
      END IF
      CALL VerifyDeviceVariableMapping(dVarMap, failurepoint)         (7318-7319)
```

For HART 7 and later devices, we test the required device variables.

```
      IF (UNIV_REVISION >= 7)

            SEND Command 51
            IF (RESPONSE_CODE != "Command not implemented")THEN
                  PV_Dvar = 246
                  SV_Dvar = 247
                  TV_Dvar = 248
                  QV_Dvar = 249

                  SEND Command 51 with PV_Dvar, SV_Dvar, TV_Dvar, QV_Dvar
                  IF (RESPONSE_CODE != 2)
                        THEN Test result is FAIL                          (7320)
                  END IF

                  IF BYTE_COUNT != 2
                        THEN Test result is FAIL                          (7321)
                  END IF

                  PV_DVAR = SV_DVAR = TV_DVAR = QV_DVAR = 245
                  SEND Command 51 with PV_Dvar, SV_Dvar, TV_Dvar, QV_Dvar
                  IF (RESPONSE_CODE != 2)
                        THEN Test result is FAIL                          (7322)
                  END IF

                  IF BYTE_COUNT != 2
                        THEN Test result is FAIL                          (7323)
                  END IF

                  PV_DVAR = SV_DVAR = TV_DVAR = QV_DVAR = 244
                  SEND Command 51 with PV_Dvar, SV_Dvar, TV_Dvar, QV_Dvar
                  IF (RESPONSE_CODE != 2)
                        THEN Test result is FAIL                          (7324)
                  END IF

                  IF BYTE_COUNT != 2
```

```
                        THEN Test result is FAIL                        (7325)
                  END IF

                  IF (PROFILE == WIRELESS) THEN
                        PV_DVAR = SV_DVAR = TV_DVAR = QV_DVAR = 243
                        SEND Command 51 with PV_Dvar, SV_Dvar, TV_Dvar, QV_Dvar
                        IF (RESPONSE_CODE != 0)
                              THEN Test result is FAIL                  (7326)
                        END IF

                        IF BYTE_COUNT != 6
                              THEN Test result is FAIL                  (7327)
                        END IF

                  END IF


               END IF

      END IF

      END TEST
```

FindMappableDevVar(slot, vMap, list, failurepoint)

This procedure is unique to CAL051. Find a mappable Device Variable for the slot indicated in Command 51

```
      PROCEDURE FindMappableDevVar(slot, vMap, list, FAILUREPOINT)
      SET dVar = 0
      SEND Command 0 to read maxDeviceVars
      DO
            Set vMap[slot] = dVar
            retVal = IssueCmd51(vMap, failurepoint)             (FAILUREPOINT)
            IF (retVal == "Success"
                  ADD dVar to list
                  IF (dVar > maxDeviceVars)
                        THEN Test result is FAIL              (FAILUREPOINT+4)
                  INCREMENT dVar
                  CALL VerifyDeviceVariableMapping(n,vmap,(FAILUREPOINT+5))
            END IF
      WHILE (dVar < 250)

      PROCEDURE END
```

IssueCmd51(failurepoint)

This procedure is unique to CAL051.

```
PROCEDURE IssueCmd51(vMap, FAILUREPOINT)
DO
      SEND Command 1
      SEND Command 51 to write dynamic variable map
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                         (FAILUREPOINT)
      END IF
WHILE (RESPONSE_CODE == "Busy")
CALL TestValidFrame
IF ( (RESPONSE_CODE == "Invalid Selection")
      IF (BYTE_COUNT != 2) )
            THEN Test result is FAIL                       (FAILUREPOINT+1)
      ELSE
            RETURN "Bad Map"
      END IF
ELSE IF (RESPONSE_CODE != "Success")
      THEN Test result is FAIL                             (FAILUREPOINT+2)
ELSE IF (BYTE_COUNT != 6)
      THEN Test result is FAIL                             (FAILUREPOINT+3)
ELSE
      RETURN "Success"
END IF

PROCEDURE END
```

VerifyDeviceVariableMapping(vMap, failurepoint)

This procedure is unique to CAL051. It uses command 50 to verify that the mappings are set as expected. The procedure loops on the command 50 until the DUT returns a non-"Busy" response to the command.

```
PROCEDURE VerifyDeviceVariableMapping(n, vmap, FAILUREPOINT)
DO
      SEND Command 1
      SEND Command 50 to read varMap
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                         (FAILUREPOINT)
      END IF
WHILE (RESPONSE_CODE == "Busy")

CALL VerifyResponseAndByteCount(0, 6)
IF (vMap != varMap)
      THEN Test result is FAIL                             (FAILUREPOINT+1)
END IF
PROCEDURE END
```

7.20 CAL052 Set Device Variable Zero

Verifies that the DUT responds properly to Command 52. Checks Addresses, Command Number, Response Code and Byte Count for Command 52, Set Device Variable Zero.

Command 52 is sent to the device with no data bytes. This is followed by Command 52 with the data byte equal to 255.

A message is then sent to the user to input the Device Variable code number that will be used to test the command.

A message is then sent to the user to establish a sensor input close to zero measurement (the value must be close enough to the zero measurement to allow the device to re-zero its calibration but greater than 1% of its normal measurement span from its present calibration). The user indicates when ready. Then Command 33 is sent to read the Device value. Command 52 is then sent to re-zero the device followed by a Command 33. If a "Busy" Response Code is detected, Command 33 will be repeated until the device no longer responds with a "Busy".

A message is sent to the user to change the sensor input to a value that is too high for the device to re-zero its calibration. The user indicates when ready. Then Command 33 is sent to read the Device value. Command 52 is sent to attempt to re-zero the device followed by a Command 33. If a "Busy" Response Code is detected, Command 33 will be repeated until the device no longer responds with a "Busy".

A message is sent to the user to change the sensor input to a value that is too low for the device to re-zero its calibration. The user indicates when ready. Then Command 33 is sent to read the Device value. Command 52 is sent to attempt to re-zero the device followed by a Command 33. If a "Busy" Response Code is detected, Command 33 will be repeated until the device no longer responds with a "Busy".

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.8, 7.20

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(52)
CALL VerifyNotWriteProtected()
```

Send command 52 with data byte = 255

```
SEND Command 52 with one data byte = 255
CALL VerifyResponseAndByteCount(2, 2)
CALL TestValidFrame
```

Get a device variable number

```
dVar = -1
IF (FindNextDeviceVariableToZero(dVar) == "No More Device Variables")
      THEN Test result is FAIL                                  (6674)
END IF
```

Zero measurement

```
DO
      Prompt user: "Establish sensor input close to zero measurement.  The
            value must be close enough to the zero measurement to allow the
            device to re-zero its calibration but greater than 1% of its
            normal measurement span from its present calibration."
      SET x = GetVariableValue(dVar)
      SEND Command 52 with variable code dVar
      CALL VerifyResponseAndByteCount(0, 3)
      CALL TestValidFrame
      SET y = GetVariableValue(dVar)
      IF y is not close to 0.0
            THEN Test result is FAIL                            (6675)
      END IF
```

High value

```
      Prompt user: "Change the sensor input to a value that is too high for
            the device to re-zero its calibration."
      SET x = GetVariableValue(dVar)
      SEND Command 52 with variable code dVar
      IF (RESPONSE_CODE != "Applied Process Too High") THEN
            THEN Test result is FAIL                            (6676)
      ELSE
            CALL VerifyResponseAndByteCount("Applied Process Too High, 2)
            CALL TestValidFrame
            SET y = GetVariableValue(dVar)
            IF y is not close to x
                  THEN Test result is FAIL                      (6677)
            END IF
      END IF
```

Low value

```
      Prompt user: "Change the sensor input to a value that is too low for
            the device to re-zero its calibration."
      SET x = GetVariableValue(dVar)
      SEND Command 52 with variable code dVar
      IF (RESPONSE_CODE != "Applied Process Too Low") THEN
            THEN Test result is FAIL                            (6678)
      ELSE
            CALL VerifyResponseAndByteCount("Applied Process Too Low", 2)
            CALL TestValidFrame
            SET y = GetVariableValue(dVar)
            IF y is not close to x
                  THEN Test result is FAIL                      (6679)
            END IF
      END IF
WHILE (FindNextDeviceVariableToZero(dVar) == "Device Variable Found")

END TEST
```

GetVariableValue(n)

This function is unique to CAL052. It is used to get the value of the variable associated with code n from the device. If the device is "Busy", it is polled until it returns a value.

```
PROCEDURE GetVariableValue(n)
LOOP
      SEND Command 33 with variable code value = n in slot 0
      IF response != "Update Failure") THEN
            EXIT LOOP
      END IF
      CALL VerifyResponseAndByteCount("Update Failure", 8)
      CALL TestValidFrame
LOOP END
CALL VerifyResponseAndByteCount(0, 6)
CALL TestValidFrame
RETURN value in slot zero (bytes 4-7)
PROCEDURE END
```

FindNextDeviceVariableToZero(dVar)

Find a Command 52 supported Device Variable

```
PROCEDURE FindNextDeviceVariableToZero(dVar)
DO
      SET dVarFound = FALSE;
      INCREMENT dVar
      IF (dVar > 249) THEN
            RETURN "No More Device Variables"
      END IF
      SEND Command 52 with one byte = dVar
      CALL TestValidFrame

      IF ( (RESPONSE_CODE == "Invalid Selection")
            IF (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                          (6672)
            END IF
      ELSE
            SET dVarFound = TRUE:
      END IF
WHILE (!dVarFound)

IF (UNIV_REVISION >= 6)
      SEND Command 0 to read maxDeviceVars
      IF (dVar > maxDeviceVars)
            THEN Test result is FAIL                                (6673)
      ELSE
            RETURN "Device Variable Found"
      END IF
END IF
PROCEDURE END
```

7.21 CAL053 Write Device Variable Units

Verifies that the DUT responds properly to Command 53. Checks Addresses, Command Number, Response Code and Byte Count for Command 53, Write Device Variable Units.

A series of Command 33 is sent with all Device Variable numbers from 0 through 253 to determine the valid Device Variable numbers. For each valid Device Variable, the following sequence of commands is sent.

- A Command 53 for each possible value of Unit Code from 0 through 249.
- A Command 53 with the original Unit Code plus an extra byte (Response Code must be "Success").
- A Command 53 with the Unit Code value set to 255 (Response Code must be "Invalid Unit Code").

In all cases, Command 33 is used to verify the actual Unit Code.

Note 1: "Too Few Data Bytes Received" Response Code is verified in CAL000.

Note 2: Some devices may support PV units of "Special" (code 253). When this occurs, the HCF may grant waiver on Failure Point 7125, 7134, 7144, 7154. To request a waiver provide detailed, written justification for the device's behavior to the HCF prior to submitting your device for registration. The HCF will assess the justification and, when appropriate, grant a waiver.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.8, 7.21

Test Procedure

```
CALL IdentifyDevice
CALL CheckForRecommendedCommand(53)
CALL VerifyNotWriteProtected()
```

If this command is supported, then Command 33 must be supported to allow verification of the Unit Code write.

```
SEND Command 33 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                   (6680)
END IF
```

There must be at least one valid Device Variable (otherwise why would the command be implemented?)

```
dVar = -1
IF (FindNextDeviceVariable(dVar) == "No More Device Variables")
      THEN Test result is FAIL                                   (6681)
END IF
```

Check "Invalid Unit Code" Response

```
      DO
            SEND Command 1
            SEND Command 53 with dVar, unit code = 255
            CALL TestValidFrame()
            IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                                 (6682)
            END IF
      WHILE (RESPONSE_CODE == "Busy")
      CALL VerifyResponseAndByteCount("Invalid Unit Code", 2)
```

For each Device Variable, check all of the unit codes possible

```
      DO
            PRINT "Unit Codes for Device Variable dVar = {"
            SET uCode0 = lastUCode = GetUnitsCode(dVar)
            FOR (uCode = [0-255]
                  DO
                        SEND Command 1
                        SEND Command 53 with dVar, uCode
                        CALL TestValidFrame()
                        IF ( (RESPONSE_CODE == "Busy")
                            AND (BYTE_COUNT != 2) )
                              THEN Test result is FAIL                     (6683)
                        END IF
                  WHILE (RESPONSE_CODE == "Busy")
                  lastUCode = VerifyCommand53Response(dVar, uCode, lastUCode)
            END FOR
            PRINT " } newline"
```

Restore the original unit code

```
            DO
                  SEND Command 1
                  SEND Command 53 with dVar, uCode0 plus one byte
                  CALL TestValidFrame()
                  IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
                        THEN Test result is FAIL                           (6684)
                  END IF
            WHILE (RESPONSE_CODE == "Busy")
            CALL VerifyResponseAndByteCount(0, 4)
            IF (lastUCode == GetUnitsCode(dVar))
                  THEN Test result is FAIL                                 (6685)
            END IF

      WHILE (FindNextDeviceVariable(dVar) == "Device Variable Found")

      END TEST
```

VerifyCommand53Response(vcode, ucode, lastUCode)

This procedure is used only by CAL053. It is used to test the validity of the DUT's response. The Response Codes are checked ; the unit code used by the DUT is verified, and the current units code returned.

```
PROCEDURE VerifyCommand53Response(dVar, uCode, lastUCode)
CALL TestValidFrame
IF (RESPONSE_CODE = 0) THEN
      CALL VerifyResponseAndByteCount(0, 4)
      IF (uCode != GetUnitsCode(dVar) )
            THEN Test result is FAIL                                 (6686)
      END IF
      PRINT " uCode"

      SEND Command 54 to read DVClassification
      IF (RESPONSE_CODE != 0)
            THEN Test result is FAIL                                 (6687)
      END IF
      IF (DVClassification == [1-63] or 240-255)
            THEN Test result is FAIL                                 (6688)
      END IF
      IF ((DVClassification==0) AND (PVUnits>169) AND (PVUnits<220))
            THEN Test result is FAIL                                 (6689)
      END IF
      IF (PVUnits == [250-255] )
            THEN Test result is FAIL                                 (6690)
      END IF
      RETURN uCode

ELSE IF (RESPONSE_CODE = "Invalid Units Code")
      CALL VerifyResponseAndByteCount("Invalid Units Code", 2)
      IF (lastUCode != GetUnitsCode(dVar) )
            THEN Test result is FAIL                                 (6691)
      END IF
      RETURN lastUCode
ELSE
      Test result is FAIL                                            (6692)
END IF
PROCEDURE END
```

GetUnitsCode(dVar)

This function is unique to CAL053. It is used to get the units code associated with Device Variable code dVar.

```
PROCEDURE GetUnitsCode(dVar)
SEND Command 33 (with one byte = dVar)
CALL TestValidFrame
IF ( (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Update Failure") )
      THEN Test result is FAIL                                       (6693)
ELSE IF (BYTE_COUNT != 8)
      THEN Test result is FAIL                                       (6694)
END IF
RETURN (units code from Command 33)

PROCEDURE END
```

7.22 CAL054 Read Device Variable Information

Verifies that the DUT responds properly to Command 54. Checks Addresses, Command Number, Response Code and Byte Count for Command 54, Read Device Variable Information.

A series of Command 54 is sent with all Device Variable numbers from 0 incrementing the variable number until a Response Code of 0 is returned. Then command 54 is sent with the valid Device Variable code and an extra byte.

The Device Variable code is incremented again and the tests repeated until all possible values of Device Variable code have been used from 0 to 254.

This Command must be implemented in devices supporting device variables and devices supporting burst mode.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

References:

Specification	**Rev.**	**Sections**
Common Practice Command Specification	7.0	6.8, 7.22

Test Procedure

```
      CALL IdentifyDevice
      SEND Command 54 with no data bytes
      IF (RESPONSE_CODE == "Command not Implemented")
            PRINT "Warning, Implementation of Command 54 is strongly
                  recommended.  This command is implemented by most
                  Field Devices and widely used in Host Applications."
      END IF
```

There must be at least one valid Device Variable (otherwise why would the command be implemented?)

```
      dVar = -1
      IF (FindNextDeviceVariable(dVar) == "No More Device Variables")
            THEN Test result is FAIL                                      (6695)
      END IF
      IF (UNIV_REVISION > 6) THEN
            SET Cmd54BC = 29
      ELSE IF (UNIV_REVISION = 6) THEN
            SET Cmd54BC = 25
      ELSE
            SET Cmd54BC = 23
      END IF
```

For each Device Variable, check Command 54 response

```
      DO
            SEND Command 54 with dVar
            CALL TestValidFrame
            CALL VerifyResponseAndByteCount(0, Cmd54BC)

            SEND Command 54 with dVar and an extra data byte
            CALL TestValidFrame
            CALL VerifyResponseAndByteCount(0, Cmd54BC)

            IF (UNIV_REVISION > 6) THEN
```

```
                  TimeBetweenUpdates = Command54.UpdateTimePeriod

                  IF ((TimeBetweenUpdates < 15 minutes) AND
                    ((TimeBetweenUpdates > 1s) AND
                    (Physical Layer = FSK)))THEN
                        SEND Command 9 with dVar
                        dVarPrim = dVar.Value
                  END IF
```

Request device variable data during the update period, if possible.

```
                  updateFail = 0
                  IF TimeBetweenUpdates > 1s THEN
                        FOR n = 0 to 5
                              SEND Command 9 with dVar
                              IF (ResponseCode == update Failure)
                                    THEN INCREMENT updateFail
                              END IF
                        END FOR

                        IF updateFail == 0
                              THEN Test result is FAIL                     (6696)
                        END IF
                  END IF
```

Request data after update period to verify it has updated.

```
                  Wait for TimeBetweenUpdates

                  SEND Command 9 with dVar
                  IF (dVarPrim == dVar.Value)
                        THEN Test result is FAIL                           (6697)
                  END IF

            END IF

      WHILE (FindNextDeviceVariable(dVar) == "Device Variable Found")

      END TEST
```

Messages

A list of accepted Device Variable codes shall be included in the report.

7.23 CAL055 Write Device Variable Damping Value

Verifies that the DUT responds properly to Command 55. Checks Addresses, Command Number, Response Code and Byte Count for Command 55, Write Device Variable Damping Value.

Since no prior knowledge of the DUT is assumed, Command 33 is used to identify the Device Variables supported by the DUT. Then, for each Device Variable supported, Command 54 is used to retrieve the Damping Value. Operation of Command 55 is confirmed for the Device Variable. Once verification of the Command 55 implementation is confirmed for a Device Variable, the next Device Variable is located until all channels have been processed.

A variety of Response Codes are available for this command. A sequence of Damping values is sent to simulate all of the possible conditions. Table 14 summarizes the conditions tested and the legal DUT responses. To allow for different implementations any one of the legal responses demonstrates compliance with the Command Specification. For example, Test Case 2 allows either of two valid responses.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

Table 14 Command 55: Damping Value Test Cases

	Input	**Output**			
	Damping	**Damping**	**BC**	**Result**	**Response Code**
1	damp0	damp0	6	*Success*	
2	1E30	U/C	2	*Error*	*Passed Parameter Too Large*
		NEW Damping	6	*Warning*	*Set to Nearest Possible Value*
3	-1.00	U/C	2	*Error*	*Passed Parameter Too Small*
		NEW Damping	6	*Warning*	*Set to Nearest Possible Value*
4	damp0 + .00001	NEW Damping	6	*Success*	
		NEW Damping	6	*Warning*	*Set to Nearest Possible Value*
5	damp0 + 1.00	NEW Damping	6	*Success*	
		U/C	2	*Error*	*Passed Parameter Too Large*
		NEW Damping	6	*Warning*	*Set to Nearest Possible Value*

Note 1:"U/C" indicates that the Damping Value in the DUT is unchanged by the command.

Note 2:The term " damp0" indicates the last Damping Value read using Command 54.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.8, 7.23

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(55)
CALL VerifyNotWriteProtected()
```

If this command is supported, then Command 54 must be supported (otherwise damping can be written but not read!)

```
SEND Command 54 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                              (6700)
END IF
```

There must be at least one valid Device Variable (otherwise why would the command be implemented?)

```
dVar = -1
IF (FindNextDeviceVariable(dVar) == "No More Device Variables")
      THEN Test result is FAIL                                              (6701)
END IF
```

Test each Device Variable

```
DO
      SEND Command 55 with dVar only
      CALL VerifyResponseAndByteCount("Too Few Data Bytes", 2)
```

Make sure this Device Variable is valid for Command 55.

```
      SEND Command 54 (with dVar) to read damp0
      CALL VerifyResponseAndByteCount(0, 25)
      SEND Command 55 (with dVar and damp0)
      CALL TestValidFrame
      IF ( (RESPONSE_CODE == "Invalid Selection")
            IF (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                                  (6702)
            END IF
      ELSE
            CALL VerifyResponseAndByteCount(0, 7)
            CALL VerifyDeviceVariableDamping(dVar)
      END IF
WHILE (FindNextDeviceVariable(dVar) == "Device Variable Found")

END TEST
```

VerifyDeviceVariableDamping(dVar)

This procedure performs each test condition for the designated Device Variable. This is where the real work is done for CAL055. First, determine initial damping value. For each failurepoint the Device Variable causing the problem must be printed along with the failurepoint number.

```
PROCEDURE VerifyDeviceVariableDamping(dVar)
SEND Command 54 (with dVar) to read damp0
CALL VerifyResponseAndByteCount(0, 25)
```

Valid message with an extra data byte.

```
SEND Command 55 (with dVar, damp0) and an extra data byte
CALL VerifyResponseAndByteCount(0, 7)
CALL CompareDeviceVariableDamping(dVar, damp0, failurepoint)    (6705-6706)
```

Message with a very large damping time constant (1E30).

```
SEND Command 55 (with dVar, 1E30)
CALL TestValidFrame
IF (RESPONSE_CODE == "SUCCESS")
      IF (BYTE_COUNT != 7)
            THEN Test result is FAIL                                 (6707)
      END IF
ELSE IF (RESPONSE_CODE == "Passed Parameter Too Large")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (6708)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 7)
            THEN Test result is FAIL                                 (6709)
      END IF
ELSE
      Test result is FAIL                                            (6710)
END IF

IF (RESPONSE_CODE == "SUCCESS")
   OR (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET dlast to the returned Device Variable Damping
ELSE
      SET dlast to damp0
END IF
CALL CompareDeviceVariableDamping(dVar, dlast, failurepoint)    (6711-6712)
```

Message with a negative damping time constant (-1).

```
SEND Command 55 (with dVar, -1)
CALL TestValidFrame
IF (RESPONSE_CODE == "Passed Parameter Too Small")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (6713)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 7)
            THEN Test result is FAIL                                 (6714)
      END IF
ELSE
      Test result is FAIL                                            (6715)
END IF

IF (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET dlast to the returned Device Variable Damping
END IF
CALL CompareDeviceVariableDamping(dVar, dlast, failurepoint)    (6716-6717)
```

Message with high precision damping value (original damping value +0.000001 sec).

```
SEND Command 55 (with dVar, (damp0 + 0.000001) )
CALL TestValidFrame
IF (RESPONSE_CODE == "SUCCESS")
      IF (BYTE_COUNT != 7)
            THEN Test result is FAIL                                 (6718)
      END IF
ELSE IF (RESPONSE_CODE == "Passed Parameter Too Large")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (6719)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 7)
            THEN Test result is FAIL                                 (6720)
      END IF
ELSE
      Test result is FAIL                                            (6721)
END IF

IF (RESPONSE_CODE == "SUCCESS")
   OR (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET dlast to the returned Device Variable Damping
END IF
CALL CompareDeviceVariableDamping(dVar, dlast, failurepoint)    (6722-6723)
```

Message with original damping value + 1 second.

```
SEND Command 55 (with dVar, (damp0 + 1.0) )
CALL TestValidFrame
IF (RESPONSE_CODE == "SUCCESS")
      IF (BYTE_COUNT != 7)
            THEN Test result is FAIL                               (6724)
      END IF
ELSE IF (RESPONSE_CODE == "Passed Parameter Too Large")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                               (6725)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 7)
            THEN Test result is FAIL                               (6726)
      END IF
ELSE
      Test result is FAIL                                          (6727)
END IF

IF (RESPONSE_CODE == "SUCCESS")
   OR (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET dlast to the returned Device Variable Damping
ELSE
      SET dlast to damp0
END IF
CALL CompareDeviceVariableDamping(dVar, dlast, failurepoint)   (6728-6729)
```

Message with original damping value.

```
SEND Command 55 with damp0
CALL VerifyResponseAndByteCount(0, 7)
CALL CompareDeviceVariableDamping(dVar, damp0, failurepoint)   (6730-6731)
PROCEDURE END
```

CompareDeviceVariableDamping(dVar, damp, failurepoint)

This procedure is unique to CAL055. It uses command 54 to verify that the Device Variable Damping has the expected value. The procedure loops on the command 54 until the DUT returns a non-"Busy" response to the command.

```
PROCEDURE CompareDeviceVariableDamping(dVar, damp, FAILUREPOINT)
DO
      SEND Command 1
      SEND Command 54 (with dVar) to read Device Variable Damping
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                       (FAILUREPOINT)
      END IF
WHILE (RESPONSE_CODE = "Busy")
CALL VerifyResponseAndByteCount(0, 25)

IF Device Variable Damping != damp
      THEN Test result is FAIL                           (FAILUREPOINT+1)
END IF
PROCEDURE END
```

7.24 CAL056 Write Device Variable Transducer Serial Number

Verifies that the DUT responds properly to Command 56. Checks Addresses, Command Number, Response Code and Byte Count for Command 56, Write Device Variable Transducer Serial Number.

Since no prior knowledge of the DUT is assumed, Command 33 is used to identify the Device Variables supported by the DUT. Then, for each Device Variable supported, Command 54 is used to retrieve the Transducer Serial Number. Operation of Command 56 is confirmed for the Device Variable. Once verification of the Command 56 implementation is confirmed for a Device Variable, the next Device Variable is located until all Device Variables have been processed.

Command 54 is sent to determine the initial Transducer Serial Number. Then a sequence of Command 56 master requests are sent to provoke each possible Response Code. After each Command 56 is sent, Command 54 will be sent to read the Transducer Serial Number. If a "Busy" Response Code is detected, Command 54 will be resent until the device no longer responds with a "Busy".

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.8, 7.24

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(56)
CALL VerifyNotWriteProtected()
```

If this command is supported, then Command 54 must be supported (otherwise Transducer Serial Number can be written but not read!)

```
SEND Command 54 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                    (6735)
END IF
```

There must be at least one valid Device Variable (otherwise why would the command be implemented?)

```
dVar = -1
IF (FindNextDeviceVariable(dVar) == "No More Device Variables")
      THEN Test result is FAIL                                    (6736)
END IF
```

Test each Device Variable

```
      DO
            SEND Command 56 with dVar only
            CALL VerifyResponseAndByteCount("Too Few Data Bytes", 2)
```

Read original serial number.

```
            SEND Command 54 (with dVar) to read sn0
            CALL VerifyResponseAndByteCount(0, 25)
            CALL TestValidFrame
```

Make sure this Device Variable is valid for Command 56.

```
            SEND Command 56 (with dVar,  sn0) and one extra data byte
            CALL TestValidFrame
            IF ( (RESPONSE_CODE == "Invalid Selection")
                  IF (BYTE_COUNT != 2) )
                        THEN Test result is FAIL                     (6737)
                  END IF
            ELSE
                  CALL VerifyResponseAndByteCount(0, 6)
                  CALL CompareDeviceVariableSNo(dVar, sn0,      (6738-6739)
                      failurepoint)
```

A valid message is sent containing a three byte serial number.

```
                  SET testsn = 3 bytes of 0x00
                  SEND Command 56 (with dVar,  testsn)
                  CALL VerifyResponseAndByteCount(0, 6)
                  CALL TestValidFrame
                  CALL CompareDeviceVariableSNo(dVar, testsn,   (6740-6741)
                      failurepoint)

                  SET testsn = 3 bytes of 0xF0
                  SEND Command 56 (with dVar,  testsn)
                  CALL VerifyResponseAndByteCount(0, 6)
                  CALL TestValidFrame
                  CALL CompareDeviceVariableSNo(dVar, testsn,   (6742-6743)
                      failurepoint)
```

A valid message is sent with an extra data byte (original serial number).

```
                  SEND Command 56 (with dVar,  sn0) and one extra data byte
                  CALL VerifyResponseAndByteCount(0, 6)
                  CALL TestValidFrame
                  CALL CompareDeviceVariableSNo(dVar, sn0,      (6744-6745)
                      failurepoint)
            END IF

      WHILE (FindNextDeviceVariable(dVar) == "Device Variable Found")

      END TEST
```

CompareDeviceVariableSNo(dVar, sn, failurepoint)

This function is unique to CAL056. Command 54 is used to read the transducer serial number for the Device Variable. Command 54 is called repeatedly, until the DUT is no longer "Busy".

```
PROCEDURE CompareDeviceVariableSNo(dVar, sn, FAILUREPOINT)

DO
      SEND Command 1
      SEND Command 54 to read Device Variable's Transducer Serial Number
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                         (FAILUREPOINT)
      END IF
WHILE (RESPONSE_CODE == "Busy")
CALL VerifyResponseAndByteCount(0, 25)
CALL TestValidFrame
IF Transducer Serial Number != sn
      THEN Test result is FAIL                             (FAILUREPOINT+1)
END IF

PROCEDURE END
```

7.25 CAL057 (Reserved)

Implementation of Common Practice Command 57 is not recommended. As a result, Field Device implementations are not tested.

7.26 CAL058 (Reserved)

Implementation of Common Practice Command 58 is not recommended. As a result, Field Device implementations are not tested.

7.27 CAL059 (Reserved)

Command 59, Write Number of Response Preambles is verified in the Data Link Layer Tests. As a result, it is not included in this Test Specification. All Field Devices must pass the Data Link Layer Tests before undertaking the Common Practice Tests.

7.28 CAL060 Read Analog Channel And Percent Of Range

Verifies that the DUT responds properly to Command 60. Checks Addresses, Command Number, Response Code and Byte Count for Command 60, Read Analog Channel And Percent Of Range.

A normal request and a request containing an extra data byte are sent to the DUT. For each Analog Channel supported, the DUT must answer each request with "Success". Since Command 48 only supports up to 24 Analog Channels, the DUT must be return "Invalid Selection" for Analog Channels 25-249.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.10, 7.28

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(60)
```

There must be at least one valid Analog Channel (otherwise why would the command be implemented?)

```
aChan = -1
IF (FindNextAnalogChannel(aChan) == "No More Analog Channels")
      THEN Test result is FAIL                                 (6750)
END IF
```

Test each Analog Channel. FindNextAnalogChannel checks normal request and the "Invalid Selection" Response Code.

```
PRINT "The Analog Channels supported by this Field Device are = {"
DO
      PRINT " aChan"
      SEND Command 60 (with aChan) and one extra data byte
      CALL TestValidFrame
      CALL VerifyResponseAndByteCount(0, 10)
WHILE (FindNextAnalogChannel(aChan) == "Analog Channel Found")
PRINT " } newline"

END TEST
```

7.29 CAL061 (Reserved)

Implementation of Common Practice Command 61 is not recommended. As a result, Field Device implementations are not tested.

7.30 CAL062 Read Analog Channels

Verifies that the DUT responds properly to Command 62. Checks Addresses, Command Number, Response Code and Byte Count for Command 62, Read Analog Channels.

Since no prior knowledge of the analog channel code assignments is assumed, there will always be some set(s) of analog channel codes which are valid, and some sets of analog channel codes which are invalid. The Analog Channels are scanned until a valid one is found, then a varying number of request data bytes are sent to verify proper Command 62 operation.

Devices are required to reply to requests having more than the required data bytes, so five request data bytes are sent and a "Success" Response Code is required.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.10, 7.30

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(62)
```

There must be at least one valid Analog Channel (otherwise why would the command be implemented?)

```
aChan = -1
IF (FindNextAnalogChannel(aChan) == "No More Analog Channels")
      THEN Test result is FAIL                                    (6755)
END IF
```

Send Command 62 with varying number of data bytes. Check command response length

```
DO
      FOR ( nBytes = [1 – 5] )
            SEND Command 62 (with nBytes of aChan)
            CALL TestValidFrame
            IF ( (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Update
                  Failure") )
                  THEN Test result is FAIL                        (6756)
            ELSE
                  IF (BYTE_COUNT != 26)
                        THEN Test result is FAIL                  (6757)
                  END IF
            END IF
      END FOR
WHILE (FindNextAnalogChannel(aChan) == "Analog Channel Found")

SEND Command 62 (with 4 bytes of 255)
CALL TestValidFrame
CALL VerifyResponseAndByteCount("Invalid Selection", 2)

END TEST
```

7.31 CAL063 Read Analog Channel Information

Verifies that the DUT responds properly to Command 63. Checks Addresses, Command Number, Response Code and Byte Count for Command 63, Read Analog Channel Information.

Devices are required to reply to requests having more than the required data bytes, so 2 request data bytes are sent and a "Success" Response Code is required.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.10, 7.31

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(63)
IF (UNIV_REVISION >= 6) THEN
      SET Cmd63BC = 19
ELSE
      SET Cmd63BC = 18
END IF
```

There must be at least one valid Analog Channel (otherwise why would the command be implemented?)

```
aChan = -1
IF (FindNextAnalogChannel(aChan) == "No More Analog Channels")
      THEN Test result is FAIL                                          (6760)
END IF
```

Test each Analog Channel.

```
PRINT "The Analog Channels supported by this Field Device are = {"
DO
      PRINT " aChan"
      SEND Command 63 (with aChan)
      CALL TestValidFrame
      CALL VerifyResponseAndByteCount(0, Cmd63BC)

      SEND Command 63 (with aChan) and an extra data byte
      CALL TestValidFrame
      CALL VerifyResponseAndByteCount(0, Cmd63BC)

WHILE (FindNextAnalogChannel(aChan) == "Analog Channel Found")
PRINT " } newline"

SEND Command 63 (with 255)
CALL TestValidFrame
CALL VerifyResponseAndByteCount("Invalid Selection", 2)

END TEST
```

7.32 CAL064 Write Analog Channel Additional Damping Value

Verifies that the DUT responds properly to Command 64. Checks Addresses, Command Number, Response Code and Byte Count for Command 64, Write Analog Channel Additional Damping Value.

Since no prior knowledge of the DUT is assumed, Command 60 is used to identify the Analog Channels supported by the DUT. Then, for each Analog Channel supported, Command 63 is used to retrieve the Damping Value. Operation of Command 64 is confirmed for the Analog Channel. Once verification of the Command 64 implementation is confirmed for an Analog Channel, the next Analog Channel is located until all channels have been processed.

This command allows some flexibility in Field Device implementation. For example, if an invalid Damping Value is received ,an implementation can simply return an error (i.e., the Damping Value was no t accepted). Alternatively, some implementations may choose to force the invalid Damping Value to a good value and return a warning. This test allows for either implementation.

A variety of Response Codes are available for this command. A sequence of Damping values are sent to simulate all of the possible conditions. Table 15 summarizes the conditions tested and the legal DUT responses. To allow for different implementations, any one of the legal responses demonstrates compliance with the Command Specification. For example, Test Case 2 allows either of two valid responses.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

Table 15 Command 64: Damping Value Test Cases

	Input	**Output**			
	Damping	**Damping**	**BC**	**Result**	**Response Code**
1	damp0	damp0	6	*Success*	
2	1E30	U/C	2	*Error*	*Passed Parameter Too Large*
		NEW Damping	6	*Warning*	*Set to Nearest Possible Value*
3	-1.00	U/C	2	*Error*	*Passed Parameter Too Small*
		NEW Damping	6	*Warning*	*Set to Nearest Possible Value*
4	damp0 + .00001	NEW Damping	6	*Success*	
		NEW Damping	6	*Warning*	*Set to Nearest Possible Value*
5	damp0 + 1.00	NEW Damping	6	*Success*	
		U/C	2	*Error*	*Passed Parameter Too Large*
		NEW Damping	6	*Warning*	*Set to Nearest Possible Value*

Note 1:"U/C" indicates that the Damping Value in the DUT is unchanged by the command.
Note 2:The term " damp0" indicates the last Damping Value read using Command 63.

References:

Specification	**Rev.**	**Sections**

Specification	**Rev.**	**Sections**
Common Practice Command Specification	7.0	6.10, 7.32

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(64)
CALL VerifyNotWriteProtected()
```

If this command is supported, then Command 63 must be supported (otherwise damping can be written but not read!)

```
SEND Command 63 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
    THEN Test result is FAIL                                        (6770)
END IF
```

There must be at least one valid Analog Channel (otherwise why would the command be implemented?)

```
aChan = -1
IF (FindNextAnalogChannel(aChan) == "No More Analog Channels")
    THEN Test result is FAIL                                        (6771)
END IF
```

Test each Analog Channel.

```
DO
    SEND Command 64 (with aChan only)
    CALL VerifyResponseAndByteCount("Too Few Data Bytes", 2)

    SEND Command 63 (with aChan ) to read damp0
    CALL VerifyResponseAndByteCount(0, 19)
```

Make sure this Analog Channel is valid for Command 64.

```
    SEND Command 64 (with aChan and damp0)
    CALL TestValidFrame
    IF ( (RESPONSE_CODE == "Invalid Selection")
        IF (BYTE_COUNT != 2) )
            THEN Test result is FAIL                                (6772)
        END IF
    ELSE
        CALL VerifyResponseAndByteCount(0, 7)
        CALL VerifyAnalogChannelDamping(aChan)
    END IF

WHILE (FindNextAnalogChannel (aChan) == "Analog Channel Found")

END TEST
```

VerifyAnalogChannelDamping(aChan)

This procedure performs each test condition for the designated Analog Channel. This is where the real work is done for CAL064. First, determine initial damping value

```
PROCEDURE VerifyAnalogChannelDamping(aChan)
SEND Command 63 (with aChan ) to read damp0
CALL VerifyResponseAndByteCount(0, 25)
```

Valid message with an extra data byte.

```
SEND Command 64 (with aChan , damp0) and an extra data byte
CALL VerifyResponseAndByteCount(0, 7)
CALL CompareAnalogChannelDamping(aChan, damp0, failurepoint)    (6773-6774)
```

Message with a very large damping time constant (1E30).

```
SEND Command 64 (with aChan , 1E30)
CALL TestValidFrame
IF (RESPONSE_CODE == "Passed Parameter Too Large")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (6775)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 7)
            THEN Test result is FAIL                                 (6777)
      END IF
ELSE
      Test result is FAIL                                            (6778)
END IF

IF (RESPONSE_CODE == "Set To Nearest Possible Value") THEN
      SET dlast to the returned Device Variable Damping
ELSE
      SET dlast to damp0
END IF
CALL CompareAnalogChannelDamping(aChan, dlast, failurepoint)    (6779-6780)
```

Message with a negative damping time constant (-1).

```
SEND Command 64 (with aChan , -1)
CALL TestValidFrame
IF (RESPONSE_CODE == "Passed Parameter Too Small")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (6781)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 7)
            THEN Test result is FAIL                                 (6782)
      END IF
ELSE
      Test result is FAIL                                            (6783)
END IF

IF (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET dlast to the returned Device Variable Damping
END IF
CALL CompareAnalogChannelDamping(aChan, dlast, failurepoint)    (6784-6785)
```

Message with high precision damping value (original damping value +0.000001 sec).

```
SEND Command 64 (with aChan , (damp0 + 0.000001) )
CALL TestValidFrame
IF (RESPONSE_CODE == "SUCCESS")
      IF (BYTE_COUNT != 7)
            THEN Test result is FAIL                                 (6786)
      END IF
ELSE IF (RESPONSE_CODE == "Passed Parameter Too Large")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (6787)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 7)
            THEN Test result is FAIL                                 (6788)
      END IF
ELSE
      Test result is FAIL                                            (6789)
END IF
IF (RESPONSE_CODE == "SUCCESS")
   OR (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET dlast to the returned Device Variable Damping
END IF
CALL CompareAnalogChannelDamping(aChan, dlast, failurepoint)    (6790-6791)
```

Message with original damping value + 1 second.

```
SEND Command 64 (with aChan , (damp0 + 1.0) )
CALL TestValidFrame
IF (RESPONSE_CODE == "SUCCESS")
      IF (BYTE_COUNT != 7)
            THEN Test result is FAIL                                 (6792)
      END IF
ELSE IF (RESPONSE_CODE == "Passed Parameter Too Large")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (6793)
      END IF
ELSE IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 7)
            THEN Test result is FAIL                                 (6794)
      END IF
ELSE
      Test result is FAIL                                            (6795)
END IF
IF (RESPONSE_CODE == "SUCCESS")
   OR (RESPONSE_CODE == "Set To Nearest Possible Value") ) THEN
      SET dlast to the returned Device Variable Damping
ELSE
      SET dlast to damp0
END IF
CALL CompareAnalogChannelDamping(aChan, dlast, failurepoint)    (6796-6798)
```

Message with original damping value.

```
SEND Command 55 with damp0
CALL VerifyResponseAndByteCount(0, 7)
CALL CompareAnalogChannelDamping(aChan, damp0, failurepoint)    (6799-6800)
PROCEDURE END
```

CompareAnalogChannelDamping(aChan, damp, failurepoint)

This procedure is unique to CAL064. It uses command 63 to verify that the Device Variable Damping has the expected value. The procedure loops on the command 63 until the DUT returns a non-"Busy" response to the command.

```
PROCEDURE CompareAnalogChannelDamping(aChan, damp, FAILUREPOINT)
DO
      SEND Command 1
      SEND Command 63 (with aChan) to read Analog Channel Damping
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                         (FAILUREPOINT)
      END IF
WHILE (RESPONSE_CODE == "Busy")
CALL VerifyResponseAndByteCount(0, 19)

IF Analog Channel Damping != damp
      THEN Test result is FAIL                               (FAILUREPOINT+1)
END IF

PROCEDURE END
```

7.33 CAL065 Write Analog Channel Range Values

Verifies that the DUT responds properly to Command 65. Checks Addresses, Command Number, Response Code and Byte Count for Command 65, Write Analog Channel Range Values.

Since no prior knowledge of the DUT is assumed, Command 60 is used to identify the Analog Channels supported by the DUT. Then, for each Analog Channel supported, Command 63 is used to retrieve the Range Values and Units Code. Operation of Command 65 is confirmed for the Analog Channel. Once verification of the Command 65 implementation is confirmed for an Analog Channel, the next Analog Channel is located until all channels have been processed.

This command allows some flexibility in Field Device implementation. For example, if an invalid Range Value is received, an implementation can simply return an error (i.e., the Range Value was not accepted). Alternatively, some implementations may choose to force the invalid Range Value to a good value, and return a warning. This test allows for either implementation.

A variety of Response Codes are available for this command. A sequence of range values is sent to simulate all of the possible conditions Table 16 summarizes the conditions tested and the legal DUT responses. To allow for different implementations, any one of the legal responses demonstrates compliance with the Command Specification. For example, Test Case 2 allows either of two valid responses.

Features Tested (see Table 16):

- Data is sent to provoke all allowed Response Codes;
- Command 63 is used to verify writes of Range Values; and
- When "Busy" is detected, Command 1 is sent to confirm proper use of this Response Code.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

Feature NOT Tested:

- The Percent Range is not verified against the Range Values applied; and
- The Unit Code is not varied to confirm that the Device Variable and Dynamic Variable Units are unaffected by Command 65.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.10, 7.33

Table 16 Command 65: Set Range Values Test Cases

	Input		**Output**				
	URV	**LRV**	**URV**	**LRV**	**BC**	**Result**	**Response Code**
1	urv0	lrv0	urv0	lrv0	12	*Success*	
2	1E30	lrv0	U/C	U/C	2	*Error*	*URV Too High*
			NEW URV	lrv0	12	*Warning*	*Set to Nearest Possible Value*
3	-1E30	lrv0	U/C	U/C	2	*Error*	*URV Too Low*
			NEW URV	lrv0	12	*Warning*	*Set to Nearest Possible Value*
4	urv0	1E30	U/C	U/C	2	*Error*	*LRV Too High*
			urv0	NEW LRV	12	*Warning*	*Set to Nearest Possible Value*
5	urv0	-1E30	U/C	U/C	2	*Error*	*LRV Too Low*
			urv0	NEW LRV	12	*Warning*	*Set to Nearest Possible Value*
6	1E30	-1E30	U/C	U/C	2	*Error*	*URV and LRV Out Of Limits*
			U/C	U/C	2	*Error*	*LRV Too Low*
			U/C	U/C	2	*Error*	*URV Too High*
			NEW URV	NEW LRV	12	*Warning*	*Set to Nearest Possible Value*
7	lrv0	lrv0	lrv0	lrv0	12	*Warning*	*Span Too Small*
			U/C	U/C	2	*Error*	*Invalid Span*
			NEW URV	NEW LRV	12	*Warning*	*Set to Nearest Possible Value*

Note 1: "U/C" indicates that the Range Value in the DUT is unchanged by the command.

Note 2: The terms "urv0" and "lrv0" indicates the last Range Value(s) read using Command 63.

Test Procedure

```
      CALL IdentifyDevice
      CALL CheckCommandImplemented(65)
      CALL VerifyNotWriteProtected()
```

If this command is supported, then Command 63 must be supported (otherwise the Range Values can be written but not read!)

```
      SEND Command 63 with no data bytes
      IF (RESPONSE_CODE == "Command not Implemented")
            THEN Test result is FAIL                                       (6805)
      END IF
```

There must be at least one valid Analog Channel (otherwise why would the command be implemented?)

```
      aChan = -1
      IF (FindNextAnalogChannel(aChan) == "No More Analog Channels")
            THEN Test result is FAIL                                       (6806)
      END IF
```

Test each Analog Channel.

```
      DO
            SEND Command 65 (with aChan only)
            CALL VerifyResponseAndByteCount("Too Few Data Bytes", 2)

            SEND Command 63 (with aChan ) to read urv0, lrv0, units0
            CALL VerifyResponseAndByteCount(0, 19)
```

Make sure this Analog Channel is valid for Command 65.

```
            SEND Command 65 (with aChan and urv0, lrv0, units0 )
            CALL TestValidFrame
            IF ( (RESPONSE_CODE == "Invalid Selection")
                  IF (BYTE_COUNT != 2) )
                        THEN Test result is FAIL                           (6807)
                  END IF
            ELSE
                  CALL VerifyResponseAndByteCount(0, 12)
                  CALL VerifyAnalogChannelRange(aChan)
            END IF

      WHILE (FindNextAnalogChannel (aChan) == "Analog Channel Found")

      END TEST
```

VerifyAnalogChannelRange(aChan)

Verifies proper operation of Command 65 for a given analog channel

```
PROCEDURE VerifyAnalogChannelRange(aChan)

SEND Command 63 (with aChan) to read lrv0, urv0, u0
SET lrvOriginial = lrv0; urvOriginal = urv0; unitsOriginal = u0
```

Send a valid message with an extra data byte.

```
SEND Command 65 (with aChan) to set LRV = lrv0, URV = urv0, units = u0,
      with one extra data byte
CALL VerifyResponseAndByteCount(0, 12)
CALL CompareAnalogChannelRange(aChan,lrv0,urv0,u0,failurepoint) (6808-6812)
```

A message with a very large positive upper range value (1E30) and original lower range value.

```
SEND Command 65 (with aChan) to set LRV = lrv0, URV = 1E30, units = u0

CALL TestValidFrame
IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 12)
            THEN Test result is FAIL                                    (6813)
      END IF
ELSE IF (RESPONSE_CODE == "Upper Range Value Too High")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                    (6814)
      END IF
ELSE
      Test result is FAIL                                               (6815)
END IF

IF (RESPONSE_CODE == "Set To Nearest Possible Value") THEN
      SET urv0 to the URV
      SET lrv0 to the LRV
END IF
CALL CompareAnalogChannelRange(aChan,lrv0,urv0,u0,failurepoint) (6816-6820)
```

A message with a very large negative upper range value (-1E30) and original lower range value.

```
SEND Command 65 (with aChan) to set LRV = lrv, URV = -1E30, units = u0

IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 12)
            THEN Test result is FAIL                                    (6821)
      END IF
ELSE IF (RESPONSE_CODE == "Upper Range Value Too Low")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                    (6822)
      END IF
ELSE
      Test result is FAIL                                               (6823)
END IF

IF (RESPONSE_CODE == "Set To Nearest Possible Value") THEN
      SET urv0 to the URV
      SET lrv0 to the LRV
END IF
CALL CompareAnalogChannelRange(aChan,lrv0,urv0,u0,failurepoint) (6825-6829)
```

A message with a very large positive lower range value (1E30) and original upper range value

```
SEND Command 65 (with aChan) to set LRV = 1E30, URV = urv0, units = u0

CALL TestValidFrame
IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 12)
            THEN Test result is FAIL                                 (6831)
      END IF
ELSE IF (RESPONSE_CODE == "Lower Range Value Too High")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (6832)
      END IF
ELSE
      Test result is FAIL                                            (6833)
END IF

IF (RESPONSE_CODE == "Set To Nearest Possible Value") THEN
      SET urv0 to the URV
      SET lrv0 to the LRV
END IF
CALL CompareAnalogChannelRange(aChan,lrv0,urv0,u0,failurepoint) (6835-6839)
```

A message with a very large negative lower range value (-1E30) and original upper range value.

```
SEND Command 65 (with aChan) to set LRV = -1E30, URV = urv0, units = u0

CALL TestValidFrame
IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 12)
            THEN Test result is FAIL                                 (6841)
      END IF
ELSE IF (RESPONSE_CODE == "Lower Range Value Too Low")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                 (6842)
      END IF
ELSE
      Test result is FAIL                                            (6843)
END IF

IF (RESPONSE_CODE == "Set To Nearest Possible Value") THEN
      SET urv0 to the URV
      SET lrv0 to the LRV
END IF
CALL CompareAnalogChannelRange(aChan,lrv0,urv0,u0,failurepoint) (6845-6849)
```

A message with a very large positive upper range value (1E30) and a very large negative lower range value (-1E30)

```
SEND Command 65 (with aChan) to set LRV = -1E30, URV = 1E30, units = u0

CALL TestValidFrame
IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 12)
            THEN Test result is FAIL                                   (6851)
      END IF
ELSE IF (RESPONSE_CODE == "Upper and Lower Range Values Out Of Limits")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                   (6852)
      END IF
ELSE IF (RESPONSE_CODE == "Lower Range Value Too Low")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                   (6853)
      END IF
ELSE IF (RESPONSE_CODE == "Upper Range Value Too High")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                   (6855)
      END IF
ELSE
      Test result is FAIL                                              (6856)
END IF

IF (RESPONSE_CODE == "Set To Nearest Possible Value") THEN
      SET lrv0 to the LRV
      SET urv0 to the URV
END IF
CALL CompareAnalogChannelRange(aChan,lrv0,urv0,u0,failurepoint) (6860-6864)
```

A message with the upper range value equal to the lower range value.

```
SEND Command 65 (with aChan) to set LRV = lrv0, URV = lrv0, units = u0

CALL TestValidFrame
IF (RESPONSE_CODE == "Set To Nearest Possible Value")
      IF (BYTE_COUNT != 12)
            THEN Test result is FAIL                                    (6866)
      END IF
ELSE IF (RESPONSE_CODE == "Span Too Small")
      IF (BYTE_COUNT != 12)
            THEN Test result is FAIL                                    (6867)
      END IF
ELSE IF (RESPONSE_CODE == "Invalid Span")
      IF (BYTE_COUNT != 2)
            THEN Test result is FAIL                                    (6868)
      END IF
ELSE
      Test result is FAIL                                               (6869)
END IF

IF ( (RESPONSE_CODE == "Set To Nearest Possible Value")
   OR (RESPONSE_CODE == "Span Too Small") ) THEN
      SET lrv0 to the LRV
      SET urv0 to the URV
END IF
CALL CompareAnalogChannelRange(aChan,lrv0,urv0,u0,failurepoint) (6870-6874)
```

A message with a units code of 0xFF.

```
SEND Command 65 (with aChan) to set LRV = lrv0, URV = lrv0
      and units = 0xFF.
CALL VerifyResponseAndByteCount(2, 2)
CALL CompareAnalogChannelRange(aChan,lrv0,urv0,u0,failurepoint) (6875-6879)
```

A message with the original range values and units code.

```
SEND Command 65 (with aChan) to set LRV = lrvOriginal,
      URV = lrvOriginal and units = uOriginal.
CALL VerifyResponseAndByteCount(0, 12)
CALL CompareAnalogChannelRange(aChan,lrv0,urv0,u0,failurepoint) (6880-6884)

PROCEDURE END
```

CompareAnalogChannelRange(aChan, lrv, urv, units, failurepoint)

This procedure is unique to CAL065. It uses command 63 to verify that the upper and lower range values are as expected. The procedure loops on the command 63 until the DUT returns a non-"Busy" response to the command.

```
PROCEDURE CompareAnalogChannelRange(aChan, lrv, urv, units, FAILUREPOINT)
DO
      SEND Command 1
      SEND Command 63 (with aChan) to read l, u, un
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                          (FAILUREPOINT)
      END IF
WHILE (RESPONSE_CODE == "Busy")

IF BYTE_COUNT != 19 THEN
      Test result is FAIL                                   (FAILUREPOINT+1)
END IF

IF lrv != l THEN
      Test result is FAIL                                   (FAILUREPOINT+2)
END IF
IF urv != u THEN
      Test result is FAIL                                   (FAILUREPOINT+3)
END IF
IF units != un THEN
      Test result is FAIL                                   (FAILUREPOINT+4)
END IF
PROCEDURE END
```

Messages

In addition, the number of analog channel codes used in the command will be reported.

7.34 CAL066 Enter/Exit Fixed Analog Channel Mode

Verifies that the DUT responds properly to Command 66. Checks Addresses, Command Number, Response Code and Byte Count for Command 66, Enter/Exit Fixed Analog Channel Mode.

Command 66 forces a DUT Analog Channel to the specified value. The Analog Channel is returned to normal operation by sending a value of "0x7F, 0xA0, 0x00, 0x00" to the DUT. When the Analog Channel is fixed, the appropriate status bits (either Device Status or Command 48 Status) must be set. Command 60 is used to identify the Analog Channels in the DUT. Table 11 summarizes the conditions used to test the DUT's response to Command 66.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

Table 17 Command 66 Test Cases

Case	Command 66	Response Code	Channel Fixed
1	MRV + Extra Data Byte	Success	Yes
2	Mid-Range value	Success	Yes
3	2*UEP -LEP	"Passed Parameter Too Large"	Yes
4	2*LEP - UEP	"Passed Parameter Too Small"	Yes
5	Units = 255	"Invalid Unit Code"	Yes
6	0x7F, 0xA0, 0x00, 0x00	Success	No

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(66)
CALL VerifyNotWriteProtected()
```

If this command is supported, then Command 48 and 70 must be supported (otherwise the minimum and maximum values for the Analog Channel cannot be read!)

```
SEND Command 48 IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                          (6890)
ELSE IF (BYTE_COUNT < 16) THEN
      Test result is FAIL                                               (6891)
END IF

SEND Command 70 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                          (6892)
END IF
```

There must be at least one valid Analog Channel (otherwise why would the command be implemented?)

```
aChan = -1
IF (FindNextAnalogChannel(aChan) == "No More Analog Channels")
      THEN Test result is FAIL                                            (6893)
END IF
```

Test each Analog Channel.

```
DO
      SEND Command 66 (with aChan only)
      CALL VerifyResponseAndByteCount("Too Few Data Bytes", 2)

      SEND Command 60 (with aChan ) to read most recent aValue
      CALL VerifyResponseAndByteCount(0, 12)
```

Make sure this Analog Channel is valid for Command 66

```
      SEND Command 66 (with aChan and aValue) and an extra data byte
      CALL TestValidFrame
      IF ( (RESPONSE_CODE == "Invalid Analog Channel Code Number")
            IF (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                                (6894)
            END IF
      ELSE
            CALL VerifyResponseAndByteCount(0, 8)
            CALL VerifyWriteAnalogChannel (aChan)
      END IF

WHILE (FindNextAnalogChannel (aChan) == "Analog Channel Found")
```

VerifyWriteAnalogChannel(aChan)

Verifies proper operation of Command 66 for a given analog channel

```
PROCEDURE VerifyWriteAnalogChannel(aChan)

SEND Command 70 (with aChan) to read uEndPoint, lEndPoint, units
CALL TestValidFrame
CALL VerifyResponseAndByteCount(0, 12)

SET delta = uEndPoint - lEndPoint

SEND Command 60 (with aChan ) to read originalValue
CALL VerifyResponseAndByteCount(0, 12)
```

Verify that a mid-range value is valid.

```
WriteAnalogChannelValue(aChan, aValue=(lEndPoint+(0.5*delta)), units )
CALL VerifyResponseAndByteCount(0, 8)
CALL CompareAnalogChannelValue(aChan, aValue, failurepoint)     (6895-6896)
SET lastValue = aValue
```

Verify aChan status.

```
      IF (aChan == 0) THEN
            IF (DEVICE_STATUS != "Loop Current Fixed")
                  THEN Test Result is FAIL                                  (6897)
            END IF
```

Since this is not the PV then Command 48 is tested for status.

```
      ELSE
            SEND Command 48 with no data bytes
            IF (analogChannelFixed[aChan] is not set)
                  THEN Test Result is FAIL                                  (6898)
            END IF
      END IF
```

A message with an over-range value.

```
      WriteAnalogChannelValue(aChan, aValue=(uEndPoint+delta), units )
      CALL VerifyResponseAndByteCount("Passed Parameter Too Large", 2)
      CALL CompareAnalogChannelValue(aChan, lastValue, failurepoint)  (6900-6901)
```

A message with an under-range value.

```
      WriteAnalogChannelValue(aChan, aValue=(lEndPoint-delta), units )
      SEND Command 66 (with aChan) to set aValue =
      CALL TestValidFrame
      CALL VerifyResponseAndByteCount("Passed Parameter Too Small", 2)
      CALL CompareAnalogChannelValue(aChan, lastValue, failurepoint)  (6902-6903)
```

A message with a units code of 0xFF.

```
      WriteAnalogChannelValue(aChan, lastValue, units = 0xFF)
      CALL VerifyResponseAndByteCount("Invalid Units Code", 2)
      CALL CompareAnalogChannelValue(aChan, lastValue, failurepoint)  (6904-6905)
```

A message to remove from fixed value mode (NaN).

```
      WriteAnalogChannelValue(aChan, aValue="0x7F,0xA0,0x00,0x00", units)
      CALL VerifyResponseAndByteCount(0, 8)
```

Verify aChan status.

```
      IF (aChan == 0) THEN
            IF (DEVICE_STATUS == "Loop Current Fixed")
                  THEN Test Result is FAIL                                  (6906)
            END IF
```

Since this is not the PV, then Command 48 is tested for status.

```
      ELSE
            SEND Command 48
            IF (analogChannelFixed[aChan] is set)
                  THEN Test Result is FAIL                                  (6907)
            END IF
      END IF
```

Verify special requirements for aChan 0 i.e., PV.

```
IF (aChan == 0) THEN
```

Command 6 changing the poll address to 1.

```
      SEND Command 6 with polling address 0 and loop current signaling
            disabled
      IF ( UNIV_REVISION >= 6 )
            CALL VerifyResponseAndByteCount(0, 4)
            IF (DEVICE_STATUS != "Loop Current Fixed")
                  THEN Test Result is FAIL                           (6910)
            END IF
      ELSE
            THEN CALL VerifyResponseAndByteCount(0, 3)
      END IF
```

Command 66 with valid mid-range value.

```
      WriteAnalogChannelValue(aChan,
          aValue=(lEndPoint+(0.5*delta)), units )
      CALL VerifyResponseAndByteCount("In Multidrop Mode", 2)
      IF (DEVICE_STATUS != "Loop Current Fixed")
            THEN Test Result is FAIL                                 (6911)
      END IF
```

Command 6 changing the poll address to 0.

```
      SEND Command 6 with polling address 0 and loop current signaling
            enabled
      IF ( UNIV_REVISION >= 6 )
            CALL VerifyResponseAndByteCount(0, 4)
            IF (DEVICE_STATUS == "Loop Current Fixed")
                  THEN Test Result is FAIL                           (6912)
            END IF
      ELSE
            THEN CALL VerifyResponseAndByteCount(0, 3)
      END IF
END IF
PROCEDURE END
```

WriteAnalogChannelValue(aChan, aValue, Units)

This procedure is unique to CAL066. This issues Command 66 taking into account "Busy"

```
PROCEDURE WriteAnalogChannelValue(aChan, aValue, Units)
DO
      SEND Command 60 (with aChan)
      CALL VerifyResponseAndByteCount(0, 12)
      SEND Command 66 (with aChan, aValue, Units)
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                                 (6913)
      END IF
WHILE (RESPONSE_CODE == "Busy")
CALL TestValidFrame
PROCEDURE END
```

Messages

In addition, the number of analog channel codes used in the command will be reported.

7.35 CAL067 Trim Analog Channel Zero

Verifies that the DUT responds properly to Command 67. Checks Addresses, Command Number, Response Code and Byte Count for Command 67, Trim Analog Channel Zero.

Since no prior knowledge of the DUT is assumed, Command 60 is used to identify the Analog Channels supported by the DUT. Then, for each Analog Channel supported, Command 70 is used to retrieve the Upper and Lower End-Points. Operation of Command 67 is confirmed for the Analog Channel. Once verification of the Command 67 implementation is confirmed for an Analog Channel, the next Analog Channel is located until all channels have been processed.

A variety of Response Codes are available for this command. A sequence of Analog Channel settings and commands are sent to provoke all of the possible conditions. Table 18 summarizes the conditions tested and the legal DUT responses.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

Table 18 Command 67 Test Cases

Case	Measured Value	Command 66	Response Code
1	LEP	LEP	"Success"
2	LEP	LEP + Extra Data Byte	"Success"
3	LEP	110% UEP	"Passed Parameter Too Large"
4	LEP	100% Less Than LEP	"Passed Parameter Too Small"
5	UEP	LEP	"Not In Proper Analog Channel Mode"
6		Unit Code = 255	"Invalid Units Code"

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.10, 7.35

Test Procedure

```
      CALL IdentifyDevice
      CALL CheckCommandImplemented(67)
      CALL VerifyNotWriteProtected()
```

If this command is supported, then Command 70 must be supported (otherwise the minimum and maximum values for the Analog Channel cannot be read!)

```
      SEND Command 70 with no data bytes
      IF (RESPONSE_CODE == "Command not Implemented")
            THEN Test result is FAIL                                       (6915)
      END IF
```

If this command is supported, then Command 63 must be supported (otherwise whether the Analog Channel is an input to the DUT cannot be read!)

```
      SEND Command 63 with no data bytes
      IF (RESPONSE_CODE == "Command not Implemented")
            THEN Test result is FAIL                                       (6916)
      END IF
```

If this command is supported, then Command 60 must be supported (otherwise the Analog Channel value cannot be read!)

```
      SEND Command 60 with no data bytes
      IF (RESPONSE_CODE == "Command not Implemented")
            THEN Test result is FAIL                                       (6917)
      END IF
```

There must be at least one valid Analog Channel (otherwise why would the command be implemented?)

```
      aChan = -1
      IF (FindNextAnalogChannel(aChan) == "No More Analog Channels")
            THEN Test result is FAIL                                       (6918)
      END IF
```

Test each Analog Channel.

```
      DO
            SEND Command 67 (with aChan only)
            CALL VerifyResponseAndByteCount("Too Few Data Bytes", 2)

            SEND Command 60 (with aChan ) to read most recent aValue
            CALL VerifyResponseAndByteCount(0, 12)
```

Make sure this Analog Channel is valid for Command 67

```
            SEND Command 67 (with aChan and aValue) and an extra data byte
            CALL TestValidFrame
            IF ( (RESPONSE_CODE == "Invalid Analog Channel Code Number")
                  IF (BYTE_COUNT != 2) )
                        THEN Test result is FAIL                           (6919)
                  END IF
            ELSE
                  CALL VerifyZeroAnalogChannel (aChan)
            END IF

      WHILE (FindNextAnalogChannel (aChan) == "Analog Channel Found")

      END TEST
```

VerifyZeroAnalogChannel(aChan)

Steps through all the test conditions for a single Analog Channel

```
PROCEDURE VerifyZeroAnalogChannel(aChan)

SEND Command 70 (with aChan) to read uEndPoint, lEndPoint, units
CALL TestValidFrame
CALL VerifyResponseAndByteCount(0, 12)

SET delta = uEndPoint - lEndPoint
```

Set the Analog Channel to the Lower End Point

```
IF (UNIV_REVISION >= 6) THEN
      SEND Command 63 to read PV Analog Channel Flags
ELSE
      Prompt user to set Analog Channel Flags
END IF
IF ("Analog Input Channel")
      Prompt user: "Set Analog Input <aChan> to <lEndPoint>"
ELSE
      SEND Command 66 (with aChan, aValue = lEndPoint)
      CALL VerifyResponseAndByteCount(0, 8)
END IF
CALL CompareAnalogChannelValue(aChan, lEndPoint, failurepoint)  (6925-6926)
```

A valid Command 67 message is sent with a current value of <lEndPoint> .

```
SEND Command 67 (with aChan, aValue = lEndPoint)
CALL VerifyResponseAndByteCount(0, 8)
CALL CompareAnalogChannelValue(aChan, lEndPoint, failurepoint)  (6927-6928)
```

Command 67 is sent with a bad unit code.

```
SEND Command 67 (with aChan, aValue = lEndPoint, units = 255)
CALL VerifyResponseAndByteCount("Invalid Units Code", 2)
CALL CompareAnalogChannelValue(aChan, lEndPoint, failurepoint)  (6929-6930)
```

Check Command 67 with an extra data byte.

```
SEND Command 67 (with aChan, aValue = lEndPoint) with an extra byte
CALL VerifyResponseAndByteCount(0, 8)
CALL CompareAnalogChannelValue(aChan, lEndPoint, failurepoint)  (6931-6932)
```

Check "Passed Parameter Too Large"

```
SEND Command 67 (with aChan, aValue = (uEndPoint+(0.1*delta)) )
CALL VerifyResponseAndByteCount("Passed Parameter Too Large", 2)
CALL CompareAnalogChannelValue(aChan, lEndPoint, failurepoint)  (6933-6934)
```

Check "Passed Parameter Too Small"

```
SEND Command 67(with aChan, aValue=(lEndPoint-delta) )
CALL VerifyResponseAndByteCount("Passed Parameter Too Small", 2)
CALL CompareAnalogChannelValue(aChan, lEndPoint, failurepoint)  (6935-6936)
```

Set Analog Channel to UEP

```
IF ("Analog Input Channel")
      Prompt user: "Set Analog Input <aChan> to <uEndPoint>"
ELSE
      SEND Command 66 (with aChan, aValue = uEndPoint)
      CALL VerifyResponseAndByteCount(0, 8)
END IF
CALL CompareAnalogChannelValue(aChan, uEndPoint, failurepoint)  (6937-6938)
SEND Command 67 (with aChan, aValue = lEndPoint)
CALL VerifyResponseAndByteCount("Not In Proper Analog Channel Mode", 2)
CALL CompareAnalogChannelValue(aChan, uEndPoint, failurepoint)  (6939-6940)
```

Remove DUT from fixed Analog Channel mode (if necessary).

```
IF (!"Analog Input Channel")
      SEND Command 66 (with aChan, aValue = "0x7F,0xA0,0x00,0x00")
      CALL VerifyResponseAndByteCount(0, 8)
END IF
```

Verify special requirements for aChan 0 i.e., PV.

```
IF (aChan == 0) THEN
```

Command 6 is sent changing the poll address to 1 (multi-drop mode).

```
      DO
            SEND Command 1
            SEND Command 6 with polling address 0 and
               loop current signaling disabled
            CALL TestValidFrame()
            IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                           (6941)
            END IF
      WHILE (RESPONSE_CODE == "Busy")
      IF ( UNIV_REVISION >= 6 )
            CALL VerifyResponseAndByteCount(0, 4)
            IF (DEVICE_STATUS != "Loop Current Fixed")
                  THEN Test Result is FAIL                           (6942)
            END IF
      ELSE
            THEN CALL VerifyResponseAndByteCount(0, 3)
      END IF
```

A Command 67 message is sent with a current value of LEP.

```
      SEND Command 67 (with aChan, aValue=  lEndPoint)
      CALL VerifyResponseAndByteCount("In Multidrop Mode", 2)
```

Command 6 changing the poll address to 0.

```
      SEND Command 6 with polling address 0 and loop current signaling
            enabled
      IF ( UNIV_REVISION >= 6 )
            CALL VerifyResponseAndByteCount(0, 4)
            IF (DEVICE_STATUS == "Loop Current Fixed")
                  THEN Test Result is FAIL                           (6943)
            END IF
      ELSE
            THEN CALL VerifyResponseAndByteCount(0, 3)
      END IF

END IF

END PROCEDURE
```

7.36 CAL068 Trim Analog Channel Gain

Verifies that the DUT responds properly to Command 68. Checks Addresses, Command Number, Response Code and Byte Count for Command 68, Trim Analog Channel Gain.

Since no prior knowledge of the DUT is assumed, Command 60 is used to identify the Analog Channels supported by the DUT. Then, for each Analog Channel supported, Command 70 is used to retrieve the Upper and Lower End-Points. Operation of Command 67 is confirmed for the Analog Channel. Once verification of the Command 67 implementation is confirmed for an Analog Channel, the next Analog Channel is located until all channels have been processed.

A variety of Response Codes are available for this command. A sequence of Analog Channel settings and commands are sent to provoke all of the possible conditions. Table 19 summarizes the conditions tested and the legal DUT responses.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

Table 19 Command 68 Test Cases

Case	Measured Value	Command 66	Response Code
1	UEP	UEP	"Success"
2	UEP	UEP+ Extra Data Byte	"Success"
3	UEP	200% UEP	"Passed Parameter Too Large"
4	UEP	10% Less Than LEP	"Passed Parameter Too Small"
5	LEP	UEP	"Not In Proper Analog Channel Mode"
6		Unit Code = 255	"Invalid Units Code"

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.10, 7.36

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(68)
CALL VerifyNotWriteProtected()
```

If this command is supported, then Command 70 must be supported (otherwise the minimum and maximum values for the Analog Channel cannot be read!)

```
SEND Command 70 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                    (6945)
END IF
```

If this command is supported, then Command 63 must be supported (otherwise whether the Analog Channel is an input to the DUT cannot be read!)

```
SEND Command 63 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                    (6946)
END IF
```

If this command is supported, then Command 60 must be supported (otherwise the Analog Channel value cannot be read!)

```
      SEND Command 60 with no data bytes
      IF (RESPONSE_CODE == "Command not Implemented")
            THEN Test result is FAIL                                     (6947)
      END IF
```

There must be at least one valid Analog Channel (otherwise why would the command be implemented?)

```
      aChan = -1
      IF (FindNextAnalogChannel(aChan) == "No More Analog Channels")
            THEN Test result is FAIL                                     (6948)
      END IF
```

Test each Analog Channel.

```
      DO
            SEND Command 68 (with aChan only)
            CALL VerifyResponseAndByteCount("Too Few Data Bytes", 2)

            SEND Command 60 (with aChan ) to read most recent aValue
            CALL VerifyResponseAndByteCount(0, 12)
```

Make sure this Analog Channel is valid for Command 67

```
            SEND Command 68 (with aChan and aValue) and an extra data byte
            CALL TestValidFrame
            IF ( (RESPONSE_CODE == "Invalid Analog Channel Code Number")
                  IF (BYTE_COUNT != 2) )
                        THEN Test result is FAIL                         (6949)
                  END IF
            ELSE
                  CALL VerifyAnalogChannelGain (aChan)
            END IF

      WHILE (FindNextAnalogChannel (aChan) == "Analog Channel Found")

      END TEST
```

VerifyAnalogChannelGain(aChan)

Steps through all the test conditions for a single Analog Channel

```
PROCEDURE VerifyAnalogChannelGain(aChan)
SEND Command 70 (with aChan) to read uEndPoint, lEndPoint, units
CALL TestValidFrame
CALL VerifyResponseAndByteCount(0, 12)

SET delta = uEndPoint - lEndPoint
```

Set the Analog Channel to the Upper End Point

```
IF (UNIV_REVISION >= 6) THEN
      SEND Command 63 to read PV Analog Channel Flags
ELSE
      Prompt user to set Analog Channel Flags
END IF
CALL TestValidFrame
CALL VerifyResponseAndByteCount(0, 19)
IF ("Analog Input Channel")
      Prompt user: "Set Analog Input <aChan> to <uEndPoint>"
ELSE
      SEND Command 66 (with aChan, aValue = uEndPoint)
      CALL VerifyResponseAndByteCount(0, 8)
END IF
CALL CompareAnalogChannelValue(aChan, uEndPoint, failurepoint)  (6950-6951)
```

A valid Command 68 message is sent with a current value of <uEndPoint> .

```
SEND Command 68 (with aChan, aValue = uEndPoint)
CALL VerifyResponseAndByteCount(0, 8)
CALL CompareAnalogChannelValue(aChan, uEndPoint, failurepoint)  (6952-6953)
```

Command 68 is sent with a bad unit code.

```
SEND Command 68 (with aChan, aValue = uEndPoint, units = 255)
CALL VerifyResponseAndByteCount("Invalid Units Code", 2)
CALL CompareAnalogChannelValue(aChan, uEndPoint, failurepoint)  (6954-6955)
```

Check Command 68 with an extra data byte.

```
SEND Command 68 (with aChan, aValue = uEndPoint) with an extra byte
CALL VerifyResponseAndByteCount(0, 8)
CALL CompareAnalogChannelValue(aChan, uEndPoint, failurepoint)  (6956-6957)
```

Check "Passed Parameter Too Large"

```
SEND Command 68 (with aChan, aValue = (uEndPoint + delta) )
CALL VerifyResponseAndByteCount("Passed Parameter Too Large", 2)
CALL CompareAnalogChannelValue(aChan, uEndPoint, failurepoint)  (6958-6959)
```

Check "Passed Parameter Too Small"

```
SEND Command 68(with aChan, aValue=(lEndPoint-(0.1*delta)) )
CALL VerifyResponseAndByteCount("Passed Parameter Too Small", 2)
CALL CompareAnalogChannelValue(aChan, lEndPoint, failurepoint)  (6960-6961)
```

Set Analog Channel to UEP

```
IF ("Analog Input Channel")
      Prompt user: "Set Analog Input <aChan> to <lEndPoint>"
ELSE
      SEND Command 66 (with aChan, aValue = lEndPoint)
      CALL VerifyResponseAndByteCount(0, 8)
END IF
CALL CompareAnalogChannelValue(aChan, lEndPoint, failurepoint)  (6962-6963)
SEND Command 68 (with aChan, aValue = uEndPoint)
CALL VerifyResponseAndByteCount("Not In Proper Analog Channel Mode", 2)
CALL CompareAnalogChannelValue(aChan, lEndPoint, failurepoint)  (6964-6965)
```

Remove DUT from fixed Analog Channel mode (if necessary).

```
      IF (!"Analog Input Channel")
            SEND Command 66 (with aChan, aValue = "0x7F,0xA0,0x00,0x00")
            CALL VerifyResponseAndByteCount(0, 8)
      END IF
```

Verify special requirements for aChan 0 i.e., PV.

```
      IF (aChan == 0) THEN
```

Command 6 is sent changing the poll address to 1 (multi-drop mode).

```
            DO
                  SEND Command 1
                  SEND Command 6 with polling address 0 and
                     loop current signaling disabled
                  CALL TestValidFrame()
                  IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
                        THEN Test result is FAIL                           (6966)
                  END IF
            WHILE (RESPONSE_CODE == "Busy")
            IF ( UNIV_REVISION >= 6 )
                  CALL VerifyResponseAndByteCount(0, 4)
                  IF (DEVICE_STATUS != "Loop Current Fixed")
                        THEN Test Result is FAIL                           (6967)
                  END IF
            ELSE
                  THEN CALL VerifyResponseAndByteCount(0, 3)
            END IF
```

A Command 68 message is sent with a current value of UEP.

```
            SEND Command 68 (with aChan, aValue = uEndPoint)
            CALL VerifyResponseAndByteCount("In Multidrop Mode", 2)
```

Command 6 changing the poll address to 0.

```
            SEND Command 6 with polling address 0 and loop current signaling
                  enabled
            IF ( UNIV_REVISION >= 6 )
                  CALL VerifyResponseAndByteCount(0, 4)
                  IF (DEVICE_STATUS == "Loop Current Fixed")
                        THEN Test Result is FAIL                           (6968)
                  END IF
            ELSE
                  THEN CALL VerifyResponseAndByteCount(0, 3)
            END IF
      END IF
      END PROCEDURE
```

7.37 CAL069 Write Analog Channel Transfer Function

Verifies that the DUT responds properly to Command 69. Checks Addresses, Command Number, Response Code and Byte Count for Command 69, Write Analog Channel Transfer Function.

A Command 69 is sent with an analog channel code of zero.

A Command 63 is sent starting with an analog channel code of one and incrementing the code through 4. When a Response Code of zero is returned, the following sequence of Command 69 followed Command 63 is used:

A Command 63 is sent to determine the original analog channel transfer function of the device.

A sequence of Command 69 is sent using all possible byte values from 0 to 254. Each Command 69 is followed by a Command 63 to read the analog channel transfer function. If a "Busy" Response Code (32) is detected, Command 63 will be repeated until the Response Code is 0. A valid message is sent containing the original transfer function.

A valid message is sent with an extra data byte (original transfer function).

An invalid message is sent which is one data byte short.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.10, 7.37

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(69)
CALL VerifyNotWriteProtected()
```

If this command is supported, then Command 63 must be supported (otherwise the transfer function can be written but not read!)

```
SEND Command 63 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                   (7440)
END IF
```

There must be at least one valid Analog Channel (otherwise why would the command be implemented?)

```
aChan = -1
IF (FindNextAnalogChannel(aChan) == "No More Analog Channels")
      THEN Test result is FAIL                                   (7441)
END IF
```

Test each Analog Channel.

```
DO
      SEND Command 69 (with aChan only)
      CALL VerifyResponseAndByteCount("Too Few Data Bytes", 2)

      SEND Command 63 (with aChan ) to read tf0
      CALL VerifyResponseAndByteCount(0, 19)
```

Make sure this Analog Channel is valid for Command 69.

```
      SEND Command 69 (with aChan and tf0)
      CALL TestValidFrame
      IF ( (RESPONSE_CODE == "Invalid Selection")
            IF (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                           (7442)
            END IF
      ELSE
            CALL VerifyResponseAndByteCount(0, 4)
            CALL VerifyAnalogChannelFunction(dVar)
      END IF
WHILE (FindNextAnalogChannel (aChan) == "Analog Channel Found")
```

VerifyAnalogChannelFunction(aChan)

Read original Transfer Function code.

```
PROCEDURE VerifyAnalogChannelFunction(aChan)
SEND Command 63 (with aChan) to read tf0
CALL VerifyResponseAndByteCount(0, 19)
CALL TestValidFrame
```

Test all possibilities for function code.

```
SET tfLast = tf0
FOR ( n = [0 – 254] )
      SEND Command 69 (with aChan,  code = n)
      IF (RESPONSE_CODE == "SUCCESS") THEN
            CALL VerifyResponseAndByteCount(0, 4)
            tfLast = n

            SWITCH on (n)
            CASE 250-254
                  Test result is FAIL                                (7443)

            CASE 231-233
                  PRINT "Warning: Transfer Function Code <n>
                        is not interoperable and should not be supported in
                        any Field Device"
            CASE DEFAULT

            END SWITCH
            ELSE IF (RESPONSE_CODE == "Invalid Selection") THEN
                  IF (BYTE_COUNT != 2)
                        THEN Test result is FAIL                           (7444)
                  END IF
            ELSE
                  Test result is FAIL                                      (7445)
            END IF
            CALL CompareAnalogChannelFunction(aChan,tfLast,           (7446-7447)
                failurepoint)
      FOR END
```

A valid message is sent with an extra data byte (original transfer function code).

```
      SEND Command 69 (with aChan, code = tf0) and one extra data byte
      CALL TestValidFrame
      CALL VerifyResponseAndByteCount(0, 4)
      CALL CompareAnalogChannelFunction(aChan, tf0,                   (7448-7449)
                failurepoint)

      PROCEDURE END
```

CompareAnalogChannelFunction(aChan, f, failurepoint)

This procedure is unique to CAL069.

```
      PROCEDURE CompareAnalogChannelFunction(aChan, f, FAILUREPOINT)
      DO
            SEND Command 60 (with aChan)
            CALL VerifyResponseAndByteCount(0, 12)
            SEND Command 63(with aChan) to read transfer function
            CALL TestValidFrame()
            IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                         (FAILUREPOINT)
            END IF
      WHILE (RESPONSE_CODE == "Busy")
      CALL TestValidFrame
      CALL VerifyResponseAndByteCount(0, 19)
      IF transfer function != f
            THEN Test result is FAIL                             (FAILUREPOINT+1)
      END IF
      PROCEDURE END
```

Messages

In addition, the number of analog channel codes used in the command and the valid analog channel transfer function codes will be reported.

7.38 CAL070 Read Analog Channel Endpoint Values

Verifies that the DUT responds properly to Command 70. Checks Addresses, Command Number, Response Code and Byte Count for Command 70, Read Analog Channel Endpoint Values.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.10, 7.38

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(70)
```

Test each analog channel.

```
FOR ( aChan = [0-24] )
      SEND Command 70 (with aChan)
      CALL TestValidFrame
      IF ( (RESPONSE_CODE == "SUCCESS") THEN
            IF (BYTE_COUNT != 12)
                  THEN Test result is FAIL                           (6975)
            ELSE
                  SEND Command 70 (with aChan) and an extra data byte
                  CALL TestValidFrame
                  CALL VerifyResponseAndByteCount(0, 12)
            END IF
      ELSE IF (RESPONSE_CODE == "Invalid Selection") THEN
            IF (BYTE_COUNT != 2)
                  THEN Test result is FAIL                           (6976)
            END IF
      ELSE
            THEN Test result is FAIL                                 (6977)
      END IF
FOR END

END TEST
```

Messages

In addition, the number of analog channel codes used in the command will be reported.

7.39 CAL071 Lock Device

Verifies that the DUT responds properly to Command 71. Checks Addresses, Command Number, Response Code and Byte Count for Command 71, Lock Device. See Table 20 for the test cases performed.

Notes:

1. "Too Few Data Bytes Received" Response Code is verified in CAL000.
2. Command 76 is verified in CAL000.

Table 20 Basic Command 71 Test Cases

Action	Pass Criteria
Invalid Lock Code	Response Code = "Invalid Lock Code"
Command 71 with an extra byte	Normal operation of the DUT occurs
Make Temporary Lock	Device Reset unlocks DUT
Make Permanent Lock	Device Reset does not Unlock DUT
Lock All	Device Reset does not Unlock DUT
Lock All	No master can change configuration

Table 21 Command 71 Test Cases Performed for Each Master

Action	Pass Criteria
Lock DUT	Successful Temporary or Permanent lock by correct master.
Lock attempt by other master	Attempt to lock fails
Unlock attempt by other master	Attempt to unlock fails
This master can write	Successful write
Other master cannot write	Write attempt fails
Other master can read	Read is successful
Other master cannot reset	Reset command fails

Test Case A applies to all HART products. Test Case B applies to wireless devices and gateways. Test Case B verifies that the wireless DUT rejects writeable commands when locked by Wireless Gateway. This test requires device access via a Wireless Gateway.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.4, 7.39

Test Case A

```
      CALL IdentifyDevice
      CALL CheckCommandImplemented(71)
      CALL VerifyNotWriteProtected()
```

If this command is supported, then Command 76 must be supported (otherwise the Lock Status cannot be read.)

```
      SEND Command 76
      IF (RESPONSE_CODE == "Command not Implemented")
            THEN Test result is FAIL                                     (7002)
      END IF
```

We have to start out unlocked

```
      IF (IsLocked() == "Device Locked")
            THEN Test Result is FAIL                                     (7003)
      END IF
```

Issue command 71 with an invalid lock code.

```
      SEND Command 71 (with lockCode = 255)
      CALL VerifyResponseAndByteCount("Invalid Lock Code", 2)
      CALL TestValidFrame
```

Issue command 71 with lock code 1 plus one data byte.

```
      SEND Command 71 (with lockCode = "Unlock") and one extra byte
      CALL VerifyResponseAndByteCount(0, 3)
      CALL TestValidFrame
```

Do the basic lock testing.

```
      TestTheLock (Primary, failurepoint)                           (6980-6987)
      TestTheLock (Secondary, failurepoint)                         (6990-6997)
      TestTheLock (All, failurepoint)                               (7420-7427)
```

Check each lock code and whether a device reset clears the lock. A temporary lock should clear on a device reset.

```
      IF (Lock(Primary, "Temporary") != "FAIL") THEN
            CALL DoDeviceReset(failurepoint)                        (6988-6989)
            IF (IsLocked() == "Device Locked")
                  THEN Test Result is FAIL                               (7004)
            END IF
      END IF
```

Issue command 71 to permanently lock the device. Issue command 42 to reset the device. verify that the device is still locked.

```
      IF (Lock(Primary, "Permanent") != "FAIL") THEN
            CALL DoDeviceReset(failurepoint)                        (6998-6999)
            IF (IsLocked() != "Device Locked")
                  THEN Test Result is FAIL                               (7005)
            END IF
            IF (Unlock(PRIMARY) == FAIL)
                  THEN Test Result is FAIL                               (7006)
            END IF
      END IF

      END TEST CASE
```

Test Case B Writelock on TDMA Products

Wireless Product locked by Wireless Gateway. Verifies that the wireless DUT rejects writeable commands when locked by Wireless Gateway. This test requires device access via a Wireless Gateway.

```
CALL IdentifyDevice

IF PROFILE < 129 THEN
      ABORT TEST
END IF

CALL CheckCommandImplemented(71)
CALL VerifyNotWriteProtected()
```

If the lock command is supported, then Command 76 must be supported (otherwise the Lock Status cannot be read.)

```
SEND Command 76
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                  (7410)
END IF

CALL VerifyResponseAndByteCount(0, 3)
```

We have to start out unlocked

```
IF (IsLocked() == "Device Locked")
      THEN Test Result is FAIL                                  (7411)
END IF
```

Do the basic lock testing.

```
PRINT ("Please issue lock command from wireless Gateway.")
```

Verify Lock.

```
lockStatus = IsLocked()
IF ( (RESPONSE_CODE != 0) OR ((lockStatus != "Device Locked") AND
      lockStatus != "Locked by Gateway"))THEN
      TEST Result is FAIL                                       (7412)
END IF
```

Test the primary and secondary Token-Passing master.

```
FOR master = 0 to 1
```

Test that the wired master cannot write

```
      SEND Command 17 (master, message=24 bytes of 0x00)
      CALL TestValidFrame
      CALL VerifyResponseAndByteCount("Access Restricted", 2)
```

and that anyone can read.

```
      DO
            SEND Command 1
            CALL VerifyResponseAndByteCount(0, 7)

            SEND Command 12 (Master)
            CALL TestValidFrame()
            IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                      (7413)
            END IF
      WHILE (RESPONSE_CODE == "Busy")
      CALL VerifyResponseAndByteCount(0, 26)
```

Only the right master can do a reset, too.

```
                  SEND Command 42 (master)
                  CALL TestValidFrame
                  IF (RESPONSE_CODE == "Command not Implemented")
                  THEN Test result is FAIL                                 (7414)
                  ELSE
                        CALL VerifyResponseAndByteCount("Access Restricted", 2)
                  END IF
```

The wired masters cannot do a reset if the lock code is Gateway.

```
                  IF (master != ALL) THEN
                        SEND Command 42 (master)
                        CALL TestValidFrame
                        IF (RESPONSE_CODE == "Command not Implemented")
                          THEN Test result is FAIL                         (7415)
                        ELSE
                          CALL VerifyResponseAndByteCount("Access Restricted", 2)
                        END IF
                  END IF
      END FOR
```

Check whether a device reset clears the lock. A Gateway lock should not clear on a device reset.

```
            CALL DoDeviceReset(failurepoint)                          (7416-7417)
            IF (IsLocked() != "Device Locked")
                  THEN Test Result is FAIL                                 (7418)
            END IF
```

The device will need to be unlocked before returning to testing.

```
      PRINT ("Please issue unlock command from wireless Gateway.")

      END TEST CASE
```

TestTheLock(master, failurepoint)

Performs a basic lock testing for a given master. First lock the Field Device.

```
      IF (Lock(master, "Temporary") == "FAIL") THEN
            IF (Lock(master, "Permanent") == "FAIL")
                  THEN Test Result is FAIL                         (FAILUREPOINT)
            END IF
      END IF
```

Now try to unlock the DUT.

```
      IF (Lock(!master, "Temporary") != FAIL)
            THEN Test Result is FAIL                             (FAILUREPOINT+1)
      END IF

      IF (Unlock(!master) != FAIL)
            THEN Test Result is FAIL                             (FAILUREPOINT+2)
      END IF
```

Verify that only the right master can write for lock codes other than "Lock All" (code = 3) and anyone can read.

```
      DO
            SEND Command 1
            SEND Command 17 (master, message=24 bytes of 0x00)
            CALL TestValidFrame()
            IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                       (FAILUREPOINT+3)
      END IF
WHILE (RESPONSE_CODE == "Busy")

IF (master != ALL) THEN
      CALL VerifyResponseAndByteCount(0, 26)
ELSE
      CALL VerifyResponseAndByteCount("Access Restricted", 2)
END IF
```

Verify that the other master cannot write

```
SEND Command 17 (!master, message=24 bytes of 0xFF)
CALL TestValidFrame
CALL VerifyResponseAndByteCount("Access Restricted", 2)
```

and anyone can read.

```
DO
      SEND Command 1
      SEND Command 12 (!master)
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                       (FAILUREPOINT+4)
      END IF
WHILE (RESPONSE_CODE == "Busy")
CALL VerifyResponseAndByteCount(0, 26)
```

and only the right master can do a reset.

```
SEND Command 42 (!master)
CALL TestValidFrame
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                             (FAILUREPOINT+5)
ELSE
      CALL VerifyResponseAndByteCount("Access Restricted", 2)
END IF
```

The initiating master cannot do a reset if the lock code is All.

```
IF (master != ALL) THEN
      SEND Command 42 (master)
      CALL TestValidFrame
      IF (RESPONSE_CODE == "Command not Implemented")
            THEN Test result is FAIL                       (FAILUREPOINT+7)
      ELSE
            CALL VerifyResponseAndByteCount("Access Restricted", 2)
      END IF
END IF

Unlock the device and we are done.
IF (Unlock(MASTER) == FAIL)
      THEN Test Result is FAIL                             (FAILUREPOINT+6)
END IF
PROCEDURE END
```

DoDeviceReset(failurepoint)

Performs a Device Reset to confirm lock integrity

```
PROCEDURE DoDeviceReset(FAILUREPOINT)
SEND Command 42
CALL TestValidFrame
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test Result is FAIL                                (FAILUREPOINT)
ELSE
      CALL VerifyResponseAndByteCount(0, 2)
END IF
```

First command after and Device Reset must Cold Start must be set

```
DO
      SEND Command 0
WHILE (COMMUNICATIONS_ERROR == "No Response")
CALL TestValidFrame
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 19)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 14)
END IF
IF (DEVICE_STATUS != "Cold Start")
      THEN Test Result is FAIL                              (FAILUREPOINT+1)
END IF
PROCEDURE END
```

Lock(master, lockCode)

Issue command 71 to lock the device. Return "SUCCESS" if Command 71 succeeds.

```
PROCEDURE Lock(master, lockCode)
CALL IssueCommand71 (master, lockCode)
lockStatus = IsLocked()
IF ( (RESPONSE_CODE != 0) OR (lockStatus != "Device Locked") )THEN
      RETURN FAIL
END IF
IF ( (lockCode == "Permanent") AND (lockStatus != "Lock is Permanent") )
      RETURN FAIL
END IF
IF ( (lockCode == "Temporary") AND (lockStatus == "Lock is Permanent") )
      RETURN FAIL
END IF
IF ( (master == PRIMARY) AND (lockStatus != "Locked by Primary Master") )
      RETURN FAIL
END IF
IF ( (master == SECONDARY) AND (lockStatus == "Locked by Primary Master") )
      RETURN FAIL
END IF
RETURN SUCCESS
PROCEDURE END
```

Unlock(whichMaster)

This procedure is unique to CAL071. Issue command 71 to unlock the device, then issue Command 12. Read Message repeatedly until the device is not "Busy". Return success if Command 71 succeeds, or failure otherwise.

```
PROCEDURE Unlock(whichMaster)
CALL IssueCommand71 (whichMaster, lockCode = "Unlock")
IF (RESPONSE_CODE == "SUCCESS") THEN
      IF (IsLocked() != "Device Locked") THEN
            RETURN SUCCESS
      ELSE
            RETURN FAIL
      END IF
ELSE
      RETURN FAIL
END IF
PROCEDURE END
```

IsLocked()

Issue command 76 to determine if the DUT is locked.

```
PROCEDURE IsLocked()

DO
      SEND Command 1
      SEND Command 76
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                                (7000)
      END IF
WHILE (RESPONSE_CODE == "Busy")
CALL VerifyResponseAndByteCount(0, 3)

RETURN (lockStatus)
PROCEDURE END
```

IssueCommand71(master, lockCode)

Issue command 71 taking care of "Busy" and Delayed Responses.

```
PROCEDURE IssueCommand71(master, lockCode)
DO
      SEND Command 1
      SEND Command 71 (with master, lockCode)
      CALL TestValidFrame()
      IF ( ( (RESPONSE_CODE == "Busy")
         OR (RESPONSE_CODE == "Delayed Response Initiated")
         OR (RESPONSE_CODE == "Delayed Response Running") )
         AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                                (7001)
      END IF
WHILE ( (RESPONSE_CODE == "Busy")
   OR (RESPONSE_CODE == "Delayed Response Initiated")
   OR (RESPONSE_CODE == "Delayed Response Running") )

RETURN RESPONSE_CODE
PROCEDURE END
```

7.40 CAL072 Squawk

Verifies that the DUT responds properly to Command 72. Checks Addresses, Command Number, Response Code and Byte Count for Command 72, Squawk.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.1, 7.40

Test Procedure

```
CALL IdentifyDevice
CALL CheckForRecommendedCommand(72)                                    (5002)
```

Let's Squawk!!!

```
FOR (50 Iterations)
      SEND Command 72
      IF (RESPONSE_CODE != [0, "Unable to Squawk", "Access Restricted" or
            "Busy"])
            Test Result is FAIL                                        (7008)
      END IF
      IF (BYTE_COUNT != 2)
            Test Result is FAIL                                        (7009)
      END IF
END FOR

END TEST
```

7.41 CAL073 Find Device

Verifies that the DUT responds properly to Command 73. Checks Addresses, Command Number, Response Code and Byte Count for Command 73, Find Device. The following conditions are evaluated:

- Issue command 73 to Find Device. No message should be returned. Repeat using the broadcast address. No message should be returned.
- Have the user arm the device. Issue command 73 to Find Device. A message with Response Code 0 should be returned. Repeat using the broadcast address. A message with Response Code 0 should be returned.
- Have the user disarm the device. Issue command 73 to Find Device. No message should be returned. Repeat using the broadcast address. No message should be returned.

References:

Specification	**Rev.**	**Sections**
Common Practice Command Specification	7.0	6.1, 7.41

Test Procedure

```
CALL IdentifyDevice
SEND Command 73
IF (COMMUNICATIONS_ERROR != "No Response")  THEN
      IF (RESPONSE_CODE = "Command not Implemented")
            PRINT "Warning, Implementation of Command 73 is strongly
              recommended.  This command is implemented by most
              Field Devices and widely used in Host Applications."
            Abort Test                                              (5002)
      ELSE
            THEN Test Result is FAIL                                (7010)
      END IF
END IF
```

Issue command 73 to Find Device. No message should be returned. Repeat using the broadcast address. No message should be returned.

```
SEND Command 73 with broadcast address
IF (COMMUNICATIONS_ERROR != "No Response")
      THEN Test Result is FAIL                                      (7011)
END IF
SEND Command 73 with DUT address
IF (COMMUNICATIONS_ERROR != "No Response")
      THEN Test Result is FAIL                                      (7012)
END IF
```

Have the user arm the device. Issue command 73 to Find Device. A message with Response Code 0 should be returned. Repeat using the broadcast address. A message with Response Code 0 should be returned. Repeat using the broadcast address.

```
PROMPT: "Arm the device."
SEND Command 73 with broadcast address
CALL TestValidFrame
CALL VerifyResponseAndByteCount(0, 19)

PROMPT: "Arm the device."
SEND Command 73 with DUT address
CALL VerifyResponseAndByteCount(0, 19)
CALL TestValidFrame
```

Have the user disarm the device. Issue command 73 to Find Device. No message should be returned. Repeat using the broadcast address. No message should be returned.

```
PROMPT: "Disarm the device."
SEND Command 73 with broadcast address
IF (COMMUNICATIONS_ERROR != "No Response")
      THEN Test Result is FAIL                                       (7013)
END IF
SEND Command 73 with DUT address
IF (COMMUNICATIONS_ERROR != "No Response")
      THEN Test Result is FAIL                                       (7014)
END IF

END TEST
```

7.42 CAL074 Verify I/O System Commands

Verifies that the DUT responds properly to Command 74, 75, 77, 84, 85, 86, 87, 88, 94, and 95. Checks Addresses, Command Number, Response Code and Byte Count for Command 74, Read I/O System Capabilities, and Command, 75 Poll Sub-Device. Command 75 and 77 will not be supported by the wired side of wireless Adapters, therefore these commands will not be tested for those devices. The following conditions are evaluated:

- Issue Command 0. Examine the "Protocol Bridge Device" bit of the Flags byte
- Issue command 74. Test that the maximum number of I/O cards, the maximum number of channels, and the maximum number of sub-devices per channel are all at least 1.
- Send Command 74 and 75 with one byte too many
- Issue Command 75 for each card, channel, and sub-device
- Issue Command 84 to Read the summary of Sub-Devices
- Send Command 77 to request Command from specific sub-device
- Send Command 85 to Read I/O Channel Statistics
- Send Command 86 to Read Sub-Device Statistics
- Send Command 87 Write I/O System Master Mode
- Send Command 88 Write I/O System Retry Count
- Send Command 94 Read I/O System Client-Side Communication Statistics

TestCase A verifies the basic I/O system and device functionality. TestCase B specifically tests features supported by HART 7 and later I/O systems. TestCase C verifies the changing of master address in HART 7 and later I/O systems. TestCase D is focused on the I/O System statistics in HART 7 and later products.

Note: "Too Few Data Bytes Received" Response Code for Commands 74 and 75 is verified in CAL000.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	7.42, 7.43, 7.45, 7.52, 7.53, 7.55, 7.56, 7.62, 7.63

Test Case A: Basic I/O and Subdevice Tests.

```
CALL IdentifyDevice
CALL VerifyAssociatedCommand(74, 75)

IF (UNIV_REVISION > 6) THEN
   CALL VerifyAssociatedCommand(74,75,77,84,85,86,87,88,94,95)
END IF
```

Issue command 0. Examine the "Protocol Bridge Device" bit of the Flags byte

```
SEND Command 0
IF (FLAGS != "Protocol Bridge Device")
   THEN ABORT "DEVICE IS NOT A Bridge device"                        (7015)
END IF
```

Issue command 74. Test that the maximum number of I/O cards, the maximum number of channels, and the maximum number of sub-devices per channel are all, at least, 1. Also send Command 74 with one byte too many.

```
SEND Command 74 to read numCards, numChannels, numSubDevices
CALL TestValidFrame
IF (UNIV_REVISION > 6)
   CALL VerifyResponseAndByteCount(0, 10)
ELSE
   CALL VerifyResponseAndByteCount(0, 5)
END IF
IF (numCards < 1)
   THEN Test Result is FAIL                                          (7016)
END IF

IF (numChannels < 1)
   THEN Test Result is FAIL                                          (7017)
END IF

IF (numSubDevices < 1)
   THEN Test Result is FAIL                                          (7018)
END IF

SEND Command 74 with one extra data byte, 0x00
CALL TestValidFrame
CALL VerifyResponseAndByteCount(0, 5)
```

Do basic testing of Command 75

```
CALL IssueCmd75(card=0, channel=0, device=0)
IF (RESPONSE_CODE == "SUCCESS")
   IF (UNIV_REVISION >= 6)    THEN SET Cmd75BC = 19
                              ELSE SET Cmd75BC = 14
   END IF
   IF (BYTE_COUNT != Cmd75BC)
         THEN Test result is FAIL                                    (7019)
   END IF
ELSE IF (RESPONSE_CODE == "No Sub-Device Found")
   IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                       (7020)
   END IF
ELSE
   Test result is FAIL                                               (7021)
END IF

CALL IssueCmd75(card=255, channel=255, device=255)
IF (RESPONSE_CODE == ["Invalid Selection", "Invalid I/O card number",
  OR "Invalid Channel number"] )
   IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                       (7022)
   END IF
ELSE
   Test result is FAIL                                               (7023)
END IF
```

Test Command 75 with an extra byte

```
DO
```

```
   SEND Command 1
   SEND Command 75 (with 0, 0, 0) and an extra byte
   CALL TestValidFrame()
   IF ( ( (RESPONSE_CODE == "Busy")
      OR (RESPONSE_CODE == "Delayed Response Initiated")
      OR (RESPONSE_CODE == "Delayed Response Running") )
      AND (BYTE_COUNT != 2) )
      THEN Test result is FAIL                                       (7024)
   END IF
WHILE ( (RESPONSE_CODE == "Busy")
   OR (RESPONSE_CODE == "Delayed Response Initiated")
   OR (RESPONSE_CODE == "Delayed Response Running") )

IF (RESPONSE_CODE == "SUCCESS")
   IF (BYTE_COUNT != Cmd75BC)
      THEN Test result is FAIL                                       (7025)
   END IF
ELSE IF (RESPONSE_CODE == "No Sub-Device Found")
   IF (BYTE_COUNT != 2)
      THEN Test result is FAIL                                       (7026)
   END IF
ELSE
   Test result is FAIL                                               (7027)
END IF
```

Based on numCards, numChannels, numSubDevices, check all legal sub-device possibilities.

```
FOR (card = [0-(numCards-1)] )
   FOR (channel = [0-(numChannels-1)] )
      FOR (device = [0-(numSubDevices-1)] )
         CALL IssueCmd75(card, channel, device)
         IF (RESPONSE_CODE == "SUCCESS")
            IF (BYTE_COUNT != Cmd75BC)
               THEN Test result is FAIL                              (7028)
            END IF
         ELSE IF (RESPONSE_CODE == "No Sub-Device Found")
            IF (BYTE_COUNT != 2)
               THEN Test result is FAIL                              (7029)
            END IF
         ELSE
            Test result is FAIL                                      (7030)
         END IF
      END FOR
   END FOR
END FOR
END TEST CASE
```

Test Case B: HART 7 I/O and Subdevice Testing

```
CALL IdentifyDevice

IF (UNIV_REVISION > 6) THEN
   CALL VerifyAssociatedCommand(74,75,77,84,85,86,87,88,94,95)
ELSE
   THEN Test result is ABORT                                      (7080)
END IF
```

Use Command 74 to determine capabilities of I/O system.

```
SEND Command 74
CALL VerifyResponseAndByteCount(0, 10)
maxNumSubDevSupported = Cmd74Rsp.numSubDevices *
                        Cmd74Rsp.numChannels *
                        Cmd74Rsp.numCards

IF maxNumSubDevSupported < 2
   THEN WARNING: Support for more than one sub-device is recommended.
END IF

numDevReported = Cmd74Rsp.NumberofDevicesDetected

IF Cmd74Rsp.NumDelayedResponses < 2
   THEN Test result is FAIL                                       (7081)
END IF

IF ((Cmd74Rsp.RetryCount < 2) OR (Cmd74Rsp.RetryCount > 5))
   THEN Test result is FAIL                                       (7082)
END IF

SEND Command 88 with RetryCount = 1
CALL VerifyResponseAndByteCount("Passed Parameter too small", 2)
SEND Command 88 with RetryCount = 6
CALL VerifyResponseAndByteCount("Passed Parameter too large", 2)
SEND Command 88 with RetryCount = 3
CALL VerifyResponseAndByteCount(0, 3)

SEND Command 74
CALL VerifyResponseAndByteCount(0, 10)
IF ((Cmd74Rsp.RetryCount != 3)
   THEN Test result is FAIL                                       (7083)
END IF
```

We are going to build a list of devices. To store device info, several arrays are used.

```
cd[]      = null
ch[]      = null
MfrID[]   = null
Addr[]    = null
UnivRev[] = null
Tag[]     = null

numDevices = 0
numDevices1 = 0
```

Poll for sub-devices to verify I/O system list.

```
FOR (card = [0-(numCards-1)] )
   FOR (channel = [0-(numChannels-1)] )
      FOR (device = [0-(numSubDevices-1)] )
         CALL IssueCmd75(card, channel, device)
         IF (RESPONSE_CODE == "SUCCESS") THEN

            cd[numDevices]      = card
            ch[numDevices]      = channel
            MfrID[numDevices]   = Cmd75Rsp.ManufacturerID
            Addr[numDevices]    = {Cmd75Rsp.ExpandedDeviceType,
                                      Cmd75Rsp.DeviceID }
            UnivRev[numDevices] = Cmd75Rsp.UniversalRevision

            INCREMENT numDevices
         END IF
      END FOR

      SET MaxPoll = 63
      Cmd0Req = {0, 0, null}
```

Let us also check how many we find using Command 77

```
      DO
         IssueCmd77(card, channel, poll, Cmd0Req, Cmd0Rsp)
         IF Cmd0Rsp.Data.ResponseCode = "Success" THEN
            INCREMENT numDevices1
            IF Cmd0Rsp.Data.UnivRev = 5
               THEN SET MaxPoll = 15
            END IF
         ELSE
            IF Cmd77.Rsp.ResponseCode != "DR_DEAD"
               THEN Test result is FAIL                              (7084)
            END IF

         END IF
      WHILE (poll < MaxPoll)

   END FOR
END FOR
```

Verify that the number of sub-devices indexed in command 74 equals the number found through sending command 0 to each address.

```
IF numDevices != numDevReported
      THEN Test result is FAIL                                       (7085)
END IF

IF numDevices != numDevices1
      THEN Test result is FAIL                                       (7086)
END IF
```

Now do basic testing of Command 77. By using the address we are confirming that Command 77 is talking to the same device we found using Command 75 (i.e., the Unique IDs match)

```
SET Cmd20Req = {Cmd = 20, Byte Count = 0, Data = null}
FOR (n = 0 to numDevReported)

   CALL IssueCmd77 (cd[n], ch[n], Addr[n], Cmd20Req, Cmd20Rsp)
   IF RESPONSE_CODE != "SUCCESS"
      THEN Test result is FAIL                                       (7087)
   END IF

   Tag[n] = Cmd20Rsp.Data
END FOR
```

Test Command 84 device list - 0th entry is the I/O system itself.

```
SEND Command 84 with SubDeviceIndex = 0
SEND Command 0
```

```
      SEND Command 20

      IF (Cmd84Rsp.IOCard != 251) OR (Cmd84Rsp.Channel != 251)
         THEN Test result is FAIL                                          (7088)
      END IF

      IF ( (Cmd20Rsp.Data != Cmd84Rsp.Tag)
        OR (Cmd0Rsp.ManufacturerID       != Cmd84Rsp.ManufacturerID)
        OR (Cmd0Rsp.ExpandedDeviceType   != Cmd84Rsp.ExpandedDeviceType)
        OR (Cmd0Rsp.DeviceID             != Cmd84Rsp.DeviceID)
        OR (Cmd0Rsp.UniversalRevision    != Cmd84Rsp.UniversalRevision) )
         THEN Test result is FAIL                                          (7090)
      END IF
```

Walk the device list returned by Command 84 to verify the list we built using Command 75 and 77 above. The order we found the devices in may not be the same as the order in the Command 84 device list.

```
      FOR (n = 1 to numDevReported+1)
         SEND Command 84 with SubDeviceIndex = n

         SET m = 0
         SET DeviceNotFound = TRUE
```

Walk the Command 84 device list. Do we have a match?

```
         DO
            IF Addr[m] != { Cmd84Rsp.ExpandedDeviceType, Cmd84Rsp.DeviceID} THEN
               SET DeviceNotFound = FALSE
            ELSE
               INCREMENT m
            END IF
         WHILE ( (m < (numDevReported+1)) AND DeviceNotFound
```

No match in our list

```
         IF DeviceNotFound
            THEN Test result is FAIL                                       (7091)
         END IF
```

Are the values the same as in our Cmd 75/77 list?

```
         IF (Cmd84Rsp.IOCard != cd[n]) OR (Cmd84Rsp.Channel != ch[n])
            THEN Test result is FAIL                                       (7092)
         END IF

         IF ( (Tag[n]      != Cmd84Rsp.Tag)
           OR (MfrID[n]    != Cmd84Rsp.ManufacturerID)
           OR (UnivRev[n]  != Cmd84Rsp.UniversalRevision) )
              THEN Test result is FAIL                                     (7093)
         END IF

      END FOR
```

Verify that the number of sub-devices indexed in command 84 equals the number found through sending command 0 to each address.

```
      SEND Command 84 with SubDeviceIndex = numDevReported+2
      CALL VerifyResponseAndByteCount("Invalid Selection", 2)

      END TestCase
```

Test Case C: Command 87 and 88 Testing

Verify operation of Command 87 while changing primary and secondary master modes.

```
      CALL IdentifyDevice
```

```
IF (UNIV_REVISION > 6) THEN
   CALL VerifyAssociatedCommand(74,75,77,84,85,86,87,88,94,95)
ELSE
   Test result is ABORT                                               (7095)
END IF
```

Use Command 74 to establish current master mode.

```
SEND Command 74
CALL VerifyResponseAndByteCount(0, 10)

masterMode = Cmd74Rsp.mastermode
```

Set the master mode to the opposite of current value.

```
SEND Command 87 with !masterMode
CALL VerifyResponseAndByteCount(0, 3)

IF Command87.mastermode != !masterMode
   THEN Test Result is FAIL                                           (7096)
END IF
```

Set the master mode to the original value.

```
SEND Command 87 with masterMode
CALL VerifyResponseAndByteCount(0, 3)

IF Command87.mastermode != masterMode
   THEN Test Result is FAIL                                           (7097)
END IF
```

Command 88 testing retry count configuration

```
SEND Command 74
IF ((Cmd74Rsp.RetryCount < 2) OR (Cmd74Rsp.RetryCount > 5))
   THEN Test result is FAIL                                           (7098)
END IF

SEND Command 88 with RetryCount = 1
CALL VerifyResponseAndByteCount("Passed Parameter too small", 2)

SEND Command 88 with RetryCount = 6
CALL VerifyResponseAndByteCount("Passed Parameter too large", 2)

SEND Command 88 with no data bytes
CALL VerifyResponseAndByteCount("Too Few Data Bytes Received", 3)

SEND Command 88 with retry count = 3 and one extra data byte
CALL VerifyResponseAndByteCount(0, 3)

SEND Command 74
CALL VerifyResponseAndByteCount(0, 10)
IF ((Cmd74Rsp.RetryCount != 3)
   THEN Test result is FAIL                                           (7099)
END IF

END TestCase
```

Test Case D:HART 7 I/O System and Sub-device Statistics

```
CALL IdentifyDevice

IF (UNIV_REVISION > 6) THEN
   CALL VerifyAssociatedCommand(74,75,77,84,85,86,87,88,94,95)
```

```
      ELSE
            THEN Test result is ABORT                                           (7100)
      END IF
```

Use Command 74 to determine capabilities of I/O system.

```
      SEND Command 74
      CALL VerifyResponseAndByteCount(0, 10)
      numDevReported = Cmd74Rsp.NumberofDevicesDetected
      retries = Cmd74Rsp.NumberRetries + 1
```

Keep track of Command 86 responses for each sub-device.

```
      Cmd86[] = null
```

Command 85 stats are by {card, channel} pairs. Since there may be more than one device per {card, channel}, we need to record the number of devices to correctly calculate the expected statistics. The CdChList contains a tuple {Card, Channel, nDevices, Cmd85Rsp}

```
      CdChList[] = null
      NumCdChChannels = 0
```

Record starting statistic values from Command 85 and 86.

```
      FOR n = 0 to numDevReported
         SEND Command 86 with SubDeviceIndex = n+1
         CALL VerifyResponseAndByteCount(0, 9)
         cmd86[n] = Cmd86Rsp

         SEND Command 84 with SubDeviceIndex = n
         CALL VerifyResponseAndByteCount(0, 46)

         SET m = 0
         SET CdChFound = FALSE
         WHILE ((m < NumCdChChannels) AND !CdChFound)
            IF CdChList[m].CardChannel == {Cmd84Rsp.card,Cmd84Rsp.channel}
               THEN SET CdChFound = TRUE
            ELSE
               INCREMENT m
            END IF
         END WHILE

         IF CdChFound THEN
            INCREMENT CdChList[m].nDevices
         ELSE
            CdChList[m].CardChannel = {Cmd84Rsp.card,Cmd84Rsp.channel}
            CdChList[m].nDevices = 1
            SEND Command 85 with Cmd84Rsp.card, Cmd84Rsp.channel
            CALL VerifyResponseAndByteCount(0, 15)
            CdChList[m].Cmd85Rsp = Cmd85Rsp
            INCREMENT NumCdChChannels
         END IF
      END FOR
```

Perform Basic testing on Command 94.

```
SEND Command 94
CALL VerifyResponseAndByteCount(0, 18)
StartCmd94 = Cmd94Rsp

SEND Command 74
CALL VerifyResponseAndByteCount(0, 10)

SEND Command 94
CALL VerifyResponseAndByteCount(0, 18)

IF ( (Cmd94Rsp.NumMsgRcv != (StartCmd94.NumMsgRcv + 2))
  OR  (Cmd94Rsp.NumMsgRsp != (StartCmd94.NumMsgRsp + 2))
  OR  (Cmd94Rsp.NumIOReq != (StartCmd94.NumIOReq))
  OR  (Cmd94Rsp.NumIORsp != (StartCmd94.NumIORsp))
     THEN Test Result is FAIL                                        (7101)
```

Send Command to a non-existent sub-device to cause retries.

```
ReqRspCnt=0
DO
   SEND Command 77 with
      IOCardChannel = CdChList[0].CardChannel,
      Preambles = 5, Delimiter = 0x82,
      Address = 0xF984000000,
      Command = 0, ByteCount = 0, Data = null
   INCREMENT ReqRspCnt
WHILE ( (RESPONSE_CODE == "Busy")
   OR (RESPONSE_CODE == "Delayed Response Initiated")
   OR (RESPONSE_CODE == "Delayed Response Running") )

IF (Cmd77Rsp.Data.ResponseCode == "Success")
   THEN Test Result is FAIL                                          (7102)
END IF
```

Send Command 86 to the I/O System to read the statistics.

```
SEND Command 85 with Cmd85[0].Card, Cmd85[0].channel
CALL VerifyResponseAndByteCount(0, 15)

IF ( (Cmd85Rsp.NumSTX != cmd86[0].NumSTX+retries)
  OR (Cmd85Rsp.NumACK != cmd86[0].NumACK) )
     THEN Test Result is FAIL                                        (7103)
END IF

SET cmd85[0].NumSTX = Cmd85Rsp.NumSTX
```

Check Command 94 again. We only had one I/O Request-Response (Cmd77) but the Delayed Response incremented the Host Interface message count a lot. Note: the I/O Response was an error.

```
SEND Command 94
CALL VerifyResponseAndByteCount(0, 18)

IF ( (Cmd94Rsp.NumMsgRcv != (StartCmd94.NumMsgRcv + 4 + ReqRspCnt))
  OR  (Cmd94Rsp.NumMsgRsp != (StartCmd94.NumMsgRsp + 4 + ReqRspCnt))
  OR  (Cmd94Rsp.NumIOReq != (StartCmd94.NumIOReq + 1))
  OR  (Cmd94Rsp.NumIORsp != (StartCmd94.NumIORsp + 1))
     THEN Test Result is FAIL                                        (7105)
```

Cycle through all sub-devices, sending commands to each to increment the statistics.

```
Cmd3Req = {3, 0, null}
FOR n = 0 to numDevReported
   SEND Command 84 with SubDeviceIndex = n+1
   CALL VerifyResponseAndByteCount(0, 46)

   FOR I = 1 to 25
      SET addr = {Cmd84Rsp.ExpandedDeviceType, Cmd84Rsp.DeviceID }
      IssueCmd77(Cmd84Rsp.card, Cmd84Rsp.channel, addr, Cmd3Req, Cmd3Rsp)
      IF Cmd3Rsp.ResponseCode != "Success"
         THEN Test Result is FAIL                                    (7106)
      END IF
   END FOR
END FOR
```

Test Command 86 communication statistics for device.

```
FOR n = 0 to numDevReported
   SEND Command 86 with index n+1
   CALL VerifyResponseAndByteCount(0, 9)

   IF command86.STX != cmd86[n].STX + 25
     OR command86.ACK != cmd86[n].ACK + 25
      THEN Test Result is FAIL                                       (7107)
   END IF
END FOR
```

Verify communication statistics for I/O channel.

```
FOR n = 0 to NumCdChChannels
   SEND Command 85 with Cmd85[n].CardChannel
   CALL VerifyResponseAndByteCount(0, 15)

   IF ( (Cmd85Rsp.STX != ((Cmd85[n].nDevices*25)+Cmd85[n].Cmd85Rsp.STX))
     OR (Cmd85Rsp.ACK != ((Cmd85[n].nDevices*25)+Cmd85[n].Cmd85Rsp.ACK)))
        THEN Test Result is FAIL                                     (7108)
   END IF
END FOR

END TestCase
```

IssueCmd75(card, channel, device)

Issue command 75 taking care of "Busy" and Delayed Responses.

```
PROCEDURE IssueCmd75(card, channel, device)
DO
   SEND Command 1
   SEND Command 75 (with card, channel, device)
   CALL TestValidFrame()
   IF ( ( (RESPONSE_CODE == "Busy")
     OR (RESPONSE_CODE == "Delayed Response Initiated")
     OR (RESPONSE_CODE == "Delayed Response Running") )
     AND (BYTE_COUNT != 2) )
        THEN Test result is FAIL                                     (7035)
   END IF
WHILE ( (RESPONSE_CODE == "Busy")
  OR (RESPONSE_CODE == "Delayed Response Initiated")
  OR (RESPONSE_CODE == "Delayed Response Running") )
RETURN RESPONSE_CODE

PROCEDURE END
```

IssueCmd77(card, ch, addr, CmdSpec)

Issue command 75 taking care of "Busy" and Delayed Responses.

```
PROCEDURE IssueCmd77(card, ch, addr, CmdSpec, CmdRsp)
DO
   SEND Command 1

   SET del = 0x86
   IF SIZEOF (addr) == 1
      THEN SET del = 0x06
   END IF

   SEND Command 77 with
      IOCard = card
      Channel = ch
      Preambles = 5
      Delimiter = del
      Address = addr
      Command, Byte Count, Data = CmdSpec

   CALL TestValidFrame()
   IF ( ( (RESPONSE_CODE == "Busy")
     OR (RESPONSE_CODE == "Delayed Response Initiated")
     OR (RESPONSE_CODE == "Delayed Response Running") )
     AND (BYTE_COUNT != 2) )
        THEN Test result is FAIL                                    (7031)
   END IF
WHILE ( (RESPONSE_CODE == "Busy")
   OR (RESPONSE_CODE == "Delayed Response Initiated")
   OR (RESPONSE_CODE == "Delayed Response Running") )

SET CmdRsp = Response {Command, Byte Count, Data}

PROCEDURE END
```

7.43 CAL078 Command Aggregation

Command 78 allows a single request to include multiple commands. The resulting response must be a single aggregated response within the size limitations.

Note: The command can be used in burst mode, but this is not explicitly tested in this testcase.

References:

Specification	Rev.	Sections
Common Practice Command Specification	9.0	7.46

Test Procedure

Determine Aggregation Support

```
CALL IdentifyDevice
CALL CheckCommandImplemented(78)
CALL VerifyNotWriteProtected()
```

Configure aggregate command to return command 14 and 15

```
SEND Command 78 with 2 commands including 14 and 15.  Byte count
  for both are 0 with no request data bytes for either command

IF (RESPONSE_CODE != "SUCCESS" )
      THEN Test Result is FAIL                                    (7033)
END IF
```

Verify that the value for Command A matches,

```
IF ( command78response.numcmds != 2 )
      THEN Test result is FAIL                                    (7034)
END IF

IF ( command78response.cmdA != 14)
      THEN Test result is FAIL                                    (7036)
END IF

IF ( command78response.byteCountA != 5)
      THEN Test result is FAIL
END IF
```

and for Command B.

```
IF ( command78response.cmdB != 15 )
      THEN Test result is FAIL                                    (7037)
END IF
IF ( command78response.byteCountB != 13)
      THEN Test result is FAIL                                    (7038)
END IF
```

Configure aggregate command with multiple commands to exceed packetsize. For this, we use an array of aggregated commands numbers. They each have 0 request bytes and no request data.

```
      aggregatedCommands[] = { 0, 7, 8, 12, 13, 14, 15, 16, 20, 48 }
```

Aggregate until we get a response code 30 indicating that the command is truncated.

```
      truncated = 0
      FOR n = 0 to 9
            SEND Command 78 with
                  Number of Commands = n+1
                  Command = aggregatedCommands[n].command
                  ByteCount = 0
                  Data Bytes = 0

            IF (RESPONSE_CODE == "Command Response Truncated" )
                  INCREMENT truncated
            END IF
      END FOR

      IF truncated == 0
            THEN Test Result is FAIL                                      (7039)
      END IF

      END TEST
```

7.44 CAL079 Write Device Variable

Verifies that the DUT responds properly to Command 79. Checks Addresses, Command Number, Response Code and Byte Count for Command 79, Write Device Variable. The following conditions are evaluated for each Device Variable supported by the DUT.

- Send command 9 to Read Device Variables With Status.
- Send command 79 "Fixed Value", "Normal" and with an invalid Command Code.
- Send command 79 with Device Variable code 255.
- Send command 79 with Device Variable status of 255. Send command 9 to read back Device Variable status. The statuses should match.
- Run the following tests for variable code 0 only.
- Send command 79 with Device Variable units code of 255.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

References:

Specification	Rev.	Sections
Common Practice Command Specification	7.0	6.8, 7.45

Test Procedure

```
CALL IdentifyDevice
CALL CheckForRecommendedCommand (79)
CALL VerifyNotWriteProtected()
```

There must be at least one valid Device Variable (otherwise why would the command be implemented?)

```
dVar = -1
IF (FindNextDeviceVariable(dVar) == "No More Device Variables")
      THEN Test result is FAIL                                          (7040)
END IF
```

For each Device Variable, check all of the Response Codes possible

```
DO
```

Send command 9 to Read Device Variables With Status. Retain the status information.

```
      SEND Command 9 (with dVar)
      CALL TestValidFrame
      IF ( UNIV_REVISION > 6 )
            THEN CALL VerifyResponseAndByteCount(0, 15)
      ELSE
            THEN CALL VerifyResponseAndByteCount(0, 11)
      END IF
      SET vClass = Device Variable classification from slot 0
      SET vUnits = Device Variable units from slot 0
      SET vValue = Device Variable value from slot 0
      SET vStatus = Device Variable status from slot 0
```

See if Command 79 works for this Device Variable.

```
      Call IssueCommand79( dVar, "Normal", vUnits, vValue, vStatus)
      CALL TestValidFrame
      IF ( (RESPONSE_CODE == "Device Variable Index Not Allowed")
            IF (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                           (7041)
            END IF
      ELSE
```

Send command 79 "Fixed Value", "Normal" and with an invalid Command Code.

```
      Call IssueCommand79( dVar, "Fixed Value", vUnits, vValue, vStatus)
            CALL TestValidFrame
            CALL VerifyResponseAndByteCount(0, 10)

            Call IssueCommand79( dVar, 255, vUnits, vValue, vStatus)
            CALL VerifyResponseAndByteCount("Invalid Write Device
                Variable Code", 2)
            CALL TestValidFrame
```

Send command 79 with Device Variable units code of 255.

```
      Call IssueCommand79( dVar, "Fixed Value", 255, vValue, vStatus)
            CALL VerifyResponseAndByteCount("Invalid Units Code", 2)
            CALL TestValidFrame
```

Send command 79 with Device Variable status of 255. Send command 9 to read back Device Variable status. The statuses should match.

```
Call IssueCommand79( dVar, "Fixed Value",vUnits,vValue,255)
            CALL VerifyResponseAndByteCount(0, 10)
            CALL TestValidFrame

            SEND Command 9 (with dVar)
            CALL TestValidFrame
            IF ( UNIV_REVISION > 6 )
                  THEN CALL VerifyResponseAndByteCount(0, 15)
            ELSE
                  THEN CALL VerifyResponseAndByteCount(0, 11)
            END IF
            IF (Device Variable status != 255)
                  THEN Test result is FAIL                           (7042)
            END IF

      Call IssueCommand79(dVar, "Normal", vUnits, vValue, vStatus)
            CALL VerifyResponseAndByteCount(0, 10)
            CALL TestValidFrame
      END IF

WHILE (FindNextDeviceVariable(dVar) == "Device Variable Found")
```

Send command 79 with Device Variable code 255.

```
Call IssueCommand79( 255, "Fixed Value", vUnits, vValue, vStatus)
CALL VerifyResponseAndByteCount("Invalid Device Variable Index", 2)
CALL TestValidFrame

END TEST
```

IssueCommand79(dVar, mode, units, value, status)

Issue command 79 taking care of "Busy" and Delayed Responses.

```
PROCEDURE IssueCommand79(dVar, mode, units, value, status)
DO
      SEND Command 1
      SEND Command 79 (with dVar, mode, units, value, status)
      CALL TestValidFrame()
      IF ( ( (RESPONSE_CODE == "Busy")
         OR (RESPONSE_CODE == "Delayed Response Initiated")
         OR (RESPONSE_CODE == "Delayed Response Running") )
         AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                                 (7045)
      END IF
WHILE ( (RESPONSE_CODE == "Busy")
   OR (RESPONSE_CODE == "Delayed Response Initiated")
   OR (RESPONSE_CODE == "Delayed Response Running") )

RETURN RESPONSE_CODE

PROCEDURE END
```

7.45 CAL080 Verify Device Variable Trim Commands

Checks Addresses, Command Number, Response Code and Byte Count for:

- Command 80, Read Device Variable Trim Points
- Command 81, Read Device Variable Trim Guidelines
- Command 82, Write Device Variable Trim Point
- Command 83, Reset Device Variable Trim

First either none or all of the trim commands must be supported. If Command 80 is supported, then support for commands 81-83 is verified. Then the following tests are performed for Device Variable codes 0 – 249:

- Issue Command 81 with an extra data byte.
- Issue Command 80 with an extra data byte.
- Issue Command 81 to read trim point code, minimum lower trim point value, maximum lower trim point value, minimum upper trim point value, minimum upper trim point value, and minimum differential value. Check that the trim point code (byte 1) is 1, 2, or 3.
- Issue Command 80 for the variable. If trim point code is 1 or 2, check that upper trim point value is NaN. Check that the lower trim point value is in range. Check that the upper trim point value is in range. Check that the upper and lower trim point differential is at least the minimum.
- Perform basic tests on Commands 82, 83.

Finally, all of the commands are tested with an invalid Device Variable code to verify that "Invalid Device Variable Index" is returned.

Note: "Too Few Data Bytes Received" Response Code is verified in CAL000.

References:

Specification	**Rev.**	**Sections**
Common Practice Command Specification	7.0	6.5, 7.46-7.49

Test Procedure

```
CALL IdentifyDevice
CALL CheckForRecommendedCommand (80)
```

Verify that the other Trim Commands are implemented

```
SEND Command 81 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                       (7050)
END IF
SEND Command 82 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                       (7051)
END IF
SEND Command 83 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Test result is FAIL                                       (7052)
END IF
```

There must be at least one valid Device Variable (otherwise why would the command be implemented?)

```
dVar = -1
IF (FindNextDeviceVariable(dVar) == "No More Device Variables")
      THEN Test result is FAIL                                       (7053)
END IF
```

Test each Device Variable. First, make sure this Device Variable is valid for the trim commands.

```
DO
      SEND Command 80 (with dVar)
      IF ( (RESPONSE_CODE == "Device Variable Index Not Allowed")
            IF (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                           (7054)
            END IF
      ELSE
            CALL VerifyResponseAndByteCount(0, 12)
            CALL VerifyTrimCommands(dVar)
      END IF
WHILE (FindNextDeviceVariable(dVar) == "Device Variable Found")
```

Verify the "Invalid Device Variable Index" Response Code

```
IssueCommand80(255)
CALL VerifyResponseAndByteCount("Invalid Device Variable Index", 2)

IssueCommand81(255)
CALL VerifyResponseAndByteCount("Invalid Device Variable Index", 2)

IssueCommand80(255, "Lower Trim Point", 0.0, 0)
CALL VerifyResponseAndByteCount("Invalid Device Variable Index", 2)

IssueCommand83(255)
CALL VerifyResponseAndByteCount("Invalid Device Variable Index", 2)

END TEST
```

VerifyTrimCommands(dVar)

Verifies operation of Commands 81-83

```
      PROCEDURE VerifyTrimCommands(dVar)
```

Issue Command 81 with an extra data byte. This reads the trim point code, minimum lower trim point value, maximum lower trim point value, minimum upper trim point value, maximum upper trim point value, and minimum differential value.

```
      DO
            SEND Command 1
            SEND Command 81 (with dVar) with an extra data byte
            CALL TestValidFrame()
            IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                           (7055)
            END IF
      WHILE (RESPONSE_CODE == "Busy")
      CALL VerifyResponseAndByteCount(0, 25)
      SET nPoints = trim points supported  (from Command 81)
      SET ltpMin = minimum lower trim point value
      SET ltpMax = maximum lower trim point value
      SET utpMin = minimum upper trim point value
      SET utpMax = maximum upper trim point value
      SET minDiff = minimum differential
      SET units = trim point units code
```

Issue Command 80 for the variable. If trim point code is 1 or 2, check that upper trim point value is NaN. Check that the lower trim point value is in range. Check that the upper trim point value is in range. Check that the upper and lower trim point differential is, at least, the minimum.

```
      DO
            SEND Command 1
            SEND Command 80 (with dVar) with an extra data byte
            CALL TestValidFrame()
            IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                           (7056)
            END IF
      WHILE (RESPONSE_CODE == "Busy")
      CALL VerifyResponseAndByteCount(0, 12)
      SET ltpValue = lower trim point value
      SET utpValue = upper trim point value

      SWITCH on (nPoints)

            CASE "Lower Trim Point Supported"
                  IF (utpValue != "0x7F, 0xA0, 0x00, 0x00")
                        THEN Test result is FAIL                     (7057)
                  END IF
                  SET delta = ltpMax - ltpMin
                  IF (ltpValue > ltpMax + 0.25*delta)
                        THEN Test result is FAIL                     (7058)
                  END IF
                  IF (ltpValue < ltpMin - 0.25*delta)
                        THEN Test result is FAIL                     (7059)
                  END IF
            CASE "Upper Trim Point Supported"
                  IF (ltpValue != "0x7F, 0xA0, 0x00, 0x00")
                        THEN Test result is FAIL                          (7060)
                  END IF
                  SET delta = utpMax - utpMin
                  IF (utpValue > utpMax + 0.25*delta)
                        THEN Test result is FAIL                          (7061)
                  END IF
                  IF (utpValue < utpMin - 0.25*delta)
                        THEN Test result is FAIL                          (7062)
                  END IF

            CASE "Lower And Upper Trim Point Supported"
                  SET delta = ltpMax - ltpMin
                  IF (ltpValue > ltpMax + 0.25*delta)
                        THEN Test result is FAIL                          (7063)
                  END IF
                  IF (ltpValue < ltpMin - 0.25*delta)
                        THEN Test result is FAIL                          (7064)
                  END IF
                  SET delta = utpMax - utpMin
                  IF (utpValue > utpMax + 0.25*delta)
                        THEN Test result is FAIL                          (7065)
                  END IF
                  IF (utpValue < utpMin - 0.25*delta)
                        THEN Test result is FAIL                          (7066)
                  END IF
                  IF ( (utpValue - ltpValue ) < 1.10 * minDiff)
                        THEN Test result is FAIL                          (7067)
                  END IF

            CASE DEFAULT
                  Test result is FAIL                                     (7070)
      END SWITCH
```

Do a basic test of Command 82. First, if lower trim point supported, try its min and max values

```
      IF ( (nPoints == "Lower Trim Point Supported")
      OR (nPoints = "Lower And Upper Trim Point Supported") ) THEN
            CALL IssueCommand82(dVar, "Lower Trim Point", units,(ltpMax+minDiff)
                  )
            CALL VerifyResponseAndByteCount("Passed Parameter Too Large",2)

            CALL IssueCommand82(dVar,"Lower Trim Point",units,(ltpMin-minDiff) )
            CALL VerifyResponseAndByteCount("Passed Parameter Too Small", 2)
      END IF
```

If upper trim point supported, try its min and max values

```
      IF ( (nPoints == "Upper Trim Point Supported")
      OR (nPoints = "Lower And Upper Trim Point Supported") ) THEN
            CALL IssueCommand82(dVar,"Upper Trim Point",units,(utpMax+minDiff) )
            CALL VerifyResponseAndByteCount("Passed Parameter Too Large",2)

            CALL IssueCommand82(dVar,"Upper Trim Point",units,(utpMin-minDiff) )
            CALL VerifyResponseAndByteCount("Passed Parameter Too Small", 2)
      END IF
```

Do a basic test of Command 83.

```
      DO
            SEND Command 1
            SEND Command 83 (with dVar) with an extra data byte
            CALL TestValidFrame()
            IF ( ( (RESPONSE_CODE == "Busy")
               OR (RESPONSE_CODE == "Delayed Response Initiated")
               OR (RESPONSE_CODE == "Delayed Response Running") )
               AND (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                              (7075)
            END IF
      WHILE ( (RESPONSE_CODE == "Busy")
         OR (RESPONSE_CODE == "Delayed Response Initiated")
         OR (RESPONSE_CODE == "Delayed Response Running") )
      CALL VerifyResponseAndByteCount(0, 3)
      PROCEDURE END
```

IssueCommand80(dVar)

Issue command 80 taking care of "Busy"

```
      PROCEDURE IssueCommand80(dVar)
      DO
            SEND Command 1
            SEND Command 80 (with dVar)
            CALL TestValidFrame()
            IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                              (7076)
            END IF
      WHILE (RESPONSE_CODE == "Busy")

      RETURN RESPONSE_CODE

      PROCEDURE END
```

IssueCommand81(dVar)

Issue command 81 taking care of "Busy"

```
      PROCEDURE IssueCommand81(dVar)
      DO
            SEND Command 1
            SEND Command 81 (with dVar)
            CALL TestValidFrame()
            IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                              (7077)
            END IF
      WHILE (RESPONSE_CODE == "Busy")

      RETURN RESPONSE_CODE

      PROCEDURE END
```

IssueCommand82(dVar, trPoint, units, value)

Issue command 82 taking care of "Busy" and Delayed Responses.

```
PROCEDURE IssueCommand82(dVar, trPoint, units, value)
DO
      SEND Command 1
      SEND Command 82 (with dVar, trPoint, units, value)
      CALL TestValidFrame()
      IF ( ( (RESPONSE_CODE == "Busy")
         OR (RESPONSE_CODE == "Delayed Response Initiated")
         OR (RESPONSE_CODE == "Delayed Response Running") )
         AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                                   (7078)
      END IF
WHILE ( (RESPONSE_CODE == "Busy")
   OR (RESPONSE_CODE == "Delayed Response Initiated")
   OR (RESPONSE_CODE == "Delayed Response Running") )

RETURN RESPONSE_CODE

PROCEDURE END
```

IssueCommand83(dVar)

Issue command 83 taking care of "Busy" and Delayed Responses.

```
PROCEDURE IssueCommand83(dVar)
DO
      SEND Command 1
      SEND Command 83 (with dVar)
      CALL TestValidFrame()
      IF ( ( (RESPONSE_CODE == "Busy")
         OR (RESPONSE_CODE == "Delayed Response Initiated")
         OR (RESPONSE_CODE == "Delayed Response Running") )
         AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                                   (7079)
      END IF
WHILE ( (RESPONSE_CODE == "Busy")
   OR (RESPONSE_CODE == "Delayed Response Initiated")
   OR (RESPONSE_CODE == "Delayed Response Running") )

RETURN RESPONSE_CODE

PROCEDURE END
```

7.46 CAL091 Trending

Trending allows the collection of monotonically spaced data samples for a specified Device Variable to be acquired. This test will verify the configuration and execution of trends. Trending requires support of all 3 trend commands:

- Command 91 Read Trend Configuration
- Command 92 Write Trend Configuration
- Command 93 Read Trend

References:

Specification	Rev.	Sections
Common Practice Command Specification	9.0	6.11, 7.59, 7.60, 7.61

Test Case A: Basic Trending Operation

Check for Trending Support

```
CALL IdentifyDevice
CALL VerifyAssociatedCommands(91, 92, 93, 54)
CALL VerifyNotWriteProtected()
```

Trending is only supported in HART 7 or later. At this point, we have verified the device supports trending, so we check to be sure it is appropriate protocol revision.

```
IF UNIV_REVISION < 7 THEN
   TEST Result is FAIL                                             (7590)
END IF
```

Find Valid Trend numbers for the device under test.

```
SEND Command 91 with trendNum=0 to read TotalNumTrends
CALL VerifyResponseAndByteCount(0, 10)
```

Find Update time for PV.

```
SEND Command 54 with 245 (PV)
pvUpdateTime = Command54.updateTimePeriod
```

Start with trend number 0 and work up from there.

```
FOR (trendNum = 0 to TotalNumTrends)
   SEND Command 91 with trendNum
   CALL VerifyResponseAndByteCount(0, 10)
```

Trend values are not reset when the trend is disabled. Reseting the ring buffer only happens on errors and if the trend configuration changes. Toggle the Device Variable Code to reset values.

```
   IssueCmd92(trendNum, Disabled, 245, pvUpdateTime*2)
   IssueCmd92(trendNum, Disabled, 245, pvUpdateTime)

   SEND Command 93 with trendNum
```

Verify the 12 values in trend ring buffer.

```
         FOR n = 0 to 12
            IF Command93.trendValue[n] != NaN
               THEN Test Result Is FAIL                                  (7591)
            END IF
               IF Command93.trendStatus[n] != bad-fixed
               THEN Test Result Is FAIL                                  (7592)
            END IF
         END FOR
      END FOR

      SEND Command 91 with Trend number = TotalNumTrends
      CALL VerifyResponseAndByteCount("Invalid Trend Number", 2)

      SEND Command 93 with Trend number = TotalNumTrends
      CALL VerifyResponseAndByteCount("Invalid Trend Number", 2)
```

Configure Trend of single device variable using Primary Value device variable.

```
      IssueCmd92(0, 1, 245, pvUpdateTime)

      WAIT 5.5* pvUpdateTime
      SEND Command 93 with Trend Number = 0
      IF TrendValue[4] == NaN
         THEN test result is FAIL                                        (7593)
      END IF
      IF TrendValue[5] != NaN
         THEN test result is FAIL                                        (7594)
      END IF

      WAIT 6.5* pvUpdateTime
      CALL TestTrend (0)
```

Disable the trend. Values in Command 92 must stay but not update.

```
      SEND Command 92 with
         Trend Number = 0
         Trend Control Code = Disabled
         Device Variable code =245
         Trend sample period = pvUpdateTime
      CALL VerifyResponseAndByteCount(0, 9)

      Send Command 93 with TrendNumber = 0
      SET CurrentTime0 = Cmd93Rsp.TimeStamp

      WAIT 2* pvUpdateTime

      Send Command 93 with TrendNumber = 0
      IF Cmd91Rsp.TimeStamp != Time0
         THEN test result is FAIL                                        (7595)
      END IF

      IssueCmd92(0, 0, 246, pvUpdateTime)

      END TEST CASE
```

Test Case B: Basic Command 92 Testing

Check for Trending Support

```
      CALL IdentifyDevice
      CALL VerifyAssociatedCommands(91, 92, 93, 54)
      CALL VerifyNotWriteProtected()
```

Find Valid Trend numbers for the device under test.

```
      SEND Command 91 with trendNum=0 to read TotalNumTrends
      CALL VerifyResponseAndByteCount(0, 10)
```

Find Update time for PV.

```
      SEND Command 54 with 245 (PV)
      pvUpdateTime = Command54.updateTimePeriod

      SEND Command 92 with
         Trend number = TotalNumTrends
         Trend Control Code = 1
         Device Variable code =246
         Trend sample period = pvUpdateTime
      CALL VerifyResponseAndByteCount("Invalid Trend Number", 2)
```

Maximum interval between two values in a trend is limited to range {pvUpdateTime - 2hours}.

```
      FOR SamplePeriod = {2.01 hours, pvUpdateTime/2}
         SEND Command 92 with
            Trend number = 0
            Trend Control Code = 1
            Device Variable code =245
            Trend sample period = SamplePeriod
```

Error Response

```
         IF (RESPONSE CODE == "Passed parameter too small")
           OR (RESPONSE CODE == "Passed parameter too large") THEN
            SEND Command 91 with Trend Number = 0
            IF Cmd91Rsp.TrendControl != "Disabled"
               THEN Test result is fail                                 (7596)
            END IF
            SEND Command 93 with Trend Number = 0
            IF Cmd93Rsp.TrendValue[0] != NaN
               THEN Test result is fail                                 (7597)
            END IF
```

Warning Response

```
         ELSE IF (RESPONSE CODE == "Set to nearest possible value"
            IF Cmd91Rsp.TrendControl == "Disabled"
               THEN Test result is fail                                 (7600)
            END IF
            SEND Command 93 with Trend Number = 0
            WAIT pvUpdateTime
            IF Cmd93Rsp.TrendValue[0] == NaN
               THEN Test result is fail                                 (7601)
            END IF

         ELSE
            Test Result is FAIL                                         (7602)
         END IF
      END FOR
```

Check with good sample intervals

```
      FOR SamplePeriod = {2.00 hours, pvUpdateTime }
         IssueCmd92 (0, 1, 245, SamplePeriod)

         WAIT pvUpdateTime
         SEND Command 93 with Trend Number = 0
         IF Cmd93Rsp.TrendValue[0] == NaN
            THEN Test result is fail                                      (7603)
         END IF
         IssueCmd92 (0, Disabled, 246, SamplePeriod)
      END FOR
```

Check with bad device variable code

```
      SEND Command 92 with
            Trend number = 0
            Trend Control Code = 1
            Device Variable code =254
            Trend sample period = SamplePeriod
      CALL VerifyResponseAndByteCount("Invalid device variable index", 2)

      SEND Command 91 with Trend Number = 0
      IF Cmd91Rsp.TrendControl != "Disabled"
         THEN Test result is fail                                         (7605)
      END IF
      SEND Command 93 with Trend Number = 0
      IF Cmd93Rsp.TrendValue[0] != NaN
         THEN Test result is fail                                         (7606)
      END IF
```

Check trend control codes

```
      SET SamplePeriod = pvUpdatePeriod
      IF pvUpdateTime < 5 seconds
         THEN SET SamplePeriod = 5 seconds
      END IF

      FOR TrendControl = {2 to 4}

         SEND Command 92 with
            Trend number = 0
            Trend Control Code = TrendControl
            Device Variable code =245
            Trend sample period = SamplePeriod

         SWITCH on TrendControl
         CASE 2:
            CALL VerifyResponseAndByteCount(0, 9)

         CASE 3:
            IF (RESPONSE_CODE != "Success") AND
              (RESPONSE_CODE != "Invalid selection")
                 THEN Test result is fail                                 (7607)
            END IF

         CASE 4:
            CALL VerifyResponseAndByteCount("Invalid selection", 2)

         END SWITCH

         IF RESPONSE_CODE = "Success" THEN
            CheckCmd91Setting(0, TrendControl, 245, SamplePeriod)

         ELSE
            SEND Command 91 with Trend Number = 0
            IF Cmd91Rsp.TrendControl != "Disabled"
               THEN Test result is fail                                   (7608)
            END IF
         END IF
```

Trend is averaged or disabled. Either way there should be no sample yet.

```
         SEND Command 93 with Trend Number = 0
         IF Cmd93Rsp.TrendValue[0] != NaN
            THEN Test result is fail                                    (7609)
         END IF

      END FOR

      END TEST CASE
```

Test Case C: Support for multiple trends.

Four scenarios are exercised: (1) two trends with identical configurations; (2) Same Device Variable s with different sample rates; (3) Different Device Variable with different sample rates; and (4) Different Device Variable with same sample rates. First, check support for multiple trends.

```
      CALL IdentifyDevice
      CALL VerifyAssociatedCommands(91, 92, 93, 54)
      CALL VerifyNotWriteProtected()
```

Does the DUT support multiple trends?

```
      SEND Command 91 with trendNum=0 to read TotalNumTrends
      CALL VerifyResponseAndByteCount(0, 10)

      IF TotalNumTrends == 1
         THEN Test result is ABORT                                        (5000)
      END IF
```

Find Update time for PV and SV

```
      SEND Command 54 with 245 (PV)
      pvUpdateTime = Command54.updateTimePeriod
      SEND Command 54 with 246 (SV)
      svUpdateTime = Command54.updateTimePeriod

      SET UpdatePeriod = 5 seconds
      IF pvUpdateTime > UpdatePeriod
         THEN SET UpdatePeriod = pvUpdateTime
      END IF
      IF svUpdateTime > UpdatePeriod
         THEN SET UpdatePeriod = svUpdateTime
      END IF

      IssueCmd92 (0, 0, 246, UpdatePeriod*2)
      IssueCmd92 (1, 0, 246, UpdatePeriod*2)
```

Scenario 1: Set them up with identical configurations

```
      IssueCmd92 (0, 0, 245, UpdatePeriod)
      IssueCmd92 (1, 0, 245, UpdatePeriod)
      CheckCmd91Setting (0, 0, 245, UpdatePeriod)
      CheckCmd91Setting (1, 0, 245, UpdatePeriod)

      WAIT 5.5* UpdatePeriod

      SEND Command 93 with Trend Number = 0
      IF TrendValue[4] == NaN
         THEN test result is FAIL                                         (7612)
      END IF
      IF TrendValue[5] != NaN
         THEN test result is FAIL                                         (7613)
      END IF

      SEND Command 93 with Trend Number = 1
      IF TrendValue[4] == NaN
         THEN test result is FAIL                                         (7614)
      END IF
      IF TrendValue[5] != NaN
         THEN test result is FAIL                                         (7615)
      END IF

      WAIT 6.5* UpdatePeriod
      CALL TestTrend (0)
      CALL TestTrend (1)
```

Scenario 2: Same Device Variable, but different sample rates.

```
      IssueCmd92 (0, 0, 246, UpdatePeriod)
      IssueCmd92 (1, 0, 246, UpdatePeriod*2)
      CheckCmd91Setting (0, 0, 246, UpdatePeriod)
      CheckCmd91Setting (1, 0, 246, UpdatePeriod*2)

      WAIT 5.5* UpdatePeriod

      SEND Command 93 with Trend Number = 0
      IF TrendValue[5] == NaN
         THEN test result is FAIL                                     (7617)
      END IF
      IF TrendValue[6] != NaN
         THEN test result is FAIL                                     (7618)
      END IF

      SEND Command 93 with Trend Number = 1
      IF TrendValue[2] == NaN
         THEN test result is FAIL                                     (7619)
      END IF
      IF TrendValue[3] != NaN
         THEN test result is FAIL                                     (7620)
      END IF

      WAIT 6.5* UpdatePeriod
      CALL TestTrend (0)

      SEND Command 93 with Trend Number = 1
      IF TrendValue[5] == NaN
         THEN test result is FAIL                                     (7621)
      END IF
      IF TrendValue[6] != NaN
         THEN test result is FAIL                                     (7622)
      END IF

      WAIT 12* UpdatePeriod
      CALL TestTrend (0)
      CALL TestTrend (1)
```

Scenario 3: Different Device Variables, different sample rates.

```
      IssueCmd92 (0, 0, 245, UpdatePeriod*2)
      IssueCmd92 (1, 0, 246, UpdatePeriod)
      CheckCmd91Setting (0, 0, 245, UpdatePeriod*2)
      CheckCmd91Setting (1, 0, 246, UpdatePeriod)

      WAIT 5.5* UpdatePeriod

      SEND Command 93 with Trend Number = 1
      IF TrendValue[5] == NaN
         THEN test result is FAIL                                     (7625)
      END IF
      IF TrendValue[6] != NaN
         THEN test result is FAIL                                     (7626)
      END IF

      SEND Command 93 with Trend Number = 0
      IF TrendValue[2] == NaN
      THEN test result is FAIL                                          (7627)
   END IF
   IF TrendValue[6] != NaN
      THEN test result is FAIL                                          (7628)
   END IF

   WAIT 6.5* UpdatePeriod
   CALL TestTrend (0)

   SEND Command 93 with Trend Number = 0
   IF TrendValue[5] == NaN
      THEN test result is FAIL                                          (7629)
   END IF
   IF TrendValue[6] != NaN
      THEN test result is FAIL                                          (7630)
   END IF

   WAIT 12* UpdatePeriod
   CALL TestTrend (0)
   CALL TestTrend (1)
```

Scenario 4: Different Device Variables, same sample rates.

```
   IssueCmd92 (0, 0, 246, UpdatePeriod)
   IssueCmd92 (1, 0, 245, UpdatePeriod)
   CheckCmd91Setting (0, 0, 246, UpdatePeriod)
   CheckCmd91Setting (1, 0, 245, UpdatePeriod)

   WAIT 5.5* UpdatePeriod

   SEND Command 93 with Trend Number = 0
   IF TrendValue[4] == NaN
      THEN test result is FAIL                                          (7632)
   END IF
   IF TrendValue[5] != NaN
      THEN test result is FAIL                                          (7633)
   END IF

   SEND Command 93 with Trend Number = 1
   IF TrendValue[4] == NaN
      THEN test result is FAIL                                          (7634)
   END IF
   IF TrendValue[5] != NaN
      THEN test result is FAIL                                          (7635)
   END IF

   WAIT 6.5* UpdatePeriod
   CALL TestTrend (0)
   CALL TestTrend (1)

   END TEST CASE
```

IssueCmd92 (TrendNum, Ctrl, DVar, Period)

Issues Command 92 and verifies setting using Command 91.

```
PROCEDURE IssueCmd92 (TrendNum, Ctrl, DVar, Period)
   SEND Command 92 with
      Trend number = TrendNum
      Trend Control Code = Ctrl
      Device Variable code = DVar
      Trend sample period = Period
   CALL VerifyResponseAndByteCount(0, 9)

   CALL CheckCmd91Setting (TrendNum, Ctrl, DVar, Period)

END PROCEDURE
```

CheckCmd91Setting (TrendNum, Ctrl, DVar, Period)

Issues Command 91 and verifies its return values against the passed parameters.

```
PROCEDURE IssueCmd92 (TrendNum, Ctrl, DVar, Period)
   SEND Command 91 with TrendNum
   CALL VerifyResponseAndByteCount(0, 9)

   IF Cmd91Rsp.TrendControl != Ctrl
      THEN Test result is fail                                  (7636)
   END IF
   IF Cmd91Rsp.DeviceVariableCode != DVar
      THEN Test result is fail                                  (7637)
   END IF
   IF Cmd91Rsp.SampleInterval != Period
      THEN Test result is fail                                  (7638)
   END IF

END PROCEDURE
```

TestTrend (TrendNum)

Verifies Command 93 response against Command 9 and 91.

```
PROCEDURE TestTrend (TrendNum)

   SEND Command 91 with TrendNum

   Send Command 9 with Cmd91Rsp.DeviceVariableCode
   SET CurrentTime0 = Cmd9Rsp.TimeStamp

   SEND Command 93 with TrendNum

   Send Command 9 with Cmd91Rsp.DeviceVariableCode
   SET CurrentTime1 = Cmd9Rsp.TimeStamp
```

Check core Command 93 parameters.

```
   IF Cmd93Rsp.DeviceVariableCode != Cmd91Rsp.DeviceVariableCode
      THEN Test result is fail                                      (7650)
   END IF
   Cmd93Rsp.Classification != Cmd9Rsp.Slot0.Classification
      THEN Test result is fail                                      (7651)
   END IF
   Cmd93Rsp.UnitCode != Cmd9Rsp.Slot0.UnitCode
      THEN Test result is fail                                      (7652)
   END IF
   Cmd93Rsp.SampleInterval != Cmd91Rsp.SampleInterval
      THEN Test result is fail                                      (7653)
   END IF
```

Do some basic sanity checks on the time stamp.

```
   IF Cmd93Rsp.TimeStamp < (CurrentTime0 - Cmd91Rsp.SampleInterval)
      THEN Test result is fail                                      (7654)
   END IF
   IF Cmd93Rsp.TimeStamp > CurrentTime1
      THEN Test result is fail                                      (7655)
   END IF
```

Verifies each trend value is a valid floating-point value and the status. Assumes Device Variable value is basically stable (unchanging) during the test.

```
   FOR n = 0 to 11
      IF Command93.TrendValue[n] != Cmd9Rsp.Slot0.Value
         THEN Test Result Is FAIL                                   (7656)
      END IF
      IF Command93.TrendStatus[n] != Cmd9Rsp.Slot0.Status
         THEN Test Result Is FAIL                                   (7657)
      END IF
   END FOR
END PROCEDURE
```

7.47 CAL101 I/O Subsystem Burst Mode

I/O sub-systems, multiplexors, and wireless adapters should allow configuration of a burst message even if the sub-device does not support burst mode. The I/O sub-system will handle the token arbitration and the update of the data from the device.

The following commands control burst mode of sub-devices on I/O subsystems:

- Command 101 Read Sub-device to Burst Message Map
- Command 102 Map Sub-device to Burst Message
- Command 108 Write Burst Mode Command Number
- Command 109 Burst Mode Control

Although the I/O subsystem must support devices of any HART revision, the sub-device used on the I/O system for this test must be HART 6 or later.

References:

Specification	**Rev.**	**Sections**
Common Practice Command Specification	9.0	7.68, 7.69, 7.75, 7.76

Test Procedure

```
      CALL IdentifyDevice
```

This testcase only applies to "Protocol Bridge Device"

```
      SEND Command 0
      IF (FLAGS != "Protocol Bridge Device")
            THEN ABORT "DEVICE IS NOT A Bridge device"
      END IF
```

Sub-device burst mapping requires burst mode.

```
      IF (UNIV_REVISION > 6) THEN
         CALL VerifyAssociatedCommand(101, 102, 103, 104, 105, 107, 108, 109, 74,
            75, 77, 84)
      ELSE
            THEN Test result is ABORT                                    (7480)
      END IF
```

The test needs at least one sub-device attached to the I/O system to test burst mode. If one sub-device is attached, the system should support a minimum of 6 burst messages. (4 burst and 2 events)

Use Command 74 to determine capabilities of I/O system.

```
      SEND Command 74

      maxNumSubDevSupported = Command74.numSubDevices *
                              Command74.numChannels *
                              Command74.numCards

      IF maxNumSubDevSupported < 2
         THEN WARNING: Support for more than one sub-device is recommended.
      END IF
      numDevAttached = Command74.NumberofDevicesDetected
```

Create two arrays of device information records to hold device information.

```
      devIndex[]= null
      n = 0
```

Proper testing requires 2 or more sub-devices. If only one is present, we can test anyway.

```
      IF numDevAttached < 2 THEN
      PRINT: WARNING: Fewer than 2 sub-devices detected.  Full testing for
            gateways, multiplexors, and other I/O systems should include all
            possible sub-devices.
      END IF
```

We will use burst update rates of 4 seconds for all burst messages.

Map a burst message of Command 48 from the I/O system.

```
      SEND Command 108 with command = 48 and burstMessage = 0
      CALL VerifyResponseAndByteCount(0, 4)
      SEND Command 102 with sub-device index = 0 and burstMessage = 0
      CALL VerifyResponseAndByteCount(0, 5)
```

Map the commands to burst messages and the burst messages to the sub-device(s).

```
      SEND Command 108 with command = 1 and burstMessage = 1
      CALL VerifyResponseAndByteCount(0, 4)
      SEND Command 102 with sub-device index = 1 and burstMessage = 1
      CALL VerifyResponseAndByteCount(0, 5)

      SEND Command 108 with command = 2 and burstMessage = 2
      CALL VerifyResponseAndByteCount(0, 4)
      SEND Command 102 with sub-device index = 1 and burstMessage = 2
      CALL VerifyResponseAndByteCount(0, 5)

      SEND Command 108 with command = 3 and burstMessage = 3
      CALL VerifyResponseAndByteCount(0, 4)
      SEND Command 102 with sub-device index = 1 and burstMessage = 3
      CALL VerifyResponseAndByteCount(0, 5)

      maxMessage = 3

      IF numDevAttached >= 2 THEN
         SEND Command 108 with command 1 and burstMessage 4
         CALL VerifyResponseAndByteCount(0, 4)
         SEND Command 102 with sub-device index 2 and burstMessage 4
         CALL VerifyResponseAndByteCount(0, 5)

         SEND Command 108 with command 2 and burstMessage 5
         CALL VerifyResponseAndByteCount(0, 4)
         SEND Command 102 with sub-device index 2 and burstMessage 5
         CALL VerifyResponseAndByteCount(0, 5)

         SEND Command 108 with command = 3 and burstMessage = 6
         CALL VerifyResponseAndByteCount(0, 4)
         SEND Command 102 with sub-device index = 2 and burstMessage = 6
         CALL VerifyResponseAndByteCount(0, 5)

         maxMessage = 6
      END IF

      IF numDevAttached >= 3 THEN
      SEND Command 108 with command = 1 and burstMessage = 7
      CALL VerifyResponseAndByteCount(0, 4)
      SEND Command 102 with sub-device index = 3 and burstMessage = 7
      CALL VerifyResponseAndByteCount(0, 5)

      SEND Command 108 with command = 2 and burstMessage = 8
      CALL VerifyResponseAndByteCount(0, 4)
      SEND Command 102 with sub-device index = 3 and burstMessage = 8
      CALL VerifyResponseAndByteCount(0, 5)

      SEND Command 108 with command = 3 and burstMessage = 9
      CALL VerifyResponseAndByteCount(0, 4)
      SEND Command 102 with sub-device index = 3 and burstMessage = 9
      CALL VerifyResponseAndByteCount(0, 5)

      maxMessage = 9
   END IF
```

Read the burst message map to verify that the mapping configuration is as desired.

```
   SEND Command 101 with burstMessage = 0
   CALL VerifyResponseAndByteCount(0, 5)
   IF sub-device index != 0 THEN
      Test Result Is FAIL                                            (7481)
   END IF

   SEND Command 101 with burstMessage = 1
   CALL VerifyResponseAndByteCount(0, 5)
   IF sub-device index != 1 THEN
      Test Result Is FAIL                                            (7482)
   END IF

   SEND Command 101 with burstMessage = 2
   CALL VerifyResponseAndByteCount(0, 5)
   IF sub-device index != 1 THEN
      Test Result Is FAIL                                            (7483)
   END IF

   SEND Command 101 with burstMessage = 3
   CALL VerifyResponseAndByteCount(0, 5)
   IF sub-device index != 1 THEN
      Test Result Is FAIL                                            (7484)
   END IF

   IF (numDevAttached >= 2 )
      SEND Command 101 with burstMessage = 4
      CALL VerifyResponseAndByteCount(0, 5)
      IF sub-device index != 2 THEN
         Test Result Is FAIL                                         (7485)
      END IF

      SEND Command 101 with burstMessage = 5
      CALL VerifyResponseAndByteCount(0, 5)
      IF sub-device index != 2 THEN
         Test Result Is FAIL                                         (7486)
      END IF

      SEND Command 101 with burstMessage = 6
      CALL VerifyResponseAndByteCount(0, 5)
      IF sub-device index != 2 THEN
         Test Result Is FAIL                                         (7487)
      END IF
      END IF

      IF (numDevAttached >= 3 )
         SEND Command 101 with burstMessage = 7
         CALL VerifyResponseAndByteCount(0, 5)
         IF sub-device index != 3 THEN
            Test Result Is FAIL                                    (7488)
         END IF

         SEND Command 101 with burstMessage = 8
         CALL VerifyResponseAndByteCount(0, 5)
         IF sub-device index != 3 THEN
            Test Result Is FAIL                                    (7489)
         END IF

         SEND Command 101 with burstMessage = 9
         CALL VerifyResponseAndByteCount(0, 5)
         IF sub-device index != 3 THEN
            Test Result Is FAIL                                    (7490)
         END IF
      END IF
```

Verify that the burst message trigger is continuous and update rate is 4 second for each message.

```
      FOR n= 0 to maxMessage
         SEND Command 104 with
            Burst Message = n
            Burst Trigger Mode = "Continuous"
            Device Variable Classification = 0
            Units Code = "None"
            Trigger Level = NaN
         CALL VerifyResponseAndByteCount(0, 10)

         SEND Command 103
            Burst Message = n
            UpdatePeriod = 4 Second
            MaxUpdatePeriod = 4 Second
         CALL VerifyResponseAndByteCount(0, 11)
      END FOR
```

Turn on Burst Mode in I/O System

```
      FOR n = 0 to maxMessage
         SEND Command 109 to enable burst mode on Burst Message n
         CALL VerifyResponseAndByteCount(0, 4)
      END FOR
```

Receive Burst Messages and verify originated from correct device.

Setup some counters for metrics on received burst messages.

```
      C1 = 0
      C2 = 0
      C3 = 0
      C48 = 0
```

Receive burst messages for 2 minutes to catch multiple publications of each command, so we verify receipt of all burst messages in proper proportion.

```
      FOR (120 seconds)
         CAPTURE BURST
         SWITCH on BURST Command Number
         CASE 77:
            SWITCH on Cmd77Rsp.Command

            CASE 1:
               INCREMENT C1

            CASE 2:
               INCREMENT C2

            CASE 3:
               INCREMENT C3

            DEFAULT:
               Test Result is FAIL                                         (7495)

         CASE 48:
            INCREMENT C48

         DEFAULT:
            Test Result is FAIL                                            (7496)
         END CASE
      END FOR

      IF C48 < 30
         THEN Test result is fail                                          (7497)
      totalReceived = C1+C2+C3
      IF C1/totalReceived < 0.3 OR
         C2/totalReceived < 0.3 OR
         C3/totalReceived < 0.3
         THEN TEST result is FAIL                                          (7498)
      END IF
```

Save the information on the devices we find.

```
      FOR n = 0 to numDevAttached+1
         SEND Command 84 with SubDeviceIndex = n
         CALL VerifyResponseAndByteCount(0, 46)
         devIndex[n].addr = {Cmd84Rsp.ExpandedDeviceType,Cmd84Rsp.DeviceID}
         devIndex[n].card = Cmd84Rsp.card
         devIndex[n].channel = Cmd84Rsp.channel
      END FOR
```

Remove a sub-device.

```
      SEND Command 84 with SubDeviceIndex = 1
      PRINT "Remove the sub-device with tag = Cmd84Rsp.LongTag from the I/O
            System"
      DO
         IF elapsed time > 30 Seconds
            THEN TEST result is FAIL                                       (7500)
         END IF
         SEND Command 48
      WHILE (Cmd48Rsp."Sub-Device List Changed" NOT SET)

      SET Cmd20Req = {Cmd = 20, Byte Count = 0, Data = null}
      IssueCmd77( devIndex[n].card, devIndex[n].channel, devIndex[n].addr,
        Cmd20Req, Cmd20Rsp)

      IF (Cmd20Rsp.Data.ResponseCode == "Success")
         THEN Test Result is FAIL                                          (7501)
      END IF
```

Verify that the device that was at index 2 is now at index 1. Make sure we had at least two devices in the beginning.

```
      IF numDevAttached > 1 THEN
         SEND Command 84 with SubDeviceIndex = 1

         IF (devIndex[2].addr !=  {Cmd84Rsp.ExpandedDeviceType Cmd84Rsp.deviceID}
            THEN Test result is Fail                                       (7505)
         END IF
```

Verify that the burst mapping was properly remapped.

```
         SEND Command 101 with message = 4
         IF (sub-device index != 1)
            THEN Test Result is Fail                                       (7506)
         END IF

         SEND Command 101 with message = 5
         IF (sub-device index != 1)
            THEN Test Result is Fail                                       (7507)
         END IF

         SEND Command 101 with message = 6
         IF (sub-device index != 1)
            THEN Test Result is Fail                                       (7508)
         END IF
      END IF
```

Cycle Power on I/O System to verify sub-devices remap.

```
      PROMPT: “Please remove power to the I/O System.\n Do not
              re-apply power yet!"
```

Verify communications loss

```
      DO
         SEND Command 0
      WHILE COMMUNICATIONS_ERROR != "No Response")
```

Power up system.

```
      PROMPT: “Please apply power to the DUT."
```

Wait for Device restart.

```
      DO
         SEND Command 0
      WHILE (COMMUNICATIONS_ERROR == "No Response")
```

We should receive no burst messages from the disconnected device.

```
      FOR (120 seconds)
         CAPTURE BURST
         SWITCH on BURST Command Number
         CASE 77:
            IF Cmd77Rsp.Addr = devIndex[1].addr
               THEN Test Result is Fail                                    (7510)
            END IF

         CASE 48:

         DEFAULT:
            Test Result is FAIL                                            (7511)
         END CASE
      END FOR
```

Reconnect the device

```
      PRINT "Reconnect the sub-device to the I/O System"
      DO
         IF elapsed time > 30 Seconds
            THEN TEST result is FAIL                                       (7512)
         END IF
         SEND Command 48
      WHILE (Cmd48Rsp."Sub-Device List Changed" NOT SET)
```

We should receive burst message from the device now.

```
      SET DeviceNotFound = TRUE
      FOR (120 seconds)
         CAPTURE BURST
         SWITCH on BURST Command Number
         CASE 77:
            IF Cmd77Rsp.Addr = devIndex[1].addr
               THEN SET DeviceFound = FALSE
            END IF

         CASE 48:

         DEFAULT:
            Test Result is FAIL                                            (7513)
         END CASE
      END FOR
      IF DeviceNotFound
         THEN Test Result is FAIL                                          (7514)
      END IF
```

Turn OFF Burst Mode in I/O System

```
      FOR i = 0 to maxMessage
         SEND Command 109 with
            Burst Mode Control Disabled
            Burst Message i
      END FOR


      END TEST
```

7.48 CAL107 (Reserved)

Command 107, Write Burst Device Variables is verified in the Data Link Layer Tests. As a result, it is not included in this Test Specification. All Field Devices must pass the Data Link Layer Tests before undertaking the Common Practice Tests.

7.49 CAL108 (Reserved)

Command 108, Write Burst Mode Command Number is verified in the Data Link Layer Tests. As a result, it is not included in this Test Specification. All Field Devices must pass the Data Link Layer Tests before undertaking the Common Practice Tests.

7.50 CAL109 (Reserved)

Command 109, Burst Mode Control is verified in the Data Link Layer Tests. As a result, it is not included in this Test Specification. All Field Devices must pass the Data Link Layer Tests before undertaking the Common Practice Tests.

7.51 CAL110 (Reserved)

Implementation of Common Practice Command 110 is not recommended. As a result, Field Device implementations are not tested.

7.52 CAL115 Event Notification

Event notification requires, and is built upon, Burst Mode operation. If Burst Mode is supported, then Event Notification should be supported. For wired devices, Event Notification must be disarmed while the device is not in Burst Mode.

The HART Protocol offers two distinct methods to display events: the device status and the Common Practice Command 48. Event Notification publishes changes in the device's status, independently from data publishing supported in other Burst Mode commands.

The following commands control Event Notification operation:

- **Command 115** is used to determine the configuration of the Event Notification.
- **Command 116** selects the bits that can trigger an Event Notification.
- **Command 117** controls the timing of Event Notifications.
- **Command 118** is used to enable or disable Event Notification
- **Command 119** is used to acknowledge the Event Notification

The device must retain Event Notification Settings through a Device Reset, Self Test or the power being removed and reapplied.

Case A and B are for all products. Case C is for profiles that support sub-devices. Case A tests the fundamental requirements of event notification. Case B tests the queueing of events for multiple events. Case C focuses on products that support sub-devices.

References:

Specification	**Rev.**	**Sections**
Common Practice Command Specification	9.0	7.82, 7.83, 7.84, 7.85, 7.86

Test Case A: Basic Tests for all HART devices

```
CALL IdentifyDevice
```

Event Notification requires implementation of burst mode. WirelessHART devices must implement Command 115 and event notification.

```
IF (DEVICE PROFILE & 0x80)
      SEND Command 115 with no data bytes
      IF (RESPONSE_CODE != "Too Few Data Bytes Received")
            THEN Test Result is FAIL                            (7530)
      END IF
END IF
CALL VerifyAssociatedCommands(115, 108, 109, 116, 117, 118, 119)

CALL VerifyNotWriteProtected()
```

If Token Passing network, then make sure burst mode is off.

```
IF TOKEN PASSING THEN
      SEND Command 109 with   Burst Mode Control Code Disabled
                              Burst Message 0
      CALL VerifyResponseAndByteCount("SUCCESS", 4)
END IF
```

Event Notification Summary

```
SEND Command 115 with   Event 0
IF ((RESPONSE_CODE != "SUCCESS") OR (BYTE_COUNT < 22))
      THEN TEST Result is FAIL                                          (7531)
END IF

IF Command115.eventNumber != 0
      THEN Test Result is FAIL                                          (7532)
END IF

IF Command115.eventStatus != 0
      THEN Test Result is FAIL                                          (7533)
END IF

IF Command115.eventNotificationControl != 0
      THEN Test Result is FAIL                                          (7534)
END IF

IF Command115.firstEventTime != 0xFFFFFFFF
      THEN Test Result is FAIL                                          (7535)
END IF
```

Send Commands with bad event number.

```
SEND Command 115 with   Event Number = Number of events supported
CALL VerifyResponseAndByteCount("Invalid Selection", 2)

SEND Command 116 with   Event Number = Number of events supported
CALL VerifyResponseAndByteCount("Invalid Selection", 2)

SEND Command 117 with   Event Number = Number of events supported
CALL VerifyResponseAndByteCount("Invalid Event Number", 2)

SEND Command 118 with   Event Number = Number of events supported
CALL VerifyResponseAndByteCount("Invalid Selection ", 2)

SEND Command 119 with   Event Number = Number of events supported
CALL VerifyResponseAndByteCount("Invalid Selection ", 2)
```

Send Command 116 with too few data bytes.

```
SEND Command 116 with no data bytes
CALL VerifyResponseAndByteCount("Too Few Data Bytes Received", 2)
```

Configuration of Event Notification

```
SEND Command 116 with   Event 0
                        Bit Mask Configuration Change Flag
CALL VerifyResponseAndByteCount("SUCCESS", 29)
```

Send Command 116 with extra data bytes.

```
SEND Command 116 with   Event 0
                        Bit Mask Configuration Change Flag
                        additional data byte
CALL VerifyResponseAndByteCount("SUCCESS", 29)
```

Send Command 115 with too few data bytes.

```
SEND Command 115 with no data bytes
```

```
CALL VerifyResponseAndByteCount("Too Few Data Bytes Received", 2)
```

Send Command 115 with extra data bytes.

```
SEND Command 115 with   event number 0
                        additional data byte
CALL VerifyResponseAndByteCount("SUCCESS", 47)
```

Configure Retry Maximum Update and De-bounce Interval

Update exceeding limit for physical layer

```
SEND Command 117 with   Event 0
                        Event Notification Retry Time = 3601s
                        Maximum Update Time = 60s
                        Debounce Interval = 1.00s
CALL VerifyResponseAndByteCount("Update Period or Debounce Interval
      Adjusted", 15)
IF (Event Notification Retry Time != 3600 )
      THEN TEST Result is FAIL                                       (7540)
END IF
```

Retry time cannot be larger than maximum update interval

```
SEND Command 117 with   Event 0
                        Event Notification Retry Time = 60s
                        Maximum Update Time = 32s
CALL VerifyResponseAndByteCount("Update Period or Debounce Interval
      Adjusted", 15)
IF (Event Notification Retry Time != Maximum Update Time )
      THEN TEST Result is FAIL                                       (7541)
END IF
```

Update below limit for physical layer

```
SEND Command 117 with   Event 0
                        Maximum Update Time = 0.050s
CALL VerifyResponseAndByteCount("Update Period or Debounce Interval
      Adjusted", 15)
```

Try an odd value and see if the device corrects it

```
SEND Command 117 with   Event 0
                        Maximum Update Time = 1.5s
CALL VerifyResponseAndByteCount("Update Period or Debounce Interval
      Adjusted", 15)
IF (Maximum Update Time != [1.0 or 2.0] )
      THEN TEST Result is FAIL                                       (7542)
END IF
```

Configure update rates to reasonable values.

```
SEND Command 117 with   Event 0
                        Event Notification Retry Time = 4.00s
                        Maximum Update Time = 60.0s
                        Debounce Interval = 2.000s
CALL VerifyResponseAndByteCount("SUCCESS", 15)
```

Token Passing networks must be in burst mode for event notification to function.

```
IF TOKEN PASSING THEN
```

Test that Command 118 cannot be written without burst mode

```
      SEND Command 118 with   Event Number 0
                              Event Notification Control Code Enabled
      CALL VerifyResponseAndByteCount("Access Restricted", 2)
```

Activate Burst Mode

```
      SEND Command 103 with   BMessage = 2 and
                              both Update Periods set to 2 sec.
```

```
            CALL VerifyResponseAndByteCount(0, 11)
            SEND Command 109 with   Burst Mode Control Code Enabled
                                    Burst Message 0
            CALL VerifyResponseAndByteCount("SUCCESS", 4)
      END IF
```

Use an array of to create a history of multiple command 119 responses.

```
      cmd119[] = null
```

Use Command 9 to get a starting timestamp.

```
      SEND Command 9 with dVar 246 to get timestamp
      startTime = command9.timeStamp
```

Activate Event Notification

```
      SEND Command 118 with   Event Number 0
                              Event Notification Control Code Enabled
```

Before changing the tag, we will save it to restore at the end of the testcase.

```
      SEND Command 20
      CALL VerifyResponseAndByteCount("SUCCESS", 34)
      startTag = command20.longTag
```

Write Long Tag to modify the device's configuration

```
      SEND Command 22 with tag HART115a
```

Use Command 9 to get an ending timestamp.

```
      SEND Command 9 with dVar 246 to get timestamp
      endTime = command9.timeStamp
```

Wait 8 seconds for event notification to be sent to master via command 119 response.

```
      Wait 8 seconds for command 119 response
      IF no Command 119 response received
            THEN Test Result Is FAIL                                   (7545)
      END IF
      Cmd119[0] = command119 response
```

Using the event's timestamp, verify that the event capture occurred during this test.

```
      IF (command119.timestamp > endTime ) OR
         (command119.timestamp < startTime) THEN
            Test Result is FAIL                                        (7546)
      END IF
```

Verify that all Command 119 responses match.

```
      SET n=1
      FOR 20 seconds
            IF (Command 119 response received) THEN
                  Cmd119[n] = Command119 response
                  INCREMENT n
            END IF
      END FOR
      IF (n < 4)
            THEN Test Result Is FAIL                                   (7547)
      END IF

      IF Cmd119[0] != Cmd119[1] != Cmd119[2]!= cmd119[3]
            THEN Test Result Is FAIL                                   (7548)
      END IF
```

Compare device status contained in Command 119 with device status retuned with the command response. The Configuration Changed event must still be latched.

```
      IF DEVICE_STATUS != "Configuration Changed"
         AND Cmd119[0].status != "Configuration Changed"
            THEN Test Result Is FAIL                                   (7549)
      END IF
```

Get Configuration Changed Counter using command 0 for comparison with command 119 response.

```
      SEND Command 0

      IF cmd119[0].configurationChangedCounter
         != command0.configurationChangedCounter
            THEN Test Result Is FAIL                                    (7550)
      END IF
```

See if we can acknowledge and event with data that does not match

```
      SEND Command 119 with   cmd119[0].eventNumber
                              (cmd119[0].configurationChangedCounter-2)
                              cmd119[0].timestamp
                              cmd119[0].cmd48DataBits
                              cmd119[0].deviceStatus

      IF RESPONSE_CODE != "SUCCESS"
            THEN Test result is FAIL                                    (7551)
      END IF
```

Use command 119 to acknowledge the Configuration Changed event.

```
      SEND Command 119 with   cmd119[0].eventNumber
                              cmd119[0].configurationChangedCounter
                              cmd119[0].timestamp
                              cmd119[0].cmd48DataBits
                              cmd119[0].deviceStatus

      IF RESPONSE_CODE != "SUCCESS"
            THEN Test result is FAIL                                    (7552)
      END IF
```

Wait for another event to indicate the previous event has been acknowledged.

```
      Wait 5 seconds for command 119 response
      IF any Command 119 response received
            THEN Test Result Is FAIL                                    (7553)
      END IF
```

While the event transitions are volatile, the configuration of the event must be non-volatile. Cycle power to see if Event Notification is still active.

```
      PRINT: "Please power down devices, including removal of any batteries (if
         applicable)."
```

Verify that the device is powered down.

```
      DO
         SEND Command 0
      WHILE (COMMUNICATIONS_ERROR != "No Response")
```

Prompt the user to power up the devices.

```
      PRINT: "Please re-apply power to devices, including connecting any
         batteries (if applicable)."
```

Wait for the device to power up.

```
DO
   SEND Command 0
WHILE (COMMUNICATIONS_ERROR == "No Response")
```

Use Command 9 to get another timestamp.

```
SEND Command 9 with dVar 246 to get timestamp
```

Wait 2x MaxUpdate times for event notification to be sent to master via command 119 response.

```
Wait 120 seconds for command 119 response
IF no Command 119 response received
      THEN Test Result Is FAIL                                      (7555)
END IF
IF (command119.timestamp < command9.timestamp)
      THEN Test Result is FAIL                                      (7556)
END IF
```

Use Command 9 to get another timestamp.

```
SEND Command 9 with dVar 246 to get timestamp
```

Clear Configuration Change flag

```
SEND Command 38 with cmd119[0].configurationChangedCounter
```

Wait for indication that the Configuration Changed event has been cleared.

```
Wait 8 seconds for command 119 response
IF no command 119 response received
      THEN Test Result Is FAIL                                      (7557)
END IF
```

Verify different event

```
IF ( cmd119[0].timestamp < command9.timeStamp)
      THEN Test Result Is FAIL                                      (7558)
END IF
```

Acknowledge clear event

```
SEND Command 119 with   cmd119.eventNumber
                        cmd119.configurationChangedCounter
                        cmd119.timestamp
                        cmd119.cmd48DataBits
                        cmd119.deviceStatus
```

Disable Event Notification

```
SEND Command 118 with   Event Number 0
                        Event Notification Control Code Disabled
CALL VerifyResponseAndByteCount("SUCCESS", 4)

IF TOKEN PASSING THEN
      SEND Command 109 with   Burst Mode Control Code Disabled
                              Burst Message 0
      CALL VerifyResponseAndByteCount("SUCCESS", 4)
END IF
```

Restore the tag.

```
SEND Command 20 with startTag
CALL VerifyResponseAndByteCount("SUCCESS", 34)

SEND Command 0
SEND Command 38 with configChangeCount
CALL VerifyResponseAndByteCount("SUCCESS", 4)

END TEST CASE
```

Test Case B: Queuing of multiple events.

This test case uses configuration changes to trigger the event notification. Multiple event transitions are queued by repeatedly changing the tag and resetting the Configuration Changed flag. Each transition (high➔low or low➔high) is another transition that must be queued by the Event Notification service in the device. Devices must queue at least 3 transitions.

```
      CALL IdentifyDevice
```

Event Notification requires implementation of burst mode.

```
      CALL VerifyAssociatedCommands(115, 108, 109, 116, 117, 118, 119)
      CALL VerifyNotWriteProtected()
```

Configuration of Event Notification

```
      SEND Command 116 with   Event 0
                              Bit Mask Configuration Changed Flag
      CALL VerifyResponseAndByteCount("SUCCESS", 29)
```

Configure Retry Maximum Update and De-bounce Interval

```
      SEND Command 117 with   Event Number = 0
                              Event Notification Retry Time = 4s
                              Maximum Update Time = 60s
                              Debounce Interval = 2s
      CALL VerifyResponseAndByteCount("SUCCESS", 15)
```

Token Passing networks must be in burst mode for event notification to function.

```
      IF TOKEN PASSING THEN
            SEND Command 103 with   BMessage = 2 and
                                    both Update Periods set to 2 sec.
            CALL VerifyResponseAndByteCount(0, 11)

            SEND Command 109 with   Burst Mode Control Code Enabled
                                    Burst Message 0
            CALL VerifyResponseAndByteCount(0, 4)
      END IF
```

Activate Event Notification

```
      SEND Command 118 with   Event Number 0
                              Event Notification Control Code Enabled
      CALL VerifyResponseAndByteCount("SUCCESS", 4)
```

Before changing the tag, we will save it to restore at the end of the testcase.

```
      SEND Command 20
      CALL VerifyResponseAndByteCount("SUCCESS", 34)
      startTag = command20.longTag
```

Use an array to create a history of multiple command 119 responses and change counter values.

```
      cmd119[] = null
      chgCntr[] = null
```

Write Long Tag to modify the device's configuration - This queues up event transition #1

```
      SEND Command 22 with HART115b
      CALL VerifyResponseAndByteCount("SUCCESS", 34)
```

Wait for indication that the Configuration Changed event has been set.

```
      Wait 5 seconds for Cmd119Rsp

      IF no Cmd119Rsp received
            THEN Test Result Is FAIL                                        (7560)
      END IF

      SET cmd119[0] = command119 response
```

Get Configuration Change Counter using command 0 for verification.

```
      SEND Command 0
      SET chgCntr[0] = command0.configurationChangeCounter
      IF configChangeCount != cmd119.configurationChangedCounter
            THEN Test result is FAIL                                       (7561)
      END IF
```

Queue up event transitions 2-5

```
      FOR n = 0-1
```

Clear Configuration Change flag - This queues up an event transition (2*n)

```
            SEND Command 38 with configChangeCount
            CALL VerifyResponseAndByteCount(0, 4)
            VerifyEventUnchanged(cmd119[0])
```

Save a copy of the Configuration Change Counter for verification.

```
            SEND Command 0
            chgCntr[2*n+1] = command0.configurationChangeCounter
```

Write Long Tag to modify the device's configuration -
This queues up another event transition (2*n+1)

```
            SEND Command 22 with HART115b1
            CALL VerifyResponseAndByteCount("SUCCESS", 34)
            VerifyEventUnchanged(cmd119[0])
```

Save a copy of the Configuration Change Counter for verification.

```
            SEND Command 0
            chgCntr[2*n+2] = command0.configurationChangeCounter
```

Write Long Tag again. Since Configuration Changed flag set this does not generate another event.

```
            SEND Command 22 with HART115b2
            CALL VerifyResponseAndByteCount("SUCCESS", 34)
            VerifyEventUnchanged(cmd119[0])
      END FOR
```

Clear Configuration Changed flag - This queues up an event transition #6

```
      SEND Command 38 with configChangeCount
      CALL VerifyResponseAndByteCount(0, 4)
      VerifyEventUnchanged(cmd119[0])

      SEND Command 0
      SET chgCntr[5] = command0.configurationChangeCounter
```

Acknowledge, thus dequeuing the first event

```
      SEND Command 119 with cmd119[0]
      Wait 5 seconds for Cmd119Rsp

      IF no Cmd119Rsp received
            THEN Test Result Is FAIL                                       (7569)
      END IF

      IF Cmd119Rsp == cmd119[0]
            THEN Test result is FAIL                                       (7562)
      END IF
```

Disable Event Notification

```
      SEND Command 118 with   Event Number 0
                              Event Notification Control Code Disabled
      CALL VerifyResponseAndByteCount("SUCCESS", 4)
```

Now de-queue the events

```
      FOR n = 1-5
```

```
            SEND Command 119 (truncated) with Event Number 0 only
            SET cmd119[n] = Cmd119Rsp
            SEND Command 119 with Cmd119Rsp
      END FOR
```

Validate the events that were queued

```
      SET NTran = 1
      SET GoodCnts[] = {0,0,1,2,3,4}
      SET CfgChgBitSeq[] = {SET, RESET, SET, RESET, SET, RESET, SET}

      FOR n = 0-5

            IF cmd119[n].timestamp = 0xFFFFFFFF THEN
                  IF (cmd119[n].ConfigChanged is SET)
                        THEN Test result is FAIL                           (7563)
                  END IF

                  IF (cmd119[n].configurationChangedCounter != chgCntr[0]+4 )
                        THEN Test result is FAIL                           (7564)
                  END IF
            ELSE
                  INCREMENT nTran

                  IF (cmd119[n].ConfigChanged != CfgChgBitSeq[n])
                        THEN Test result is FAIL                           (7565)
                  END IF

                  IF (cmd119[n].configurationChangedCounter != chgCntr[n])
                        THEN Test result is FAIL                           (7566)
                  END IF

                  IF (cmd119[n].configurationChangedCounter !=
                    GoodCnts[n]+chgCntr[0])
                        THEN Test result is FAIL                           (7567)
                  END IF

            END IF
      END FOR

      IF nTran < 3
            THEN Test result is FAIL                                       (7568)
      ELSE IF nTran < 6 THEN
            Print "DUT supports nTran Event Transitions"
      ELSE
            Print "DUT supports at least 6 Event Transitions"
      END IF
```

Cleanup and finish the test case. First turn off burst mode if necessary

```
      IF TOKEN PASSING THEN
            SEND Command 109 with   Burst Mode Control Code Disabled
                                    Burst Message 0
            CALL VerifyResponseAndByteCount("SUCCESS", 4)
      END IF
```

Restore the tag.

```
      SEND Command 20 with startTag
      CALL VerifyResponseAndByteCount("SUCCESS", 34)

      SEND Command 0
      SEND Command 38 with configChangeCount
      CALL VerifyResponseAndByteCount("SUCCESS", 4)
      END TestCase
```

Test Case C: Events and sub-devices

I/O systems must monitor sub-devices for the specified events and publish the Event Notification(s) on their behalf. This test sets up two different events for the subdevice (Configuration Changed and Cold Start) and verifies the events are published correctly.

```
      CALL IdentifyDevice
      IF (UNIV_REVISION < 7) THEN
            ABORT "Device is not HART 7 or later"
      END IF
```

Issue command 0. Examine the "Protocol Bridge Device" bit of the Flags byte

```
      SEND Command 0
      IF (FLAGS != "Protocol Bridge Device")
            THEN ABORT "DEVICE IS NOT A Bridge device"
      END IF

      CALL VerifyAssociatedCommands(74, 75, 77, 84, 85, 86, 87, 88, 94)
      CALL VerifyAssociatedCommands(115, 108, 109, 101, 102, 116, 117, 118, 119)
```

Are any sub-devices present?

```
      SEND Command 74
      IF (Number of devices detected < 2)
         THEN Test Result Is FAIL                                       (7570)
      END IF
```

Get the sub-device information.

```
      SEND Command 84 with SubDeviceIndex = 1
      devIndex[1]= Command 84 reponse
```

Before changing the tag, we will save it to restore at the end of the testcase.

```
      SEND Command 77 using devIndex[1] with Command 13
      CALL VerifyResponseAndByteCount("SUCCESS", 35)
      startTag = command13.Tag
```

Token Passing networks must be in burst mode for event notification to function.

```
      IF TOKEN PASSING THEN
            SEND Command 103 with   BMessage = 2 and
                                    both Update Periods set to 2 sec.
            CALL VerifyResponseAndByteCount(0, 11)

            SEND Command 109 with   Burst Mode Control Code Enabled
                                    Burst Message 0
            CALL VerifyResponseAndByteCount(0, 4)
      END IF
```

Basics are done, now map the Events to the sub-device using Command 102. Hi order bit in "Burst Message" field must be set set to indicate this is an event we are mapping

```
      SEND Command 102 with   SubDeviceIndex = 1,
                              BurstMessage = 0 || 0x80 and event bit = 1
      VerifyResponseAndByteCount ("SUCCESS",5)

      SEND Command 102 with   SubDeviceIndex = 1,
                              BurstMessage = 1 || 0x80 and event bit = 1
      VerifyResponseAndByteCount ("SUCCESS",5)
```

Configure the events. First, setup Event 0 - Configuration Changed using Cmds 116, 117, and 118

```
      SEND Command 116 with   Event 0
                              Device Status = Configuration Changed
                              Event Mask = 25Bytes of 0xFF
```

```
    CALL VerifyResponseAndByteCount("SUCCESS", 29)

    SEND Command 117 with   Event Number = 0
                            Event Notification Retry Time = 4s
                            Maximum Update Time = 60s
                            Debounce Interval = 2s
    CALL VerifyResponseAndByteCount("SUCCESS", 15)

    SEND Command 118 with   Event Number 0
                            Event Notification Control Code Enabled
    CALL VerifyResponseAndByteCount("SUCCESS", 4)
```

Setup Event 1 - Cold Start

```
    SEND Command 116 with   Event 0
                            Device Status = Cold Start Flag
                            Event Mask = 25Bytes of 0xFF
    CALL VerifyResponseAndByteCount("SUCCESS", 29)

    SEND Command 117 with   Event Number = 0
                            Event Notification Retry Time = 4s
                            Maximum Update Time = 60s
                            Debounce Interval = 2s
    CALL VerifyResponseAndByteCount("SUCCESS", 15)

    SEND Command 118 with   Event Number 0
                            Event Notification Control Code Enabled
    CALL VerifyResponseAndByteCount("SUCCESS", 4)
```

Now we will get an event publishing by modifying the sub-device's tag. Once that is publishing we will reset (cycle power) on the sub-device. Then we should be receiving the Configuration Cahanged and Cold Start events.

Use an array to store the history of the multiple command 119 responses

```
    cmd119[] = null
```

Write Tag to the sub-device to modify the sub-device's configuration to start Event 0 publishing

```
    SEND Command 77 using devIndex[1] with Command 18 and tag=HART115C
    VerifyResponseAndByteCount ("SUCCESS",35)
```

Wait for Event 0

```
    Wait 5 seconds for command 119 response

    IF (no command 119 response received)
       THEN Test Result Is FAIL                                          (7571)
    END IF

    IF (Cmd119Rsp.EventNumber != 0)
       THEN Test Result Is FAIL                                          (7572)
    END IF
```

Now let's trip Event 1 (Cold Start)

```
    PRINT: "Please power down the SubDevice, including removal of any batteries
       (if applicable)."
```

Verify that the device is powered down.

```
    DO
       SEND Command 77 using devIndex[1]with Command 0
    WHILE (RESPONSE_CODE != "DR_DEAD")
```

Prompt the user to power up the devices.

```
PRINT: "Please re-apply power to devices, including connecting any
   batteries (if applicable)."
```

Wait for the device to power up.

```
DO
   SEND Command 77 using devIndex[1] with SEND Command 0
WHILE (RESPONSE_CODE != "SUCCESS")
```

Now we should be getting both events.

```
SET EventHistory = 0

FOR 5 seconds
      IF (Command 119 response received) THEN
            SWICTH on Cmd119Rsp.EventNumber
                  CASE 0:
                        SET EventHistory = EventHistory || 1
                        SET cmd119[0] = Cmd119Rsp
                  CASE 1:
                        SET EventHistory = EventHistory || 2
                        cmd119[1] = command119 response
            END SWITCH
      END IF
END FOR

IF (EventHistory != 3)
      THEN Test Result Is FAIL                                        (7575)
END IF

IF ( cmd119[0].timestamp == cmd119[1].timestamp )
   THEN Test Result Is FAIL                                           (7576)
END IF
```

Actually two transitions must be queued for event 1 (power fail is a one-shot). Acknowledge the event notifications

```
SEND Command 119 with cmd119[0] information
SEND Command 119 with cmd119[1] information
```

Wait for the queued Event 1

```
Wait 5 seconds for command 119 response

IF command 119 response received
   THEN Test Result Is FAIL                                           (7577)
END IF

IF (Cmd119Rsp.EventNumber != 1)
   THEN Test Result Is FAIL                                           (7578)
END IF
```

Clear the event

```
SEND Command 119 with Cmd119Rsp
```

Clean up and finish the test case. Disable Event Notification and turn off burst mode if necessary

```
SEND Command 118 with   Event Number 0
                        Event Notification Control Code Disabled
CALL VerifyResponseAndByteCount("SUCCESS", 4)

SEND Command 118 with   Event Number 1
                        Event Notification Control Code Disabled
CALL VerifyResponseAndByteCount("SUCCESS", 4)

IF TOKEN PASSING THEN
```

```
            SEND Command 109 with   Burst Mode Control Code Disabled
                                    Burst Message 0
            CALL VerifyResponseAndByteCount("SUCCESS", 4)
      END IF
```

Restore the tag.

```
      SEND Command 77 using devIndex[1] with Command 18 and tag=startTag
      VerifyResponseAndByteCount ("SUCCESS",35)

      SEND Command 77 using devIndex[1] with Command 38 and no data
      VerifyResponseAndByteCount ("SUCCESS", [12 or 14] )

      END TestCase
```

VerifyEventUnchanged(Cmd119Rsp)

Waits for an Even Notification and verifies it matches our expectation.

```
      PROCEDURE VerifyEventUnchanged(Cmd119Rsp)

      Wait 5 seconds for command 119 response
      IF no command 119 response received
            THEN Test Result Is FAIL                                       (7589)
      END IF
      IF cmd119 response != Cmd119Rsp
            THEN Test result is FAIL                                       (7588)
      END IF

      PROCEDURE END
```

7.53 CAL512 Country Code

Verifies that the DUT responds properly to Command 512 and 513. Checks Country Code and SI Units Control Code. The following conditions are evaluated.

- The DUT must support Command 512 if Command 513 is implemented.
- Check for valid country code from ISO-3166.
- Verify byte counts.
- Change country code to value different from one shipped.

Commands tested include:

- Command 512, Read Country Code
- Command 513, Write Country Code

The test does not verify that each ISO-3166 Country Code is valid.

References:

Specification	**Rev.**	**Sections**
Common Practice Command Specification	9.0	7.87, 7.88

Test Procedure

```
CALL IdentifyDevice
CALL CheckCommandImplemented(512)
CALL CheckCommandImplemented(513)
CALL VerifyNotWriteProtected()
```

Read Country Code.

```
SEND Command 512
CALL VerifyResponseAndByteCount(0, 4)
```

Verify country code is not 00

```
IF (Country Code == 00)
      THEN Test result is FAIL                                     (7400)
END IF

IF (SI UNIT CODE > 1)
      THEN Test result is FAIL                                     (7401)
END IF
```

Send Command 513 with too few data bytes.

```
SEND Command 513 with too few data bytes
CALL VerifyResponseAndByteCount(5, 2)
```

Send Command 513 with extra data bytes.

```
SEND Command 513 with additional data bytes
CALL VerifyResponseAndByteCount(0, 4)
```

Send Command 512 with extra data bytes.

```
SEND Command 512 with additional data bytes
CALL VerifyResponseAndByteCount(0, 4)
```

Send Command 513 with country code not in ISO3166

```
SEND Command 513 with non-ISO3166 country code
CALL VerifyResponseAndByteCount(0, 4)
```

Send Command 513 with random country code

```
SEND Command 513 with different country code
IF (RESPONSE_CODE != "Success")
      THEN Test result is FAIL                                        (7402)
END IF

END TEST
```

ANNEX A. REUSABLE TEST PROCEDURE DEFINITIONS

The procedures in this appendix are used in two or more of the CAL test definitions. They are presented here as reusable procedures to remove redundancy in the Test Body.

A.1 CheckCommandImplemented (Cmd)

The following procedure is used to verify that the DUT supports the indicated command

```
PROCEDURE CheckCommandImplemented (Cmd)
SEND Cmd with no data bytes
IF (RESPONSE_CODE = "Command not Implemented")
      THEN Abort Test                                                (5000)
END IF
PROCEDURE END
```

A.2 CheckForRecommendedCommand (Cmd)

The following procedure is used to verify that the DUT supports the indicated command

```
PROCEDURE CheckForRecommendedCommand (Cmd)
SEND Cmd with no data bytes
IF (RESPONSE_CODE = "Command not Implemented")
      PRINT "Warning, Implementation of Command Cmd is strongly
            recommended.  This command is implemented by most
            Field Devices and widely used in Host Applications."
      Abort Test                                                     (5002)
END IF
PROCEDURE END
```

A.3 CompareAnalogChannelValue(aChan, aValue, failurepoint)

This procedure uses Command 60 to verify that the Analog Channel Value is as expected.

```
PROCEDURE CompareAnalogChannelValue(aChan, aValue, FAILUREPOINT)
DO
      SEND Command 60 (with aChan) to read mrVal
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Update Failure") AND (BYTE_COUNT != 12) )
            THEN Test result is FAIL                         (FAILUREPOINT)
      END IF
WHILE (RESPONSE_CODE == "Update Failure")
CALL TestValidFrame()
CALL VerifyResponseAndByteCount(0, 12)

IF (aValue != mrVal) THEN
      Test result is FAIL                                  (FAILUREPOINT+1)
END IF
PROCEDURE END
```

A.4 FindNextAnalogChannel(aChan)

Find a supported Analog Channel. While a device can support more than 24 analog channels most of that support would be made using device specific commands. As a result, we stop looking at 24.

```
PROCEDURE FindNextAnalogChannel(aChan)
DO
      INCREMENT aChan
      IF (aChan > 24) THEN
            RETURN "No More Analog Channels"
      END IF
      SEND Command 60 with one byte = aChan
      CALL TestValidFrame
      IF ( (RESPONSE_CODE == "Invalid Selection") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                                 (5130)
      END IF
WHILE (RESPONSE_CODE == "Invalid Selection")

IF ( (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Update Failure") )
      THEN Test result is FAIL                                       (5131)
ELSE IF (BYTE_COUNT != 12)
      THEN Test result is FAIL                                       (5132)
ELSE
      RETURN "Analog Channel Found"
END IF

PROCEDURE END
```

A.5 FindNextDeviceVariable(dVar)

Find a supported Device Variable using Command 33

```
PROCEDURE FindNextDeviceVariable(dVar)
DO
      SET dVarFound = FALSE;
      INCREMENT dVar
      IF (dVar > 249) THEN
            RETURN "No More Device Variables"
      END IF
      SEND Command 33 with one byte = dVar
      CALL TestValidFrame

      IF ( (RESPONSE_CODE == "Invalid Selection")
            IF (BYTE_COUNT != 2) )
                  THEN Test result is FAIL                           (5140)
            END IF
      ELSE IF ( (RESPONSE_CODE != "Update Failure")
         AND (RESPONSE_CODE != 0 ))
            THEN Test result is FAIL                                 (5141)
      ELSE IF (BYTE_COUNT != 8)
            THEN Test result is FAIL                                 (5142)
```

If we get a NaN response make sure all the other fields are set correctly

```
      ELSE IF (dVar.Value == "7F A0 00 00"(NaN) THEN
            IF (dVar.Units != 250)
                  THEN Test result is FAIL                           (5143)
            END IF
```

Response is "Success" or "Update Failure", not a NaN, and the right Byte Count. I think we have it!

```
      ELSE
            SET dVarFound = TRUE:
      END IF
WHILE (!dVarFound)

IF (UNIV_REVISION >= 6)
      SEND Command 0 to read maxDeviceVars
      IF (dVar > maxDeviceVars)
            THEN Test result is FAIL                                 (5146)
      ELSE
            RETURN "Device Variable Found"
      END IF
END IF
PROCEDURE END
```

A.6 IdentifyDevice ( )

Identify the device, check its revision, record the number of preambles it desires for later requests, and note its unique identifier for later requests.

```
PROCEDURE IdentifyDevice()
Set NUMBER_REQUEST_PREAMBLES to 15
pollAddress = 0, deviceFound = FALSE

While (( pollAddress < 63 ) AND (!deviceFound))
      SEND short frame Command 0 using POLL_ADDRESS = pollAddress
      IF  ( COMMUNICATIONS_ERROR == "No Response" )
            THEN increment pollAddress
      ELSE  IF ( COMMUNICATIONS_ERROR OR (RESPONSE_CODE != 0) )
            THEN Test result is FAIL                                (5100)
      ELSE
            deviceFound = TRUE
      END IF
END WHILE

IF (!deviceFound)
      THEN Test result is FAIL                                      (5101)
END IF

Set NUMBER_REQUEST_PREAMBLES. UNIV_COMMAND_REVISION, POLL_ADDRESS
IF UNIV__REVISION < 5
      THEN Abort Test (i.e., test is not applicable to this device)  (5001)
END IF
IF UNIV_REVISION == 5 AND pollAddress > 15
      THEN FAIL                                                     (5002)
END IF
IF UNIV__REVISION > 7
      THEN Abort Test (i.e., test is not applicable to this device)  (5003)
END IF
PROCEDURE END
```

A.7 ReadPV()

This procedure uses command 1 to read PV

```
PROCEDURE ReadPV()
DO
      SEND Command 1 to read PV
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Update Failure") AND (BYTE_COUNT != 7) )
            THEN Test result is FAIL                                (5135)
      END IF
WHILE (RESPONSE_CODE == "Update Failure")
VerifyResponseAndByteCount(0,7)

PROCEDURE END
```

A.8 TestValidFrame ( )

This procedure checks that the DUT replies with the correct information from the command. It compares framing information in a request command and a reply command.

```
PROCEDURE TestValidFrame()
IF reply address does not agree with manufacturer id masked with 0x3f,
   manufacturer device type byte and the three byte ID number
      THEN Test Result is FAIL                                       (5115)
END IF
IF reply Command  != request Command
      THEN Test Result is FAIL                                       (5116)
END IF
PROCEDURE  END
```

A.9 VerifyLoopCurrent(v, failurepoint)

This procedure uses command 2 to verify that the primary variable value is equal to the loop current value. The procedure loops on the command 2 until the DUT returns a valid response.

```
PROCEDURE VerifyLoopCurrent(v, FAILUREPOINT)
DO
      SEND Command 2 to read loop current
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Update Failure") AND (BYTE_COUNT != 10) )
            THEN Test result is FAIL                         (FAILUREPOINT)
      END IF
WHILE (RESPONSE_CODE == "Update Failure")
VerifyResponseAndByteCount(0,10)

IF loop current != v THEN
      Test result is FAIL                                  (FAILUREPOINT+1)
END IF
PROCEDURE END
```

A.10 VerifyNotWriteProtected()

The following procedure verifies that the DUT is not in write protect mode.

```
PROCEDURE VerifyNotWriteProtected()
DO
      SEND Command 15
      IF (COMMUNICATIONS_ERROR)
            THEN Test result is FAIL                                 (5120)
      END IF
WHILE (RESPONSE_CODE == "Busy" )
```

Note: 251, "Not Used" is a valid response and equivalent to not "Write Protected"

```
IF (WRITE_PROTECT_CODE != [ 0, 1, or 251 ] )
      THEN Test result is FAIL                                       (5121)
END IF
IF (DUT is in "Write Protect")
      THEN Test result is FAIL                                       (5122)
END IF
PROCEDURE END
```

A.11 VerifyRangeAndPV(lrv, urv, units, PV, failurepoint)

This procedure uses command 1 to verify that the primary variable value is as expected. The procedure loops on the command 15 until the DUT returns a non-"Busy" response to the command.

```
PROCEDURE VerifyRangeAndPV(lrv, urv, units, PV, FAILUREPOINT)

DO
      SEND Command 1 to read p
      IF ( (RESPONSE_CODE == "Update Failure") AND (BYTE_COUNT != 7) )
            THEN Test result is FAIL                         (FAILUREPOINT)
      END IF
WHILE (RESPONSE_CODE = "Update Failure")

VerifyResponseAndByteCount(0,7)
IF (p != PV THEN)
      Test result is FAIL                                  (FAILUREPOINT+1)
END IF
DO
      SEND Command 15 to read l, u, un
      CALL TestValidFrame()
      IF ( (RESPONSE_CODE == "Busy") AND (BYTE_COUNT != 2) )
            THEN Test result is FAIL                       (FAILUREPOINT+2)
      END IF
WHILE (RESPONSE_CODE = "Busy")
IF ( UNIV_REVISION >= 6 )
      CALL VerifyResponseAndByteCount(0, 20)
ELSE
      THEN CALL VerifyResponseAndByteCount(0, 19)
END IF
IF lrv != l THEN
      Test result is FAIL                                  (FAILUREPOINT+3)
END IF
IF urv != u THEN
      Test result is FAIL                                  (FAILUREPOINT+4)
END IF
IF units != un THEN
      Test result is FAIL                                  (FAILUREPOINT+5)
END IF
PROCEDURE END
```

A.12 VerifyResponseAndByteCount(r, b)

Verify that the reply to a command matches the list of responses [r] and byte count b.

```
PROCEDURE VerifyResponseAndByteCount(r, b)
CALL TestValidFrame()
IF (RESPONSE_CODE != r)
      THEN Test result is FAIL                                       (5110)
END IF
IF (BYTE_COUNT != b)
      THEN Test result is FAIL                                       (5111)
END IF
PROCEDURE END
```

A.13 VerifyAssociatedCommands(Cmd[0], Cmd[1], Cmd[2]…)

Verify support of a command and any commands that must be supported as a result of that command.

```
PROCEDURE VerifyAssociatedCommands (Cmd[0], Cmd[1], Cmd[2],...,Cmd[n])
SEND Cmd[0] with no data bytes
IF (RESPONSE_CODE == "Command not Implemented")
      THEN Abort Test                                                 (5125)
ELSE
      FOR n = 1 to 10
            IF Cmd[n] != 0
                  SEND Cmd[n]
                  IF RESPONSE_CODE == "Command Not Implemented" THEN
                        Test Result is FAIL                           (5126)
                  END IF
            END IF
      END FOR
END IF
PROCEDURE END
```

ANNEX B. FAILURE POINT CROSS REFERENCE

The following table cross-references the failure point codes to the test where they can be found. The table consists of groups of ten codes (0-9) per row. An 'x' indicates the code was used in the test indicated for that row in the table.

FP Codes	Test	0	1	2	3	4	5	6	7	8	9
6300	CAL033	x	x	x	x	x	x	x	x	x	
6310											
6320	CAL034	x	x	x	x	x	x	x	x	x	x
6330	CAL034	x	x	x	x	x	x	x	x	x	x
6340	CAL034	x									
6350	CAL035	x					x				
6360	CAL035	x	x	x	x		x				
6370	CAL035	x	x	x	x		x				
6380	CAL035	x	x	x	x		x				
6390	CAL035	x	x	x	x	x	x				
6400	CAL035	x	x	x	x	x	x				
6410	CAL035		x	x	x	x	x	x	x		
6420	CAL035	x					x			x	
6430											
6440											
6450	CAL036		x	x	x	x	x	x	x	x	x
6460	CAL036	x	x	x	x	x	x	x		x	x
6470	CAL036								x	x	x
6480	CAL036	x	x	x	x	x	x	x	x	x	x
6490	CAL036	x									
6490	CAL037						x	x	x	x	x
6500	CAL037	x	x	x	x	x	x	x	x	x	x
6510	CAL037	x	x	x	x	x	x	x	x	x	x
6520	CAL037	x	x	x	x	x	x	x	x	x	x
6530	CAL037	x	x	x	x	x	x	x	x	x	x
6540	CAL037	x	x	x	x	x	x	x	x		
6550											
6560	CAL040								x	x	x
6570	CAL040	x	x	x	x	x	x	x	x	x	x
6580	CAL040	x	x	x	x	x	x	x	x		
6580	CAL043									x	x
6590	CAL042	x	x	x	x	x					
6590	CAL043						x	x	x	x	
6600	CAL045	x	x	x	x	x	x	x	x	x	x

FP Codes	Test	0	1	2	3	4	5	6	7	8	9
6610	CAL045	x	x	x	x	x	x	x	x		
6610	CAL046									x	x
6620	CAL046	x	x	x	x	x	x	x	x	x	x
6630	CAL046	x	x	x	x	x					
6630	CAL047						x	x	x	x	x
6640	CAL047	x	x	x							
6650	CAL049						x	x	x	x	x
6660	CAL049	x	x	x							
6660	CAL050						x	x	x	x	x
6670	CAL050	x	x								
6670	CAL052			x	x	x	x	x	x	x	x
6680	CAL053	x	x	x	x	x	x	x	x	x	x
6690	CAL053	x	x	x	x	x					
6690	CAL054						x	x	x		
6700	CAL055	x	x	x			x	x	x	x	x
6710	CAL055	x	x	x	x	x	x	x	x	x	x
6720	CAL055	x	x	x	x	x	x	x	x	x	x
6730	CAL055	x	x								
6730	CAL056						x	x	x	x	x
6740	CAL056	x	x	x	x	x	x				
6750	CAL060	x									
6750	CAL062						x	x	x		
6760	CAL063	x									
6770	CAL064	x	x	x	x	x	x		x	x	x
6780	CAL064	x	x	x	x	x	x	x	x	x	x
6790	CAL064	x	x	x	x	x	x	x	x	x	x
6800	CAL064	x									
6800	CAL065						x	x	x	x	x
6810	CAL065	x	x	x	x	x	x	x	x	x	x
6820	CAL065	x	x	x	x		x	x	x	x	x
6830	CAL065		x	x	x		x	x	x	x	x
6840	CAL065		x	x	x		x	x	x	x	x
6850	CAL065		x	x	x		x	x			
6860	CAL065	x	x	x	x	x		x	x	x	x

FP Codes	Test	0	1	2	3	4	5	6	7	8	9
6870	CAL065	x	x	x	x	x	x	x	x	x	x
6880	CAL065	x	x	x	x	x					
6890	CAL066	x	x	x	x	x	x	x	x	x	
6900	CAL066	x	x	x	x	x	x	x	x		
6910	CAL066	x	x	x	x						
6910	CAL067						x	x	x	x	x
6920	CAL067						x	x	x	x	x
6930	CAL067	x	x	x	x	x	x	x	x	x	x
6940	CAL067	x	x	x	x						
6940	CAL068						x	x	x	x	x
6950	CAL068	x	x	x	x	x	x	x	x	x	x
6960	CAL068	x	x	x	x	x	x	x	x	x	
6970	CAL070						x	x	x		
6980	CAL071	x	x	x	x	x	x	x	x	x	x
6990	CAL071	x	x	x	x	x	x	x	x	x	x
7000	CAL071	x	x	x	x	x	x	x			
7000	CAL072									x	x
7010	CAL073	x	x	x	x	x					
7010	CAL074						x	x	x	x	x
7020	CAL074	x	x	x	x	x	x	x	x	x	x
7030	CAL074		x				x				
7030	CAL078				x	x		x	x	x	x
7040	CAL079	x	x	x			x				
7050	CAL080	x	x	x	x	x	x	x	x	x	x
7060	CAL080	x	x	x	x	x	x	x	x		
7070	CAL080	x					x	x	x	x	x
7080	CAL074	x	x	x	x	x	x	x	x	x	
7090	CAL074	x	x	x	x		x	x	x	x	x
7100	CAL074	x	x	x	x		x	x	x	x	
7110											
7120	CAL044	x	x	x	x	x	x	x			
7130	CAL044	x	x	x	x	x	x				
7140	CAL044	x	x	x	x	x	x				
7150	CAL044	x	x	x	x	x	x				
7160											
7170	CAL001	x	x	x	x	x	x	x	x	x	x
7180	CAL001	x					x	x	x		
7190	CAL001	x	x	x	x	x	x	x	x	x	x
7200	CAL001	x	x	x	x	x	x	x	x	x	x
7210	CAL001	x	x	x	x	x	x	x	x	x	x

FP Codes	Test	0	1	2	3	4	5	6	7	8	9
7220	CAL001	x	x								x
7230	CAL001	x	x	x	x	x	x	x	x	x	x
7240	CAL001	x	x	x	x	x	x	x	x	x	x
7250	CAL001	x	x	x	x	x	x	x			
7260											
7270	CAL051	x	x	x	x	x	x	x	x	x	x
7280	CAL051	x	x	x	x	x	x	x	x	x	x
7290	CAL051	x	x	x	x	x	x	x	x	x	x
7300	CAL051	x	x	x		x	x	x	x	x	x
7310	CAL051	x	x	x	x	x	x	x	x	x	x
7320	CAL051	x	x	x	x	x	x	x	x		
7330	CAL001	x	x	x	x	x	x	x	x	x	x
7340	CAL001	x	x	x	x	x	x				x
7350	CAL001	x	x	x	x	x	x	x	x	x	x
7360	CAL001	x	x	x	x	x	x	x	x	x	x
7370	CAL001	x	x	x	x	x	x	x			x
7380	CAL001	x	x	x	x	x					
7390											
7400	CAL512	x	x	x							
7410	CAL071	x	x	x	x	x	x	x	x	x	
7420	CAL071	x	x	x	x	x	x	x	x		
7430											
7440	CAL069	x	x	x	x	x	x	x	x	x	x
7450	CAL033								x	x	x
7460	CAL033	x	x	x	x	x	x				
7470											
7480	CAL101	x	x	x	x	x	x	x	x	x	x
7490	CAL101	x					x	x	x	x	
7500	CAL101	x	x				x	x	x	x	
7510	CAL101	x	x	x	x	x					
7520											
7530	CAL115	x	x	x	x	x	x				
7540	CAL115	x	x	x			x	x	x	x	x
7550	CAL115	x	x	x	x		x	x	x	x	
7560	CAL115	x	x	x	x	x	x	x	x	x	x
7570	CAL115	x	x	x			x	x	x	x	
7580	CAL115									x	x
7590	CAL091	x	x	x	x	x	x	x	x		
7600	CAL091	x	x	x	x		x	x	x	x	x
7610	CAL091			x	x	x	x		x	x	x

FP Codes	Test	0	1	2	3	4	5	6	7	8	9
7620	CAL091	x	x	x			x	x	x	x	x
7630	CAL091	x		x	x	x	x	x	x	x	
7640	CAL091										
7650	CAL091		x	x	x	x	x	x	x		
8000	CAL000	x	x	x	x	x	x	x	x	x	x
8010	CAL000	x	x	x	x	x	x	x	x	x	x
8020	CAL000	x	x	x	x	x	x	x	x	x	x
8030	CAL000	x	x	x	x	x	x	x	x	x	x
8040	CAL000	x	x	x	x	x	x	x	x	x	x
8050	CAL000	x	x	x	x	x	x	x	x	x	x
8060	CAL000	x	x	x	x	x	x	x	x	x	x
8070	CAL000	x	x	x	x	x	x	x	x	x	x
8080	CAL000	x	x	x	x						
9000	CAL000	x	x	x	x	x	x	x	x	x	x
9010	CAL000	x	x	x	x	x	x	x	x	x	x
9020	CAL000	x	x	x	x	x	x	x	x	x	x
9030	CAL000	x	x	x	x	x	x	x	x	x	x
9040	CAL000	x	x	x	x	x	x	x	x	x	x
9050	CAL000	x	x	x	x	x	x	x	x	x	x
9060	CAL000	x	x	x	x	x	x	x	x	x	x
9070	CAL000	x	x	x	x	x	x	x	x	x	x
9080	CAL000	x	x	x	x						
10000	CAL000	x	x	x	x	x	x	x	x	x	x
10010	CAL000	x	x	x	x	x	x	x	x	x	x
10020	CAL000	x	x	x	x	x	x	x	x	x	x
10030	CAL000	x	x	x	x	x	x	x	x	x	x
10040	CAL000	x	x	x	x	x	x	x	x	x	x
10050	CAL000	x	x	x	x	x	x	x	x	x	x
10060	CAL000	x	x	x	x	x	x	x	x	x	x
10070	CAL000	x	x	x	x	x	x	x	x	x	x
10080	CAL000	x	x	x	x						
5000	Annex A	x		x							
5100	Annex A	x	x	x	x						
5110	Annex A	x	x				x	x			
5120	Annex A	x	x	x			x	x			
5130	Annex A	x	x	x			x				
5140	Annex A	x	x	x	x			x			

ANNEX C. TEST REPORT

The following Test Report must be completed for each Field Device tested.

1. Test Operator

Name Company
Title Address
Tel. No.
FAX No.
EMail

2. Certification

I hereby affirm that all data provided in this Test Report is accurate and complete.

Signature Date
Name
Title

3. Test Device Identification

Manufacturer Name: Model Name(s):
Manufacture ID Code: (Hex) Device Type Code: (Hex)
Device ID Hex

HART Protocol Revision Device Revision:
Hardware Revision Software Revision:

Revision Release Date

Physical Layers Supported Notes:
Physical Device Category

4. Test Data

Test	Result
CAL000 Checks for Common Practice Commands	□ Pass □ Fail Not Recommended Commands (List) ________________________________
CAL001 Write Protect Test	□ Pass □ Fail
CAL033 Read Device Variables	□ Pass □ Fail Not Applicable Device Variables (List) ________________________________
CAL034 Write Primary Variable Damping Value	□ Pass □ Fail □ Not Applicable
CAL035 Write Primary Variable Range Values PV Units Code Unaffected By Command 35	□ Pass □ Fail □ Not Applicable □ Pass □ Fail
CAL036 Set Primary Variable Upper Range Value	□ Pass □ Fail □ Not Applicable
CAL037 Set Primary Variable Lower Range Value	□ Pass □ Fail □ Not Applicable
CAL040 Enter/Exit Fixed Current Mode	□ Pass □ Fail □ Not Applicable
CAL041 Perform Self Test	□ Pass □ Fail □ Not Applicable
CAL042 Perform Device Reset	□ Pass □ Fail □ Not Applicable
CAL043 Set Primary Variable Zero	□ Pass □ Fail □ Not Applicable
CAL044 Write Primary Variable Units	□ Pass □ Fail □ Not Applicable Valid PV Units Codes (List) ________________________________ ________________________________
CAL045 Trim Loop Current Zero	□ Pass □ Fail □ Not Applicable
CAL046 Trim Loop Current Gain	□ Pass □ Fail □ Not Applicable
CAL047 Write Primary Variable Transfer Function	□ Pass □ Fail □ Not Applicable Supported Transfer Functions (List) ________________________________
CAL049 Write Primary Variable Transducer Serial Number	□ Pass □ Fail □ Not Applicable
CAL050 Read Dynamic Variable Assignments	□ Pass □ Fail □ Not Applicable

Test	Result
CAL051 Write Dynamic Variable Assignments	□ Pass □ Fail □ Not Applicable Device Variables Assignable to PV (List) ________________________________ Device Variables Assignable to SV (List) ________________________________ Device Variables Assignable to TV (List) ________________________________ Device Variables Assignable to QV (List) ________________________________
CAL052 Set Device Variable Zero	□ Pass □ Fail □ Not Applicable Device Variables Supported (List) ________________________________
CAL053 Write Device Variable Units	□ Pass □ Fail □ Not Applicable For each Device Variable the Unit Codes it supports must be attached
CAL054 Read Device Variable Information	□ Pass □ Fail □ Not Applicable
CAL055 Write Device Variable Damping Value	□ Pass □ Fail □ Not Applicable
CAL056 Write Device Variable Transducer Serial Number	□ Pass □ Fail □ Not Applicable
CAL060 Read Analog Channel And Percent Of Range	□ Pass □ Fail □ Not Applicable Analog Channels supported (List) ________________________________
CAL062 Read Analog Channels	□ Pass □ Fail □ Not Applicable Analog Channels supported (List) ________________________________
CAL063 Read Analog Channel Information	□ Pass □ Fail □ Not Applicable Analog Channels supported (List) ________________________________
CAL064 Write Analog Channel Additional Damping Value	□ Pass □ Fail □ Not Applicable Analog Channels supported (List) ________________________________
CAL065 Write Analog Channel Range Values	□ Pass □ Fail □ Not Applicable Analog Channels supported (List) ________________________________
CAL066 Enter/Exit Fixed Analog Channel Mode	□ Pass □ Fail □ Not Applicable Analog Channels supported (List) ________________________________

Test	Result
CAL067 Trim Analog Channel Zero	□ Pass □ Fail □ Not Applicable Analog Channels supported (List) ______________________________
CAL068 Trim Analog Channel Gain	□ Pass □ Fail □ Not Applicable Analog Channels supported (List) ______________________________
CAL069 Write Analog Channel Transfer Function	□ Pass □ Fail □ Not Applicable Analog Channels supported (List) ______________________________
CAL070 Read Analog Channel Endpoint Values	□ Pass □ Fail □ Not Applicable Analog Channels supported (List) ______________________________
CAL071 Lock Device	□ Pass □ Fail □ Not Applicable
CAL072 Squawk	□ Pass □ Fail □ Not Applicable
CAL073 Find Device	□ Pass □ Fail □ Not Applicable
CAL074 Verify I/O System Commands	□ Pass □ Fail □ Not Applicable
CAL078 Command Aggregation	□ Pass □ Fail □ Not Applicable
CAL079 Write Device Variable	□ Pass □ Fail □ Not Applicable
CAL080 Verify Device Variable Trim Commands	□ Pass □ Fail □ Not Applicable
CAL091 Trending	□ Pass □ Fail □ Not Applicable
CAL101 Subsystem Burst Mode	□ Pass □ Fail □ Not Applicable
CAL114 Trigger	□ Pass □ Fail □ Not Applicable
CAL115 Event Notification	□ Pass □ Fail □ Not Applicable
CAL512 Country Code	□ Pass □ Fail □ Not Applicable

ANNEX D. REVISION HISTORY

D.1 Changes from Revision 3.0 to 4.0

This Test Specification is a companion to Revision 9.1 of the *Common Practice Command Specification*. The principal change to this version of the test specification is to provide support for HART 7. Changes to this document include:

- CAL000: Added support (response codes and byte count) for commands introduced and modified by HART 7 including the new 16-bit Common Practice command. Presence and handling of factory only commands are now also assessed.

- CAL001 Verify Write Protect: Added support for HART 7 write commands 77, 87, 88, 89, 92, 97, 99, 102, 103, 104, 106, 108, 109, 116, 117, 118, and 513.

- CAL033 Read Device Variables: Added support for the device variable codes required in HART 7 and later devices.

- CAL040, CAL045, CAL046: Guidance added for assessing 1-5V devices (i.e., devices with analog signaling conections other then 4-20mA).

- CAL042 Perform Device Reset: Added support for the HART 7 command 0 Byte Count.

- CAL044, CAL053: Guidance added for assessing actuators and other devices whose transducer units differ from the PV or Device Variable units.

- CAL051 Write Dynamic Variable Assignments: Added support for the device variable codes required in HART 7 and later devices.

- CAL054 Read Device Variable Information: Added support for "update time period" returned in command 54 responses from HART 7 and later devices.

- CAL071 Lock Device: Added support for wireless products and the additional lock codes introduced in HART 7.

- CAL074 Verify I/O System Commands. Added support for HART 7 commands 77, 84-88, and 94. The original test is now Test Case A. Test Case B, C and D assess the new capabilities introduced with HART 7.

- (New test) CAL078 Command Aggregation was added.

- (New test) CAL091 Trending was added.

- (New test) CAL101 I/O Subsystem Burst Mode was added.

- CAL107 was moved to HCF_TEST-001.

- (New test) CAL115 Event Notification was added.

- (New test) CAL512 Country Code was added.
- IdentifyDevice now fails any device returning Universal Revision > 7
- VerifyAssociatedCommands common test procedure was added
- Failure Point codes were reviewed throughout the document. Duplicate assignments were corrected in several tests.

In addition, the entire document was reviewed for consistency with HART 7 requirements and updated accordingly. A number of minor modifications resulted from this review.

D.2 Changes from Revision 2.0 to 3.0

Test procedures now include support for the Common Practice Command Specification Revision 7 (HART 5) and 8 (HART 6). Thus, manufacturers may use these procedures to check device compatibility with either of these revisions. Specific changes to individual Tests include:

IdentifyDevice now fails any device returning Universal Revision > 6

VerifyNotWriteProtected now treats 251 as identical to not Write Protected.

In addition, a number of minor typos were also corrected.

D.3 Changes from Revision 1.0 to 2.0

This document was updated with Revision 1.2 to reflect changes to referenced documents and to reformat certain document elements. Changes to this document include:

- Tests CAL039, CAL057, CAL058, CAL061, and CAL110 are no longer included because the corresponding command are not recommended. The command specifications are included in the *Common Practice Command Specification* for backward compatibility purposes. Only existing Field Devices migrating to HART 6 that already support these commands should use them.
- Tests CAL059, CAL108, and CAL109 are no longer included as they duplicate testing performed in the *Slave Data Link Layer Test Specification*.
- Tests CAL071 - CAL074, CAL079, and CAL080 are totally new.
- All test were updated to include the new features found in Revision 8.0 of the *Common Practice Command Specification*.

Furthermore, the document as a whole has been reformatted to include new sections: Preface, Introduction, Scope, References, Definitions, Symbols/Abbreviations, Approach, and Deliverables. The additional sections and the new format improves the clarity and consistency of the test specifications.