TEST SPECIFICATION

HART

COMMUNICATION PROTOCOL

Slave Token-Passing Data Link Layer Test Specification

HCF_TEST-001, Revision 3.0

Release Date: 25 February 2009

Release Date: 25 February 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.

HART COMMUNICATION FOUNDATION

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.

The Token-Passing Data Link Layer, at the core of the HART Protocol for traditional devices, establishes the format of HART messages and ensures the reliability of communications. This Test Specification is a companion to Revision 8.2 of the *Token-Passing Data Link Layer Specification* and Revision 7 of the HART Communication Protocol in general.

This is a major revision to the test specifications that includes testing for wireless adapters, wireless field devices (e.g., via the maintenance port), I/O Systems, as well as traditional HART devices. This revision corrects typo's, provides additional clarifications, and makes minor modifications to fine-tune its alignment with the intent of the Protocol Specifications and the Test Specifications themselves. Specific changes include:

- Created a new classification for Burst Mode related tests in Section 5
- Added a new test case to DLL024 to confirm DUT response times for extended (i.e., 16-bit) command numbers.
- DLL036 was modified to confirm proper operation of multiple burst messages. Added macro WriteBurstCommand in DLL036
- DLL037 was modified to accommodate devices supporting TDMA and token-passing data link layers. In addition, three new test cases were added to DLL037
- Added a power Off/On cycle to DLL040. After the power cycle, confirm we found as many devices after the power cycle as before it. In addition a check was added to confirm all of device info after power cycle is unchanged.
- DLL043 was moved from the *Common Practice Test Specifications* (HCF_TEST-4) and improved for HART 7 and wireless device operation.
- DLL044 was added to test HART 7 extended and mixed capabilities of Burst Mode including multiple burst messages and changing burst message configuration on-the-fly.
- DLL045 was added to verify operation of Smart Data Publishing features like threshold burst messages and mixed update intervals.
- Five (5) new test macros were created and CheckReadyForBurst was modified.

In addition, all tests were reviewed and updated, as needed, to reflect any new HART 7 requirements.

Introduction

Testing is an important part of developing a quality product. In developing a HART compatible Field Device, a variety of informal ad-hoc testing and formal tests are performed. HART Protocol testing is an essential part of this effort and mandatory for device registration. The HART Communication Foundation simplifies the testing effort by supplying Test Specifications to developers. 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 Test Plans that must be developed by the manufacturer.
- Can be used early in the development effort to informally verify functionality (e.g., message framing algorithms) during implementation.
- Should be completed along with the Test Report prior to product release.
- Must be completed along with the Test Report for product registration with the HCF.
- Are an essential part 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 Data Link Layer requirements. Data Link Layer Tests become useful early in the development life-cycle, usually, after initial Physical Layer development and testing is complete. In addition to Data Link Layer requirements, several Application Layer commands are used to execute these tests. For example, the implementation of Command 0 should be among the first tests. Command 0 returns identity information that allows the formation of long frame commands that are used throughout the Data Link Layer tests.

The Data Link Layer tests can be classified as follows:

- **Frame Detection And Recognition**. These are a series of tests that confirm a slave properly decodes the byte stream received from the Physical Layer. The Protocol has many features to ensure that communication is error free. This all begins with the detection and framing of a message. Testing includes:
  - Legal and illegal Preamble and Delimiter sequences. The correct detection of a Start of Message and the Delimiter allows the location of the Byte Count to be discerned.
  - Testing of short, long and broadcast addresses.
  - Detection and response to communication errors (e.g., Parity or Check Byte Errors).
  - Decoding of the Byte Count and the reception of the Data Field.

- **Frame Generation**. Once a message addressed to the Field Device has been received, the Field Device must answer. Frame Generation tests confirm that the messages from the Field Device are well formed and error-free. The proper preamble sequence must be generated, the message transmitted with no gaps (inter-character idle time), and the Field Device must release the network with fewer than 2 dribble bytes.
- **Bus Arbitration**. These test ensures the Field Device only transmits messages when indicated by Protocol requirements. The Field Device can accept two levels of responsibility for Bus Arbitration:
  - In the simplest form, the Field Device needs only to answer promptly when addressed by a master. All messages addressed to the Field Device must be answered within the STO interval specified by the Protocol.
  - Field Devices may also support Burst-Mode. Burst-Mode is an optional active communication mode. In this mode the Field Device continuously publishes selected process data without intervention being required by the master. Most of the Bus Arbitration tests are applicable to Burst-Mode operation.
- **Data Link Layer Services**. In addition to message transfer the Data Link Layer provides some additional services to the Application Layer. These include features like the ability to set the number of Preamble characters included in the slave response. In addition, there are several services that support the assignment of Field Device addresses (e.g., Polling Address, Tag and Long Tag). These, in turn, allow the identification of the Field Device (e.g., using Command 11).

Any communications can be successful under normal, well behaved conditions. One measure of the quality of a communication protocol is its performance under adverse conditions. The HART Protocol has an excellent track record for robustness and reliability in the "real world". These tests ensure that Field Devices continue to meet the standards end users have come to expect from HART compatible products.

Compliance with all Protocol requirements is mandatory. All tests in this specification must be executed and passed on all slave devices to demonstrate compliance. Furthermore, all devices must be submitted to the HCF for compliance verification and registration (see *HCF Quality Assurance Program*. HCF_PROC-12).

1. SCOPE

Conformance with the *Data Link Layer Specification* is mandatory. This Test Specification provides Field Device implementers with a set of tests to assist in verifying conformance to the Data Link Layer requirements. Field Devices must successfully complete all applicable tests in this document (see Section 5.2.5).

Note: Some tests may not be applicable to a specific implementation. The test procedure will explicitly state the conditions under which that test is not applicable.

1.1 Features Tested

All major features of the Field Device Data Link Layer are tested. For the Logical Link Control, the features tested include:

- Start of Message Detection;
- Message Framing;
- Short, Long and Broadcast Addresses;
- Error Detection including Parity and Check Byte Errors;
- For asynchronous Physical Layers, Gap and Framing Errors;
- Frame Generation

Tests of the Medium Access Control include

- Slave Response (i.e., ACK) time-out;
- Use of Burst-Mode and Master Address bits; and
- Burst-Mode Operation.

Many of the Data Link Layer and Physical Layer Service Access Points are implicitly tested. In addition, the following Data Link Layer Services are explicitly tested:

- Response Preamble Length;
- Polling Addresses; and
- Long and Short Tags.

1.2 Features Not Tested

A few Data Link Layer requirements are not tested including:

- The Data Link Layer Services to support "Catch Device Variable" (Commands 113, 114) are not tested.
- While many of the tests are applicable to the RS-485 and C8PSK Physical Layers, these Physical Layers are (in general) not explicitly referenced by the tests.
- The response of a slave to a mid-message loss of carrier is not tested.

2. REFERENCES

2.1 The HART-Field Communications Protocol Specifications

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

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

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 *Communications Protocol Specification.* Terms used in this document include: ASCII, Broadcast Address, Data Link Layer, Delayed Response, Delayed Response Mechanism, Device Reset, Device Variable, Busy, 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 Specification* are:

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

DEVICE_STATUS This is the second communication status byte returned in each slave ACK or BACK. The Device Status byte provides 8 bits of status indicating the health and condition of the Field Device (see the *Command Summary Specification*).

Dribble Byte The bytes inadvertently transmitted by a device immediately following the Check Byte. Devices must immediately release the network after transmitting the Check Byte. In other words, a device must not transmit more than one Dribble Byte.

Gap Error A Gap Error occurs when more than one character time elapses after the last receiving a preamble or message byte. For FSK this occurs 9.167ms after the last stop bit is received. Gap Errors are applicable to asynchronous Physical Layers (e.g., FSK, RS-485).

NUMBER_REQUEST_ PREAMBLES This is the number of preamble bytes a slave requests a master to send as indicated in identity (e.g., Command 0) responses.

Preamble A modem training sequence that precedes a HART frame (message). The preamble synchronizes the receiver circuitry and software to allow the successful reception of the message. For FSK, the preamble consists of 2 or more 0xFF bytes received (error free and with no gaps or other intervening characters) immediately before (with no gap) the delimiter field.

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 communications 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*).

Response Time The interval of time between stop bit of the Check Byte in one message and the first bit received in the preamble of the next message. For FSK, the first bit in the preamble is the start bit of the first 0xFF in the preamble.

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
STO	**S**lave **T**ime-**O**ut
SOM	**S**tart **O**f **M**essage

5. APPROACH

This Test Specification uses a "black box" approach to confirming compliance with Data Link Layer requirements. Testing is decomposed into a series of narrowly focused Tests, each containing one of 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 (failure) points are uniquely numbered. This allows cross referencing should the DUT fail a test.

The tests should be performed in sequence shown in Section 5.1. Since there are a relatively large number of tests, for convenience, they are classified by function in Section 5.2.

Burst-Mode support and Data Link Layer services provided by Common Practice Commands are included in the Tests. However, support for these features is mandatory for WirelessHART devices and optional for other devices. As a result, the test procedures verifiy support for any optional requirements prior to the main body of the test. For example, Burst-Mode tests issue Command 109 (with no data bytes). If the Field Device response is "Command Not Implemented" then the test procedure aborts the test. An abort of a test simply indicates that the test is not applicable.

However, if an optional feature is supported (e.g. Command 59 or Burst-Mode), then the Field Device must meet Protocol requirements exactly as specified. The test will proceed to confirm compliance.

5.1 Testing Sequence

Since each test verifies compliance with a specific Data Link Layer 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	DLL032
2	DLL001
3	DLL002
4	DLL003
5	DLL004
6	DLL005
7	DLL006
8	DLL007
9	DLL009
10	DLL010
11	DLL011
12	DLL012
13	DLL013
14	DLL014
15	DLL015
16	DLL040
17	DLL041
18	DLL042
19	DLL017
20	DLL018
21	DLL020
22	DLL024
23	DLL037
24	DLL036
25	DLL016
26	DLL019
27	DLL021
28	DLL022
29	DLL023
30	DLL025
31	DLL026
32	DLL027
33	DLL028
34	DLL029
35	DLL030
36	DLL033
37	DLL034
38	DLL038
39	DLL035
40	DLL039
41	DLL043
42	DLL044
43	DLL045

Note: DLL039, Slave Time-Out Stress Test verifies the long term operation of the DUT Data Link Layer. This test requires DUT operation for a significant amount of time.

5.2 Test Classification

The tests are designed to verify conformance to specific Protocol requirements and the scope of each test is narrow. This section classifies the tests by function.

5.2.1 Frame Detection And Recognition

Frame detection and recognition tests are designed to confirm that the Field Device properly decodes message traffic. Slave responses or the lack of a response verifies conformance.

Table 2 Frame Detection And Recognition Tests

Test	Description	Test	Description
DLL001	FSK Preamble Check.	DLL011	Framing Error Check.
DLL002	Delimiter Check.	DLL012	Check Byte Test.
DLL003	Frame Expansion Check.	DLL013	FSK Gap Receive Timeout Test.
DLL004	Short Frame Check.	DLL014	Long Message Test.
DLL005	Master Address Bit Check.	DLL015	Start Of Message In Data Field Check.
DLL006	Burst Mode Bit Check.		
DLL007	Long Frame Address Check.	DLL040	Unique Address Test
		DLL041	Framing Successive Messages
DLL009	Incorrect Byte Count Check.	DLL042	Command Number Expansion
DLL010	Vertical Parity Check.		

5.2.2 Frame Generation

Tests listed in this section confirm the DUT correctly generates messages.

Table 3 Frame Generation Tests

Test	Description	Test	Description
DLL017	Preamble Check For ACK Frames	DLL020	Dribble Byte Check For ACK Frames
DLL018	Gap Errors in ACK Frames Check		

5.2.3 Bus Arbitration

Bus arbitration tests verify the DUT generates messages when specified by the Protocol and within the correct time limits.

Table 4 Basic Bus Arbitration Tests

Test	Description	Test	Description
DLL024	Test Slave Responds Within STO	DLL039	Slave Time-Out Stress Test

5.2.4 Data Link Layer Services

The DUT must provide services to ensure correct operation of Data Link Layer. These tests are the minimum set required to ensure the Data Link Layer implementation is compliant.

Table 5 Data Link Layer Services Tests

Test	Description	Test	Description
DLL032	Read Unique Identifier (Command 0)	DLL035	Write Number Of Response Preambles (Command 59)
DLL033	Write Polling Address (Command 6)	DLL038	Read Unique Identifier With Long Tag (Command 21)
DLL034	Read Unique Identifier with Tag (Command 11)		

5.2.5 Burst Mode

Burst Mode is a data publishing technology present in HART since Revision 5 of the Protocol was introduced in 1989. In the Token-Passing Data-Link, Burst Mode affects bus arbitration and framing in addition to publishing process data. The tests identified in this subsection confirm proper implementation in Burst Mode capable devices.

In addition, there are many services associated with Burst Mode operation. Standardized tests for these services are also provided in this Test Specification and identified in this subsection.

Token-Passing Data-Link Specific Tests

Burst Mode Frame Generation and Bus Arbitration tests are specific to the Token-Passing Data-Link and are identified in this subsection. Frame Generation test are identified in Table 6.

Table 6 Burst Mode Frame Generation

Test	Description	Test	Description
DLL016	Preamble Check For BACK Frames	DLL022	Host Address Bit in BACK Frames
DLL019	Gap Check For BACK Frames	DLL023	Burst Mode Bit in BACK Frames
DLL021	Dribble Bytes in BACK Frames		

Bus Arbitration ensures that the right device access the bus at the right time. The presence of a Burst Mode device (i.e., a Slave with Burst Mode enabled) significantly changes the Token-Passing sequence. Tests confirming proper Bus Arbitration by a Burst Mode Slave are identified in Table 7.

Table 7 Burst Mode Bus Arbitration Tests

Test	Description	Test	Description
DLL025	Burst Hold During Master Preamble	DLL028	BACK Timing with STXs Errors
DLL026	Burst Response Time After a DUT ACK	DLL029	Burst Mode Timeout On Other Slave
DLL027	BACK Timing Between Bursts	DLL030	Burst Response Time after ACK from Other Slave

Burst Mode Services

Burst Mode Services are applicable to both Token-Passing and TDMA Data-Link. These services allow the Burst Mode process data publishing to be configured and controlled. Applicable tests are identified in Table 8.

Table 8 Burst Mode Services

Test	Description	Test	Description
DLL036	Write Burst Mode Command Number	DLL044	Burst Message Extended Operation
DLL037	Burst Mode Control	DLL045	Smart Data Publishing
DLL043	Write Burst Device Variables		

Burst Mode Command Coverage

The following commands configure and control Burst Mode Operation.

Command 103 Write Burst Period (HART 7 and later)

Command 104 Write Burst Trigger (HART 7 and later)

Command 105 Read Burst Mode Configuration (HART 6 and later)

Command 107 Write Burst Device Variables

Command 108 Write Burst Mode Command Number

Command 109 Burst Mode Control

They must be implemented as a set (i.e., either all or none of the commands shall be implemented). This subsection identifies the test coverage of these commands. Table 9 indicates the commands exercised in each of the Burst Mode tests. In general, Framing and Bus Arbitration only need to enable/disable Burst Mode operation. The Service-related test exercise individual commands more completely.

Table 9 Test Coverage of Burst Mode Commands

Test	Command					
	103	104	105	107	108	109
DLL016 Preamble Check For BACK Frames						X
DLL019 Gap Check For BACK Frames						X
DLL021 Dribble Bytes in BACK Frames						X
DLL022 Host Address Bit in BACK Frames						X
DLL023 Burst Mode Bit in BACK Frames						X
DLL025 Burst Hold During Master Preamble						X
DLL026 Burst Response Time After a DUT ACK						X
DLL027 BACK Timing Between Bursts						X
DLL028 BACK Timing with STXs Errors						X
DLL029 Burst Mode Timeout On Other Slave						X
DLL030 Burst Response Time after ACK from Other Slave						X
DLL036 Write Burst Mode Command Number			X		X	X
DLL037 Burst Mode Control			X			X
DLL043 Write Burst Device Variables			X	X	X	X
DLL044 Burst Message Extended Operation	X	X	X	X	X	X
DLL045 Smart Data Publishing	X	X	X	X	X	X

5.3 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.

5.3.1 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

5.3.2 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 10 Common Response Codes

Slave Indication	Code No.
"Success"	0
"Invalid Selection"	2
"Too Few Data Bytes Received"	5
"Update Failure"	8
"Invalid Extended Command Number"	20
"Busy"	32
"Delayed Response Initiated"	33
"Delayed Response Running"	34
"Command Not Implemented"	64

5.3.3 Device Status

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

- Device Malfunction
- More Status Available
- PV Out of Limits
- Non-PV Out of Limits
- Loop Current Fixed
- Loop Current Saturated
- Cold Start
- Configuration Changed

6. DELIVERABLES

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

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

This 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 should 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.

7. TEST DEFINITIONS

The actual tests are specified in this section. Each test consists of one or more test cases and the corresponding test procedures.

7.1 DLL001 FSK Preamble Check

The Start of Message (SOM) is detected by the reception of a valid Preamble followed immediately by a valid Delimiter. For asynchronous Physical Layers, a valid Preamble consists of receiving two error-free consecutive 0xFF bytes. This test verifies that a DUT only answers messages preceded by a valid preamble. More specifically this test verifies that:

1. Messages preceded by a valid preamble consisting of a varying number of consecutive preamble bytes are accepted by the DUT;
2. That messages preceded by bytes consisting of incorrect preamble values are rejected;
3. Messages preceded by fewer than two preamble bytes are rejected; and
4. That messages preceded by fewer than two consecutive preamble bytes are rejected.

While the requirement is simple to state, the number of possible conditions that can prevent SOM detections is significant. The Test Cases included in this Test are summarized in Table 11. Further testing of FSK Preamble byte sequences is included in DLL010 (parity error), DLL011 (framing error), and DLL013 (gap error).

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	8.1, 8.2, 8.3

Table 11 Preamble Test Case Summary

	Test Case	Requirement
A	Number preamble bytes >= `NUMBER_REQUEST_PREAMBLES`	DUT must answer
B	5 < Number preamble bytes < `NUMBER_REQUEST_PREAMBLES`	DUT must answer intermittently
C	Number preamble bytes = 0 or 1	No DUT response

D	Preamble = 0xFF, 0xFF, … 0xFF, 0x07
	Preamble = 0xFF, 0xFF, … 0xFF, 0x87
	Preamble = 0xFF, 0xFF, … 0xFF, 0x07, 0xFF
	Preamble = 0xFF, 0xFF, … 0xFF, 0x87, 0xFF
		No DUT response

E	Preamble = 0xFF, 0xFF, … 0xFF, 0x07, 0xFF, 0xFF
	Preamble = 0xFF, 0xFF, … 0xFF, 0x87, 0xFF, 0xFF
		DUT must answer

F	Preamble = 0x01, 0x01, … 0x01
	Preamble = 0x02, 0x02, … 0x02
	Preamble = 0xFE, 0xFE, … 0xFE
		No DUT response

Notes

- The only requirement in the Protocol is that framing begins when an SOM is detected. As a result, the only concern of the DUT is whether (at least) two good preambles are received prior to receiving a valid delimiter. The total number of preambles that may be sent by a master is not limited by the Protocol and of no interest to the DUT. As a result, this Primitive Test generates up to 30 0xFF bytes for convenience only.
- While the number of preamble bytes that are required by SOM detect is two, a Field Device can request that the master send more preamble characters. This results in two Test Cases:
  - For messages preceded by the number of Preamble bytes requested by the DUT (or more), the DUT must answer.
  - For messages preceded by 2 Preamble bytes up to the number of Preamble bytes requested by the DUT (or more) the DUT should occasionally answer. (The Protocol requires a Field Device to answer when it sees 2 or more Preamble Bytes.)
- This test is performed using short frame Command 0 and long frame Command 1. SOM detection is assumed to be identical for all other commands and, as a result, not tested.
- When doing the check that messages with fewer than 2 preambles are rejected, the test assumes that one preamble will be lost in the modems (i.e., *n* + 1 preambles are actually transmitted).

Test Case A: More Preamble Bytes Than Requested

Test for responses when a valid number of preamble bytes precede the message starting with the number requested by the DUT.

```
CALL IdentifyDevice
FOR Number_FFs = NUMBER_REQUEST_PREAMBLES to 30
```

Check Response to short frame Command 0.

```
      SEND short frame Command 0
      IF ( COMMUNICATIONS_ERROR )
            THEN Test result is FAIL                                  (600)
      END IF
```

Check response to Command 1.

```
      SEND Command 1
      IF ( COMMUNICATIONS_ERROR )
            THEN Test result is FAIL                                  (601)
      END IF
END FOR
END TEST CASE
```

Test Case B: From 5 up to Number Requested Preamble Bytes

Test for responses when a valid number of preamble bytes precede the message starting from 5 up to the number requested by the DUT. If the DUT answers any messages, then the DUT passes

```
CALL IdentifyDevice
```

Note: this loop is executed one time if `NUMBER_REQUEST_PREAMBLES` is 5

```
FOR Number_FFs =  5 to (NUMBER_REQUEST_PREAMBLES - 1)
      Number_Tries = 0, Pass = FALSE
      WHILE ((Number_Tries < 100) and !Pass)
            SEND Command 2
            IF ( no COMMUNICATIONS_ERROR )
                  THEN Pass = TRUE
            END IF

            Increment Number_Tries
            CALL CheckDeviceAlive
      END WHILE
      IF (!Pass)
            THEN  Test result is FAIL                                 (602)
      END IF
END FOR
END TEST CASE
```

Test Case C: 0 or 1 Preamble Bytes

Now messages are sent with fewer than 2 preceding preamble bytes. The DUT must not answer. The test assumes one preamble byte is lost by the modem during the carrier turn-on transient.

```
CALL IdentifyDevice
FOR Number_FFs = 0 to 1
```

Check Response to short frame Command 0.

```
      SEND short frame Command 0
      IF ( no COMMUNICATIONS_ERROR )
            THEN Test result is FAIL                                  (603)
      END IF
      CALL CheckDeviceAlive
```

Check response to Command 1.

```
            SEND Command 1 in long frame format
            IF ( no COMMUNICATIONS_ERROR )
                  THEN Test result is FAIL                                 (604)
            END IF
            CALL CheckDeviceAlive

      END FOR
      END TEST CASE
```

Test Case D: Non 0xFF in Last Two Preamble Bytes

The following sequence verifies that a DUT does not respond when an invalid preamble character is detected in the preamble.

```
      CALL IdentifyDevice
      FOR (last preamble byte), (second to last preamble byte)
```

Check response to short frame Command 0.

```
            Initialize the preamble byte sequence to all 0xFF's
            Change preamble byte value to 0x07
            SEND (short frame)Command 0 with modified preamble
                  IF ( no COMMUNICATIONS_ERROR )
                        THEN Test result is FAIL                           (605)
                  END IF
            CALL CheckDeviceAlive
```

Check response to Command 1.

```
            Initialize the preamble byte sequence to all 0xFF's
            Change preamble byte value to 0x87
            SEND Command 1 with modified preamble sequence
                  IF ( no COMMUNICATIONS_ERROR )
                        THEN Test result is FAIL                           (606)
                  END IF
            CALL CheckDeviceAlive

            END FOR
      END TEST CASE
```

Test Case E: Non 0xFF Followed By Two 0xFF Bytes

If there is an incorrect preamble byte followed by two valid preamble bytes, the DUT must answer.

```
      CALL IdentifyDevice
      Initialize the preamble byte sequence to all 0xFF's
      Change third to last preamble byte value to 0x07
      SEND short frame Command 0 with modified preamble sequence
      IF ( COMMUNICATIONS_ERROR )
            THEN Test result is FAIL                                       (607)
      END IF

      Initialize the preamble byte sequence to all 0xFF's
      Change third to last preamble byte value to 0x87
      SEND Command 1 with modified preamble sequence
      IF ( COMMUNICATIONS_ERROR )
            THEN Test result is FAIL                                       (608)
      END IF
      END TEST CASE
```

Test Case F: Preamble Sequences with Non 0xFF Values

Check that all possible byte values are used as the value of the preamble bytes. The DUT must not answer.

```
CALL IdentifyDevice
FOR Preamble_Value = 0x00 to 0xFE
      Initialize the preamble byte sequence to Preamble_Value
      SEND Command 1 with modified preamble sequence
      IF ( no COMMUNICATIONS_ERROR )
            THEN Test result is FAIL                                  (609)
      END IF
      CALL CheckDeviceAlive

END FOR
END TEST CASE
```

7.2 DLL002 Delimiter Check

This test verifies that the DUT rejects messages with incorrect Delimiter values. The test sends 255 requests with Command 0, with the delimiter byte being set in value from 0x00 to 0xFE inclusive.

Bits 0-2 indicate the Frame Type and they can only have a value of 1, 2, or 6. Bits 3-4 indicate the Physical Layer and the DUT must answer even if the master sets these incorrectly. Bits 5-6 are used for frame expansion. Since these bits are reserved for Foundation use and not currently assigned, 0(zero) is the only legal value. Bit 7 indicates long or short frame and either value (0 or 1) is valid. As a result, a HART 6 DUT must only respond to messages with delimiter values listed below:

- 0x02, 0x0A, 0x12, 0x1A Short frame format STX.
- 0x82, 0x8A, 0x92, 0x9A Long frame format STX.

The DUT must answer with the delimiter set for the appropriate physical layer (e.g., 0x06 or 0x86 for FSK).

HART 5 stated that the middle 4 bits of the delimiter byte are reserved and set to zero. Consequently, a HART 5 DUT is recommended to support HART 6 delimiters but, not required to do so. In other words, a HART 5 DUT is allowed to respond to only 0x02 and 0x82.

The following delimiters are valid, but they indicate frame expansion. The DUT must not answer.

- 0x22, 0x2A, 0x32, 0x3A, 0x42, 0x4A, 0x52, 0x5A, 0x62, 0x6A, 0x72, 0x7A Short frame format STX.
- 0xA2, 0xAA, 0xB2, 0xBA, 0xC2, 0xCA, 0xD2, 0xDA 0xE2, 0xEA, 0xF2, 0xFA, Long frame format STX.

The test is only carried out for Command 0. It is assumed that processing of the delimiter byte by the DUT will not vary for different command numbers.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.1.1
Data Link Layer Specification	7.1	2.2.4

Test Procedure

```
CALL IdentifyDevice
```

First we will test all the delimiters that might be a short frame command. Then we repeat the process with the delimiters that might be a long frame command.

```
FOR Delimiter byte values from 0 Hex to 0x7F Hex
      SWITCH on (Delimiter Value)

      CASE Delimiter = 0x02
            SEND Command 0 in short frame format
            IF ( COMMUNICATIONS_ERROR )
                  THEN Test result is FAIL                             (616)
            END IF
            IF ((Physical Layer = FSK) AND (DUT Delimiter != 0x06))
                  THEN Test result is FAIL                             (617)
            END IF
            IF ((Physical Layer = PSK) AND (DUT Delimiter != 0x0E))
                  THEN Test result is FAIL                             (618)
            END IF

                  CASE Delimiter = [0x0A, 0x12, or 0x1A]
            SEND Command 0 in short frame format
            IF ( COMMUNICATIONS_ERROR ) AND (UNIV__REVISION == 6)
                  THEN Test result is FAIL                             (620)
            END IF
            IF ((Physical Layer = FSK) AND (DUT Delimiter != 0x06))
                  THEN Test result is FAIL                             (621)
            END IF
            IF ((Physical Layer = PSK) AND (DUT Delimiter != 0x0E))
                  THEN Test result is FAIL                             (622)
            END IF

      CASE Delimiter = [0x22, 0x2A, 0x32, 0x3A, 0x42, 0x4A, 0x52, 0x5A,
         0x62, 0x6A, 0x72, or 0x7A]
            SEND Command 0 in short frame format with frame expansion
            IF ( no COMMUNICATIONS_ERROR )
                  THEN Test result is FAIL                             (623)
            END IF

      CASE DEFAULT
            SEND Command 0 in short frame format
            IF ( no COMMUNICATIONS_ERROR )
                  THEN Test result is FAIL                             (624)
            END IF
      END SWITCH
      CALL CheckDeviceAlive
END FOR
```

```
FOR Delimiter byte values from 0x80 Hex to 0xFE Hex
      SWITCH on (Delimiter Value)

      CASE Delimiter = 0x82
            SEND Command 0 in Long frame format
            IF ( COMMUNICATIONS_ERROR )
                  THEN Test result is FAIL                            (612)
            END IF
            IF ((Physical Layer = FSK) AND (DUT Delimiter != 0x86))
                  THEN Test result is FAIL                            (613)
            END IF
            IF ((Physical Layer = PSK) AND (DUT Delimiter != 0x8E))
                  THEN Test result is FAIL                            (614)
            END IF

      CASE Delimiter = [0x8A, 0x92, or 0x9A]
            SEND Command 0 in Long frame format
            IF ( COMMUNICATIONS_ERROR ) AND (UNIV__REVISION == 6)
                        THEN Test result is FAIL                      (625)
            END IF
            IF ((Physical Layer = FSK) AND (DUT Delimiter != 0x86))
                  THEN Test result is FAIL                            (626)
            END IF
            IF ((Physical Layer = PSK) AND (DUT Delimiter != 0x8E))
                  THEN Test result is FAIL                            (627)
            END IF

      CASE Delimiter = [0xA2, 0xAA, 0xB2, 0xBA, 0xC2, 0xCA, 0xD2, 0xDA
         0xE2, 0xEA, 0xF2, or 0xFA]
            SEND Command 0 in long frame format with frame expansion
            IF ( no COMMUNICATIONS_ERROR )
                  THEN Test result is FAIL                            (628)
            END IF

      CASE DEFAULT
            SEND Command 0 in long frame format
            IF ( no COMMUNICATIONS_ERROR )
                  THEN Test result is FAIL                            (629)
            END IF
      END SWITCH
CALL CheckDeviceAlive
END FOR

END TEST
```

7.3 DLL003 Frame Expansion Check

This test verifies that the DUT can frame messages with expanded frames (i.e., extra byte added between the address and command fields). Since the DUT must not answer a message containing expansion bytes, DUT conformance is verified indirectly.

This test will generate a long frame Command 0 to the DUT and embed it in the data field of a command from a non-existent field device. The header is manipulated such that if a HART 6 DUT does not parse an expanded frame correctly, it will "receive" the Command 0 and erroneously answer the master. (HART 5 DUTs may answer the Command 0 because they may only recognize 0x02 and 0x82 delimiters.)

> Note: DLL002 verified that the DUT does not respond to messages starting with a delimiter indicating frame expansion.

Table 12 Frame Expansion Test Vectors

Expansion Bytes	Test Vector (In Hexadecimal Bytes)
1	`FF FF, …, FF FF A6 AF FA 12 34 56 55 01 0F CD FF FF FF FF FF <Command 0> <check byte>`
2	`FF FF, …, FF FF C6 AF FA 12 34 56 55 02 01 0F CF FF FF FF FF FF <Command 0> <check byte>`
3	`FF FF, …, FF FF E6 AF FA 12 34 56 55 03 02 01 0F CC FF FF FF FF FF <Command 0> <check byte>`

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.1.3

Test Procedure

```
CALL IdentifyDevice
Generate Command 0 to the DUT

FOR (Number of expanded bytes from 1 to 3)
      Generate test message for number of expansion bytes
      SEND test message
      IF (( no COMMUNICATIONS_ERROR ) AND (UNIV__REVISION >= 6))
            THEN Test result is WARNING                      (640 + Number)
      END IF
      CALL CheckDeviceAlive
END FOR
END TEST
```

7.4 DLL004 Short Frame Check

The test checks that only Command 0 is implemented in Short Frame Format. A short frame request with command numbers from 0 to 255, without request data bytes, is sent. The device must only respond to the request with Command 0.

Note: IdentifyDevice uses short frame Command 0. As a result, Command 0 is not explicitly retested in the body of the Test Procedure.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.3.2, 5.3.4
Command Summary Specification	8.0	10.1

Test Procedure

```
CALL IdentifyDevice

FOR (Command number set from 1 to 255).
      SEND short frame command
      IF ( no COMMUNICATIONS_ERROR )
            THEN Test result is FAIL.                                  (650)
      END IF
CALL CheckDeviceAlive
END FOR

END TEST
```

7.5 DLL005 Master Address Bit Check

The test checks that the slave device recognizes valid commands from both Primary and Secondary masters and correctly echoes the master address bit.

The test is only done for Command 0; it is assumed that master address bit processing is the same for other commands.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.3.1

Test Procedure

```
CALL IdentifyDevice

SEND short frame Command 0 with master address bit set to 0.
IF ( COMMUNICATIONS_ERROR )
      THEN Test result is FAIL                                        (660)
END IF
IF ( Master address bit in response is 1)
      THEN Test result is FAIL                                        (661)
END IF

SEND short frame Command 0 with master address bit set to 1
IF ( COMMUNICATIONS_ERROR )
      THEN Test result is FAIL                                        (662)
END IF

IF ( Master address bit in response is 0 )
      THEN Test result is FAIL                                        (663)
END IF

SEND long frame Command 0 with master address bit set to 0
IF ( COMMUNICATIONS_ERROR )
      THEN Test result is FAIL                                        (664)
END IF
IF ( Master address bit in response is 1)
      THEN Test result is FAIL                                        (665)
END IF

SEND long frame Command 0 with master address bit set to 1
IF ( COMMUNICATIONS_ERROR )
      THEN Test result is FAIL                                        (666)
END IF
IF ( Master address bit in response is 0 )
      THEN Test result is FAIL                                        (667)
END IF

END TEST
```

7.6 DLL006 Burst Mode Bit Check

The Burst-Mode bit is used by the master to detect a Burst-Mode Slave on the network and (thus) change the state machine used by the master. This test checks that if the Burst-Mode bit is set in a master request frame, the DUT:

- Ignores the Burst Mode bit;
- Responds with the burst bit reset (i.e., set to zero).

The test is only done for commands 0 and 1; it is assumed that address processing by the DUT is the same for other commands.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.3.1

Test Procedure

This serves to check DUT device when burst mode bit is 0

```
      CALL IdentifyDevice
```

Ensure device is not in Burst Mode. Ignore the response code in case the device does not support Burst Mode.

```
      SEND Command 109 to make device Exit Burst Mode
      IF (COMMUNICATIONS_ERROR)
            THEN Test result is FAIL                                    (401)
      END IF

      SEND Command 0 in short frame format with Burst Mode bit 0
      CALL TestBurstBitResponse(0)                                 (670, 675)
      SEND Command 0 in short frame format with Burst Mode bit 1
      CALL TestBurstBitResponse(1)                                 (671, 676)
      SEND Command 1 in long frame format with Burst Mode bit 0
      CALL TestBurstBitResponse(2)                                 (672, 677)
      SEND Command 1 in long frame format with Burst Mode bit 1
      CALL TestBurstBitResponse(3)                                 (673, 678)

      END TEST
```

TestBurstBitResponse (Location)

The following procedure is used to validate the responses for each combination of commands sent in DLL006. The DUT device must never answer with burst bit set.

```
      PROCEDURE TestBurstBitResponse (Location)
            IF ( COMMUNICATIONS_ERROR )
                  THEN Test result is FAIL                   (670 + Location)
            END IF

            IF Response has Burst Mode bit set.
                  THEN Test result is FAIL                   (675 + Location)
            END IF
      PROCEDURE END
```

7.7 DLL007 Long Frame Address Check

This test checks that the DUT compares all three components of the unique identifier (Manufacturer's Identifier, Device Type, Device Identifier) when doing an address match in long frame format.

The test is only done for Command 1; it is assumed that address processing by the DUT is the same for other commands.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.3.2
Universal Command Specification	6.0	6.1

Test Procedure

```
CALL IdentifyDevice
```

Check for response to long frame address

```
CALL CheckDeviceAlive
```

Now check that there is no response when each byte of address is changed

```
SEND Command 1 with the Manufacturer ID incremented by 1
IF (any response received)
      THEN Test result is FAIL                                     (680)
END IF
CALL CheckDeviceAlive
SEND Command 1 with the Device Type incremented by 1
IF (any response received)
      THEN Test result is FAIL                                     (681)
END IF
CALL CheckDeviceAlive
SEND Command 1 with the Byte 1 of Device ID incremented by 1
IF (any response received)
      THEN Test result is FAIL                                     (682)
END IF
CALL CheckDeviceAlive
SEND Command 1 with the Byte 2 of Device ID incremented by 1
IF (any response received)
      THEN Test result is FAIL                                     (683)
END IF
CALL CheckDeviceAlive
SEND Command 1 with the Byte 3 of Device ID incremented by 1
IF (any response received)
      THEN Test result is FAIL                                     (684)
END IF
CALL CheckDeviceAlive

END TEST
```

7.8 DLL008 (Reserved)

In previous versions of this document, DLL008 confirmed that commands 4 and 5 are not implemented. This testing is now performed in the *Slave Application Layer, Universal Command Test Specification*.

7.9 DLL009 Incorrect Byte Count Check

This test checks that a device correctly deals with messages that have an incorrect byte count (i.e. the byte count does not agree with the number of data bytes actually transmitted). The DUT must respond as follows:

- If the byte count is greater than the number of data bytes, then the device must not respond. This is effectively one type of gap error.
- If the byte count is less than the number of data bytes then the device must respond with a "Longitudinal Parity Error" in the Communications Status byte. The device must not return any data bytes in the response.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.2, 5.4, 7.2.1
Universal Command Specification	8.0	6.1, 6.4

Test Procedure

```
CALL IdentifyDevice
FOR Command = (long frame)0, 3
```

Check response with correct Byte Count

```
        SEND Command with Byte Count of 0 and 0 bytes of data
        IF ( COMMUNICATIONS_ERROR )
              THEN Test result is FAIL                                  (700)
        END IF
```

Check response Byte Count > number of Data Bytes. DUT must not answer

```
        SEND Command with Byte Count of 9 and 5 bytes of data
        IF ( any response received )
              THEN Test result is FAIL                                  (701)
        END IF
        CALL CheckDeviceAlive
```

Check response Byte Count < number of Data Bytes. Ensure in this test that the fifth extra data byte, which will be interpreted as the Checksum Byte, does not give a correct checksum

```
        SEND Command with Byte Count of 4 and 5 bytes of data
        IF (COMMUNICATIONS_ERROR != "Longitudinal Parity Error")
              THEN Test result is FAIL                                  (702)
        END IF
        IF ( Byte Count is not 2 )
              THEN Test result is FAIL                                  (703)
        END IF

END FOR
END TEST
```

7.10 DLL010 Vertical Parity Check

This test checks that a device correctly deals with messages that have parity errors in each of the fields in the frame. The device must handle parity errors as specified in the *Data Link Layer Specification*.

The test for parity error in the Data Bytes inserts data bytes into commands 0 and 2; strictly, these do not contain data bytes. This confirms that parity error checking is independent of command numbers.

Note: While the Preamble, strictly speaking, is a Physical Layer requirement, parity errors in either of the two preamble bytes immediately preceding the Delimiter prevent SOM from being detected. Consequently, any parity errors in the preamble (or delimiter) must result in no response from the DUT. This condition is tested here along with the other fields.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.4
Command Summary Specification	8.0	7.4.1, 9

Test Procedure

```
      CALL IdentifyDevice

      FOR (Command short frame 0, Command long frame 0, Command 2)
```

Check response to valid command

```
            SEND command.
            IF ( COMMUNICATIONS_ERROR )
                  THEN Test result is FAIL                                  (710)
            END IF
```

Test fatal (i.e., no response) parity errors.

```
            FOR (Preamble, Delimiter, Each Address Byte, Byte Count)
                  SEND command with parity error designated field
                  IF   ( COMMUNICATIONS_ERROR != "No Response" )
                        THEN Test result is FAIL                (711 + Iteration)
                  END IF
                        CALL CheckDeviceAlive
            END FOR
```

Test that the DUT device really aborts the message with the parity error on the Byte Count.

```
            SEND command with Byte Count = 240 and parity error
                   immediately followed (i.e. with no gap) by a good command
            IF   ( COMMUNICATIONS_ERROR != "No Response" )
                  THEN Test result is FAIL                                  (715)
            END IF
```

Test non-fatal parity errors. For Data Field send the command with 5 data bytes and a parity error in the second data byte

```
            FOR (Command, Data Byte, Check Byte)
                  SEND command with parity error in the designated field
                  IF (COMMUNICATIONS_ERROR != "Parity Error")
                        THEN Test result is FAIL                (716 + Iteration)
                  END IF
            END FOR

      END FOR

      END TEST
```

7.11 DLL011 Framing Error Check

This test checks that a device correctly deals with messages that have framing errors in each of the bytes of the frame. The device must handle parity errors as specified in the *Data Link Layer Specification*.

The test for framing error in the Data Bytes inserts data bytes into commands 0 and 2; strictly, these do not contain data bytes. This confirms that framing error checking is independent of command numbers.

> Note: While the Preamble, strictly speaking is a Physical Layer requirement, framing errors in either of the two preamble (0xFF) bytes immediately preceding the Delimiter prevent SOM from being detected (i.e., the DUT must not answer the message). This condition is tested here along with the other fields.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.4
Command Summary Specification	8.0	7.4.1, 9

Test Procedure

```
CALL IdentifyDevice

FOR (Command short frame 0, Command long frame 0, Command 2)
```

Check response to valid command

```
      SEND command
      IF ( COMMUNICATIONS_ERROR )
            THEN Test result is FAIL                                  (720)
      END IF
```

Test fatal (i.e., no response) framing errors

```
      FOR (Preamble, Delimiter, Each Address Byte, Byte Count)
            SEND command with framing error designated field
            IF   ( COMMUNICATIONS_ERROR != "No Response" )
                  THEN Test result is FAIL                (721 + Iteration)
            END IF
      END FOR
```

Test that the DUT device really aborts the message with the framing error on the Byte Count.

```
      SEND command with Byte Count = 240 and framing error
             immediately followed (i.e. with no gap) by a good command
      IF   ( COMMUNICATIONS_ERROR != "No Response" )
            THEN Test result is FAIL                                  (725)
      END IF
```

Test non-fatal framing errors. For Data Field send the command with 5 data bytes and a framing error in the second data byte

```
            FOR (Command, Data Byte, Check Byte)
                  SEND command with framing error designated field
                  IF (COMMUNICATIONS_ERROR != "Framing Error")
                        THEN Test result is FAIL                (726 + Iteration)
                  END IF
            END FOR

      END FOR

      END TEST
```

7.12 DLL012 Check Byte Test

This test checks that a DUT validates the Check Byte and reports a "Longitudinal Parity Error" when an incorrect Check Byte is sent.

The test only checks Command 0 and 3, it is assumed that other commands are processed in the same way.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	

Test Procedure

```
      CALL IdentifyDevice

      FOR(short frame Command 0, long frame Command 0, Command 3)
```

Check response to valid command

```
            SEND command
            IF (COMMUNICATIONS_ERROR)
                  THEN Test result is FAIL                                  (730)
            END IF
```

Test for validation of checksum.

```
            SEND command with incorrect checksum
            IF  (COMMUNICATIONS_ERROR != "Longitudinal Parity Error")
                  THEN Test result is FAIL                                  (731)
            ELSE
                  IF (BYTE_COUNT != 2
                        THEN Test result is FAIL                            (402)
                  END IF
            END IF
      END FOR

      END TEST
```

7.13 DLL013 FSK Gap Receive Timeout Test

This test verifies that the DUT aborts message framing when any gap time between successive bytes is greater than the Gap Time limit (9.167 ms). After the timeout, the DUT must be able to receive a subsequent message. There are three fundamental test conditions:

- Message 1 is terminated after a specified field (i.e., Preamble, Delimiter, Address, Command, Byte Count, Data). After a 14ms gap time, Message 2 is sent. The Message 2 must be answered. The test messages are shown in Table 13.
- A 14ms gap is injected between two fields in Message 1. After the gap, the balance of the message is transmitted. The message must not be answered.
- A 4ms gap is injected between two fields in a Message 1. After the gap, the balance of the message is transmitted. The message must be answered.

The gap times used are approximately 0.5 (4ms) and 1.5 (14ms) character times. The carrier must not transition during any of these test sequences. No dribble bytes may occur during the gap.

Note: This test does not simulate a mid-message loss of carrier and verify the DUT's response to this condition.

Table 13 STX's for Receive Gap Tests

	Message Content
1	0xFF 0xFF, ... , 0xFF, 82, <DUT address>, 0x00, 0x20, 0x01, 0x02, 0x03, ... , 0x1E, 0x1F, 0x20, <check byte>
2	0xFF, 0xFF, ... , 0xFF, 82, <DUT address>, 0x02, 0x00, <check byte>

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.4, 7.2.1, 8.1, 8.2

Test Procedure

Check for gaps after each of the Preamble, Delimiter, Address, Command and Byte Count bytes followed by a good message.

```
CALL IdentifyDevice
FOR (FIELD = [Preamble, Delimiter, Each Address Byte, Command, Byte Count])
      SEND Message 1 up to and including FIELD
      Wait 14 ms
      SEND normal Message 2
      IF (COMMUNICATIONS_ERROR)
            THEN Test result is FAIL                        (450 + Iteration)
      END IF
      IF (response Command != 2)
            THEN  Test result is FAIL                       (460 + Iteration)
      EDN IF
END FOR
```

```
FOR (BYTE_COUNT = [1 to 0x20] )
      SEND Message 1 up to and including BYTE_COUNT
      Wait 14 ms
      SEND normal Message 2
      IF (COMMUNICATIONS_ERROR)
            THEN Test result is FAIL                                  (490)
      END IF
END FOR
```

Check for gaps after each of the Preamble, Delimiter, Address, Command and Byte Count bytes followed by no message.

```
FOR (FIELD = [Preamble, Delimiter, Each Address Byte, Command, Byte Count])
      SEND Message 1 up to and including FIELD
      IF (no COMMUNICATIONS_ERROR)
            THEN Test result is FAIL                      (470 + Iteration)
      END IF
END FOR

FOR (BYTE_COUNT = [1 to 0x20] )
      SEND Message 1 up to and including BYTE_COUNT
      IF (no COMMUNICATIONS_ERROR)
            THEN Test result is FAIL                                  (491)
      END IF
END FOR

END TEST
```

Insert some idle time but not enough for a gap error after each of the Preamble, Delimiter, Address, Command and Byte Count bytes.

```
CALL IdentifyDevice
FOR (FIELD = [Preamble, Delimiter, Each Address Byte, Command, Byte Count])
      SEND Message 1 up to and including FIELD
      Wait 4 ms
      SEND the rest of the message
      IF (COMMUNICATIONS_ERROR)
            THEN Test result is FAIL                      (480 + Iteration)
      END IF
END FOR

FOR (BYTE_COUNT = [1 to 0x20] )
      SEND Message 1 up to and including BYTE_COUNT
      Wait 4 ms
      SEND the rest of the message
      IF (COMMUNICATIONS_ERROR)
            THEN Test result is FAIL                                  (492)
      END IF
END FOR

END TEST
```

7.14 DLL014 Long Message Test

This test checks that the device generates the Buffer Overflow error in the Communications Error Summary (response byte 1) only when the number of data bytes in a message exceeds 89 for WirelessHART, 32 for HART 6 and 7, and 24 for HART 5.

The test is only run for commands 0 and 3, it is assumed that all other commands are processed in the same way.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	

Test Procedure

```
CALL IdentifyDevice

IF (UNIV__REVISION < 6)
      THEN minBufferSize = 25
ELSE IF (UNIV__REVISION >= 6 AND NOT WIRELESS)
      THEN minBufferSize = 33
ELSE
      THEN minBufferSize = 90
END IF


FOR Commands 0 and 3
   FOR Number of data bytes in frame = [0 to minBufferSize], 40, 128, 240

      SEND long frame command
      IF (COMMUNICATIONS_ERROR == "No Response") THEN
         PRINT <Byte Count>
         Test result is FAIL                                          (750)
      END IF

      IF ( (COMMUNICATIONS_ERROR == "Buffer Overflow") AND
                  (Number of data bytes was < minBufferSize) ) THEN
         PRINT <Byte Count>
         Test result is FAIL                                          (751)
      END IF

   END FOR
END FOR

END TEST
```

7.15 DLL015 Start Of Message In Data Field Check

The ability of the device to ignore messages with another message embedded in the data field is tested. Some of the requests and embedded messages are addressed to the device and some are not.

The test cases in Table 14 specify the format of the request, the address in the request and the address of the command in the data field. The right hand column specifies whether a response is expected . For example, test case 1 sends:

> "Short frame Command 0, not addressed to the DUT, Command 1 not addressed to the DUT in the data field. No response must be received from the device under test."

Table 14 Embedded Message Test Cases

Case	Command 0 Format	Command 0 Address	Command 1 Embedded Address	Response Expected
1	SF	Not to DUT	Not to DUT	No
2	SF	Not to DUT	To DUT	No
3	SF	To DUT	Not to DUT	Yes, Command 0
4	SF	To DUT	To DUT	Yes, Command 0
5	LF	Not to DUT	Not to DUT	None
6	LF	Not to DUT	To DUT	None
7	LF	To DUT	Not to DUT	Yes, Command 0
8	LF	To DUT	To DUT	Yes, Command 0

LEGEND for Format Code

SF	Short frame format
LF	Long frame format

Note: in Test Cases 3, 4, 7, and 8 the Response Code must equal Success (i.e. 0).

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.1.6

Test Procedure

```
CALL IdentifyDevice

FOR The request and embedded message cases in Table 14

      SWITCH on (TEST_CASE)
      CASE TEST_CASE = [1, 2, 5 or 6]
            IF any Response received
                  THEN Test result is FAIL                (760 + TEST_CASE)
            END IF

      CASE TEST_CASE = [3, 4, 7 or 8]
            IF (No Response received) OR (Response != Command 0)
                  THEN Test result is FAIL                (760 + TEST_CASE)
            END IF

END FOR

END TEST
```

7.16 DLL016 Preamble Check For BACK Frames

Checks the number of preamble characters in DUT burst acknowledges (BACKs). The number of Preambles in a DUT response must be in the range of 2 to 20 inclusive.

This test assumes that the DUT sends enough preambles to compensate for preamble characters that may be lost by the receiving modem.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	8.1, 8.2
Common Practice Command Specification	8.0	7.27

Test Procedure

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode

```
CALL CheckReadyForBurst()

SEND Command 109 to put the DUT into Burst Mode
FOR (5 minutes)
      IF (BACK preamble count > 20)
            THEN Test result is FAIL                                   (770)
      END IF
      IF (BACK preamble count < 2)
            THEN Test result is FAIL                                   (771)
      END IF
END FOR

SEND Command 109 to take the DUT out of Burst Mode
IF (number of BACKs < 539)
      THEN Test result is FAIL                                         (391)
END IF

END TEST
```

7.17 DLL017 Preamble Check For ACK Frames

Checks the number of preamble characters in short and long frame DUT responses (ACKs). The number of Preambles in a DUT response must be in the range of 2 to 20 inclusive.

This test assumes that the DUT sends enough preambles to compensate for preamble characters that may be lost by the receiving modem.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	8.1, 8.2
Common Practice Command Specification	8.0	7.27

Test Procedure

```
CALL IdentifyDevice

FOR (100 iterations)
      SEND Short Frame Command 0
      IF (COMMUNICATIONS_ERROR)
            THEN Test result is FAIL                                   (780)
      END IF
      IF (ACK preamble count > 20)
            THEN Test result is FAIL                                   (781)
      END IF
      IF (ACK preamble count < 2)
            THEN Test result is FAIL                                   (782)
      END IF

      SEND Command 3
      IF (COMMUNICATIONS_ERROR)
            THEN Test result is FAIL                                   (783)
      END IF
      IF (ACK preamble count > 20)
            THEN Test result is FAIL                                   (784)
      END IF
      IF (ACK preamble count < 2)
            THEN Test result is FAIL                                   (785)
      END IF
END FOR

END TEST
```

7.18 DLL018 Gap Errors in ACK Frames Check

Verifies that bytes within a DUT response are "contiguous". There must be no gaps greater than one byte time between the bytes of a message. More specifically, the start bit of one character must follow the stop bit of the previous character within 9.167 ms.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.1, 7.2.1, 7.2.3, 8.1.2, 8.2.2

Test Procedure

```
CALL IdentifyDevice

FOR (100 iterations)
      FOR (Commands 1, 3, 12, and 13)
            SEND Command
            IF (COMMUNICATION_ERROR)
                  THEN Test result is FAIL                            (790)
            END IF
            IF (any Gap Errors)
                  THEN Test result is FAIL                            (791)
            END IF
      END FOR

      SEND Short Frame Command 0
      IF (No Response)
            THEN Test result is FAIL                                  (792)
      END IF
      IF (any Gap Errors)
            THEN Test result is FAIL                                  (793)
      END IF
END FOR

IF UNIV__REVISION > 5 THEN
      SEND Command 20 to read LONG_TAG
```

Find the supported Device Variables

```
            SET dVarList = null
            dVar = -1
            While (FindNextDeviceVariable(dVar) != "No More Device Variables")
                  ADD dVar to dVarList
            END WHILE

            Print the dVarList

            FOR (100 iterations)
                  FOR ( all device variables )
                        SEND Command 9 with sets of 4 device variables
                        IF (COMMUNICATION_ERROR)
                              THEN Test result is FAIL                      (794)
                        END IF
                        IF (RESPONSE_CODE != 0) AND
                          (RESPONSE_CODE != "Update Failure")
                              THEN Test result is FAIL                      (796)
                        END IF
                        IF (any Gap Errors)
                              THEN Test result is FAIL                      (795)
                        END IF
                  END FOR

                  SEND Command 21
                  IF (COMMUNICATION_ERROR)
                        THEN Test result is FAIL                            (276)
                  END IF
                  IF (RESPONSE_CODE != 0)
                        THEN Test result is FAIL                            (277)
                  END IF
                  IF (any Gap Errors)
                        THEN Test result is FAIL                            (278)
                  END IF
            END FOR
      END IF

      END TEST
```

7.19 DLL019 Gap Check For BACK Frames

Verifies that bytes within a DUT response are "contiguous". There must be no gaps greater than one byte time between any bytes in a message. More specifically, the start bit of one character must follow the stop bit of the previous character within 9.167 ms.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.1, 7.2.1, 7.2.3, 8.1.2, 8.2.2

Test Procedure

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode

```
CALL CheckReadyForBurst()

SEND Command 109 to put the DUT into Burst Mode

FOR (5 Minutes)
      IF (Any Gap Errors)
            THEN Test result is FAIL                                 (799)
      END IF
END FOR

SEND Command 109 to take the DUT out of Burst Mode

END TEST
```

7.20 DLL020 Dribble Byte Check For ACK Frames

This test checks for dribble bytes after DUT ACKs.

Masters must begin their transmission within two character times after a Field Device ACK. As a result, slave devices must promptly release the network after completing a message. This test verifies that the DUT is off the network within one character after the Check Byte. In other words, the DUT must generate no more than one dribble byte.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	
FSK Physical Layer Specification	8.1	

Test Procedure

```
CALL IdentifyDevice

SEND Short Frame Command 0
IF COMMUNICATIONS_ERROR)
      THEN Test Result FAIL                                                (800)
END IF
IF (RESPONSE CODE != 0)
      THEN Test Result FAIL                                                (801)
END IF
IF more than one Dribble Byte Received
      THEN Test Result FAIL                                                (802)
END IF

FOR Command 3, 13, 11, 12)
      SEND Command
      IF COMMUNICATIONS_ERROR)
            THEN Test Result FAIL                                          (803)
      END IF
      IF (RESPONSE CODE != 0)
        AND ((RESPONSE CODE != Upadate Failure) OR (Command != 3))
            THEN Test Result FAIL                                          (804)
      END IF
      IF more than one Dribble Byte Received
            THEN Test Result FAIL                                          (805)
      END IF
END FOR

END TEST
```

7.21 DLL021 Dribble Byte Test For BACK Frames

This test checks for dribble bytes after DUT BACKs.

Masters must begin their transmission within two character times after a Field Device BACK. As a result, slave devices must promptly release the network after completing a message. This test verifies that the DUT is off the network within one character after the Check Byte. In other words, the DUT must generate no more than one dribble byte.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	See Caveats

Test Procedure

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode

```
CALL CheckReadyForBurst()

SEND Command 109 to put the DUT into Burst Mode

FOR (5 minutes)
      IF (More Than One Dribble Byte Received With Any BACK)
            THEN Test result is FAIL                                  (810)
      END IF
END FOR
SEND Command 109 to take the DUT out of Burst Mode

END TEST

Note: If fewer than 539 BACKs received
      THEN Test result is FAIL                                        (811)
```

7.22 DLL022 Test Host Address Bit For BACK Frames

When the Slave is in Burst Mode, for each Burst, it must change the Host address. The master address in the BACK controls which master gets access to the network.

Test will put the DUT into Burst Mode and then periodically inject requests from Primary and Secondary Hosts. Then the DUT Responses are analyzed for HART Conformance. The DUT must toggle the burst address every burst (even if a master incorrectly accesses the network).

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.3.1

Test Procedure

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode

```
CALL CheckReadyForBurst()

SEND Command 109 to put the DUT into Burst Mode
Record BACK Response Time
IF (Response Time > STO)
      THEN Test result is FAIL                                  (404)
ELSE  IF (Response Time > (RT2))
      THEN Test result is WARNING                               (114)
END IF
IF (First BACK does NOT have this master's address)
      THEN Test Result FAIL                                     (422)
END IF

FOR (5 minutes)
      IF (any two successive BACKs have the same master address)
            THEN Test result is FAIL                            (812)
      END IF
END FOR

FOR (20 BACKs)
      SEND Command 3 with primary master address after every BACK
      IF (No Response)
            THEN Test Result FAIL                               (813)
      END IF
      IF (RESPONSE CODE != "Sucess" or "Update Failue")
            THEN Test Result FAIL                               (375)
      END IF
      IF (ACK to secondary master)
            THEN Test result is FAIL                            (814)
      END IF
      IF (any two successive BACKs have the same master address)
            THEN Test result is FAIL                            (815)
      END IF
END FOR
```

```
FOR (20 BACKs)
      SEND Command 3 with secondary master address after every BACK
      IF (No Response)
            THEN Test Result FAIL                                       (816)
      END IF
      IF (RESPONSE CODE != "Sucess" or "Update Failue")
            THEN Test Result FAIL                                       (376)
      END IF
      IF (ACK to primary master)
            THEN Test result is FAIL                                    (817)
      END IF
      IF (any two successive BACKs have the same master address)
            THEN Test result is FAIL                                    (818)
      END IF
END FOR

SEND Command 109 to take the DUT out of Burst Mode

END TEST
```

7.23 DLL023 Test Burst Mode Bit Of Burst-Mode Slave Frames

When the Field Device is in Burst Mode, all Responses must have the Burst Mode bit set. The configuration of the Burst Mode Bit in a Host Request must be irrelevant to the Field Device.

Test will put the DUT into Burst Mode and then periodically inject requests from Primary and Secondary Hosts, and will check to make sure that responses to Host requests as well as Burst responses all have the Burst Mode bit set.

Note: DLL006 Burst Mode Bit Check confirms that the DUT resets the burst mode bit when the DUT is not in burst mode.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.3.1

Test Procedure

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode

```
CALL CheckReadyForBurst
SEND Command 109 to put the DUT into Burst Mode
FOR (Master = Primary, and Secondary)
      FOR (20 BACKs)
            SEND Command 3 with BURST bit reset
            IF (No Response)
                  THEN Test result is FAIL                              (820)
            END IF
            IF (ACK with BURST bit reset)
                  THEN Test result is FAIL                              (821)
            END IF
      END FOR
      Pause for at least 3 Bursts to occur
      FOR (20 BACKs)
            SEND Command 3 with BURST bit set
            IF (No Response)
                  THEN Test result is FAIL                              (822)
            END IF
            IF (ACK with BURST bit reset)
                  THEN Test result is FAIL                              (823)
            END IF
      END FOR
      Pause for at least 3 Bursts to occur
END FOR
SEND Command 109 to take the DUT out of Burst Mode
If (number of message cycles < 80)
      THEN Test result is FAIL                                          (373)
END IF

END TEST
```

7.24 DLL024 Test Slave Responds Within STO

A Field Device must respond to all valid master requests within the STO. This test verifies compliance (i.e., the response times are validated) for both Short and Long Frame requests.

Note: DLL039 Slave Time-Out Stress Test performs further statistical validation of this requirement.

In Test Case B all commands (except 11, 21, 39, 41, 42 or 73) are tested for response time. This confirms that the message parser in the DUT can always parse and generate an answer within STO. Test procedures for commands 11 and 21 are in DLL034 and DLL038 respectively. Test procedures for commands 39, 41, 42, and 73 can be found in the *Slave Common Practice Command, Test Specification*.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	7.2.3

Test Case A: Verify STO for Selected Universal Commands

```
CALL IdentifyDevice

FOR (100 iterations)
      FOR (command = [3, 12, and 13] )
      Call CheckSlaveSTO (command)
      END FOR

      Call CheckSlaveSTO (short frame command 0)

      IF UNIV__REVISION > 5
            FOR (command = [9, and 20] )
                  Call CheckSlaveSTO (command)
            END FOR
      END IF
END FOR

END TEST CASE
```

Test Case B: Verify STO for All Commands

```
CALL IdentifyDevice

For (command = 1-253)
      IF (command != [11, 21, 39, 41, 42 OR 73] ) THEN
            Call CheckSlaveSTO (command)
      ENDIF
END FOR
END TEST CASE
```

Test Case C: Verify STO for Extended Command Numbers

Warning: This test could take several hours to complete

```
CALL IdentifyDevice

IF (UNIV_REVISION < 6) THEN
   SEND Command 31 with no data bytes
   Record Slave Response Time
   IF (No Response)
      THEN Test result is FAIL                                            (435)
   END IF
   IF (RESPONSE_CODE != "Command not Implemented")
      THEN Test result is FAIL                                            (436)
   END IF
   IF (Response Time > STO)
      THEN Test result is FAIL                                            (437)
   END IF
ELSE
   SEND Command 31 with no data bytes
   IF (RESPONSE_CODE = "Command not Implemented")
      THEN Abort Test (i.e., test is not applicable to this device)
   END IF
```

Now test all 65K possible commands, these are all sent with 16bit command numbers and the expansion flag (command 31). Device must answer even 8 bit commands this way.

```
   FOR (command = 0-0xFFFF)
      IF (command != [11, 21, 39, 41, 42 OR 73] ) THEN
         Call CheckSlaveSTO (with expanded 16-bit command)
      ENDIF
   END FOR

ENDIF

END TEST CASE
```

7.25 DLL025 Burst Hold During Master Preamble

This test verifies that the Burst-Mode Slave defers its burst while the preamble of the master accessing the network is in progress. In other words, the preamble from a master will normally last longer than the link grant time. However, while the preamble is in progress, the DUT must not issues a BACK. The following specific cases are tested:

- If spurious preambles are seen for longer than RT2, then the DUT should burst immediately after their conclusion.
- If a master message's preamble completes after RT2 but is contiguous with the delimiter, then the DUT must answer and follow the ACK immediately with a BACK
- If spurious characters are observed, the DUT must enter its RCV_MSG routine and stay there until no further characters are observed (i.e., until `ENABLE.indicate` is de-asserted). If this transient condition persists past RT2 then the DUT must immediately burst.
- The burst must start as soon as possible but no later than STO after the last BACK.

Note: RT2 is the maximum amount of time required to detect the immediate transmission of a message (see Annex C of the *Data Link Layer Specification*). As a result, it is used as the time limit in this Test.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	7.2.4, 8, 8.1, 8.2

Test Procedure

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode

```
CALL CheckReadyForBurst()

SEND Command 109 to put the DUT into Burst Mode

CALL CheckDeviceAlive
```

The DUT must begin answer within STO of last BACK. This requirement is measured as RT1 after the BACK (BACK-BACK Response Time) or RT2 after the generated disturbance (Pushed BACK Response Time), whichever is greater.

```
FOR (100 iterations)
   FOR NUM_PREAMBLES = (9, 10, 16, 20, 32)
      SEND NUM_PREAMBLES of 0xFFs and release network
      Record Pushed BACK Response Time
      RECORD BACK-BACK RESPONSE TIME

      IF (COLLISION)
         THEN TEST RESULT IS FAIL                                     (825)
      END IF

      IF (NUM_PREAMBLES == 32) THEN
         IF (PUSHED BACK RESPONSE TIME > RT2)
            THEN TEST RESULT IS FAIL                                  (826)
```

```
          END IF
        ELSE
          IF (BACK-BACK RESPONSE TIME > RT1)
            THEN TEST RESULT IS FAIL                                       (415)
          END IF
        END IF

        SEND COMMAND 1, WITH NUM_PREAMBLES
        IF (COMMUNICATIONS_ERROR)
          THEN TEST RESULT IS FAIL                                         (827)
        END IF
        IF (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "UPDATE FAILURE")
          THEN TEST RESULT IS FAIL                                         (828)
        END IF

        SEND NUM_PREAMBLES OF RANDOM BYTE VALUES AND RELEASE NETWORK
        RECORD PUSHED BACK RESPONSE TIME
        RECORD BACK-BACK RESPONSE TIME

        IF (NUM_PREAMBLES == 32) THEN
          IF (PUSHED BACK RESPONSE TIME > RT2)
            THEN TEST RESULT IS FAIL                                       (829)
          END IF
        ELSE
          IF (BACK-BACK RESPONSE TIME > RT1)
            THEN TEST RESULT IS FAIL                                       (416)
          END IF
        END IF

        IF (COLLISION)
          THEN TEST RESULT IS FAIL                                         (832)
        END IF

        SEND NUM_PREAMBLES OF RANDOM BYTE VALUES FOLLOWED BY VALID COMMAND 1
        IF (COMMUNICATIONS_ERROR)
          THEN TEST RESULT IS FAIL                                         (830)
        END IF
        IF (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "UPDATE FAILURE")
          THEN TEST RESULT IS FAIL                                         (831)
        END IF
      END FOR
    END FOR

    SEND COMMAND 109 TO TAKE THE DUT OUT OF BURST MODE

    END TEST

   Note: If fewer than 1000 Command 1 STX's detected (e.g. due to a Collision)
        THEN Test result is FAIL                                           (833)
```

7.26 DLL026 Test Burst Response Time After a DUT ACK

The DUT should begin a BACK immediately after the Check Byte of an ACK. Under no circumstances can the BACK begin later than STO.

The test puts the DUT into Burst Mode and then send Short and Long frame requests to the DUT verifying the BACK time after each ACK.

> Note RT2 is the maximum amount of time required to detect the immediate transmission of a message (see Annex C of the *Data Link Layer Specification*). As a result, it is used to confirm the BACK started immediately after the ACK.

References:

Specification	**Rev.**	**Sections**
Data Link Layer Specification	8.0	7.2.3

Test Procedure

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode

```
CALL CheckReadyForBurst()

SEND Command 109 to put the DUT into Burst Mode

FOR (100 iterations)
      FOR (command = [3, 12, and 13] )
            Call CheckSlaveSTO (command)
            CALL CheckBackTime()
      END FOR

      CALL CheckSlaveSTO (short frame Command 0)
      CALL CheckBackTime()
```

See if the DUT supports HART 6 or later

```
      IF (UNIV__REVISION > 5)
            FOR (command = [9, and 21] )
                  Call CheckSlaveSTO (command)
                  CALL CheckBackTime()
            END FOR
      END IF
END FOR

SEND Command 12 (to read msg0)
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                          (423)
END IF
IF (RESPONSE_CODE != 0)
      THEN Test result is FAIL                                          (424)
END IF

Create msg1 by bit-wise inverting msg0
SEND Command 17 (with msg1)
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                          (405)
```

```
END IF
IF (RESPONSE_CODE != 0)
      THEN Test result is FAIL                                        (406)
END IF
CALL CheckBackTime()

SET BUSY_CNT = 0
DO
      SEND Command 2
      IF (COMMUNICATIONS_ERROR)
            THEN Test result is FAIL                                  (407)
      END IF
      IF (RESPONSE_CODE != "Success", "UpdateFailure", or "Busy")
            THEN Test result is FAIL                                  (408)
      END IF
      IF BUSY_CNT > 10
            THEN Test result is FAIL                                  (431)
      END IF
WHILE (RESPONSE_CODE == "Busy")

SEND Command 17 (with msg0)
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                        (409)
END IF
IF (RESPONSE_CODE != 0)
      THEN Test result is FAIL                                        (425)
END IF
CALL CheckBackTime()

SEND Command 109 to take the DUT out of Burst Mode

END TEST
```

CheckBackTime ( )

The following procedure is used to validate ACK to BACK response times.

```
PROCEDURE CheckBackTime ( )
Record ACK to BACK Response Time
IF (Response Time > STO)
      THEN Test result is FAIL                                        (835)
ELSE IF (Response Time > RT2)
      THEN PRINT "WARNING: Long ACK to BACK response                  (100)
        time threatens bus arbitration integrity. BACK
        should immediately follow ACK with no gap"
END IF
PROCEDURE END

Note: If there is a large number of Preamble bytes recieved
      THEN PRINT "Warning large number of preambles
                  degrade Cyclic data throughput                      (112)
```

7.27 DLL027 Test Response Time Between Consecutive Bursts

A Field Device that is in Burst mode is required to leave a minimum gap of RT2 between the Check Byte of a BACK and the preamble of the next BACK. This allows the appropriate Host to access the network. This test puts the DUT in Burst Mode; waits while the network is monitored; and then takes the DUT out of Burst Mode. The response time between BACKs is validated.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	7.2.3, 7.2.4

Test Procedure

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode

```
CALL CheckReadyForBurst()

SEND Command 109 to put the DUT into Burst Mode
For (5 minutes)
      Record BACK Response Time
      IF (Response Time > STO)
            THEN Test result is FAIL                                  (838)
      ELSE  IF (Response Time > (RT2 + 2 Character Times))
                  THEN Test result is WARNING                         (101)
      END IF
      IF (Response Time < RT2)
            THEN Test result is FAIL                                  (839)
      END IF
END FOR

SEND Command 109 to take the DUT out of Burst Mode
IF (number of BACKs < 539)
      THEN Test result is FAIL                                        (374)
END IF
END TEST
```

7.28 DLL028 BACK Timing with STXs Errors

This Test is only applicable to asynchronous Physical Layers (e.g., FSK, RS-485).

The objective of this test is to confirm that a Burst-Mode Slave properly handles STXs received with errors. Two test cases are included:

- **STXs with gap errors:** The DUT must disregard the partial STX and immediately burst
- **STXs with parity errors:** The DUT must wait for a possible slave ACK (i.e. RT1)

The requests are purposely made to a different Slave device, so that the proper states are exercised.

Note 1: This Test was significantly modified from that found in previous revisions. This test is now harmonized with Gap Error handling requirements. Field Devices have always been required to disregard messages containing a gap error.

Note 2: This Test assumes DLL007 has already been passed by the DUT.

Note 3: RT2 is the maximum amount of time required to detect the immediate transmission of a message (see Annex C of the *Data Link Layer Specification*). As a result, it is used as the time limit in this Test.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.4.1, 7.2.3, 7.2.4

Test Procedure

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode

```
CALL CheckReadyForBurst()
```

Use a Command that the DUT does not think it needs to answer.

```
SET testCmd[] = 0x82, 0xAF, 0xFA, 0x12, 0x34, 0x56, 0x80, 0x20, 0x01, 0x02,
      0x03, ... , 0x1E, 0x1F, 0x20, <check byte>]

SEND Command 109 to put the DUT into Burst Mode
```

The DUT must begin answer within STO of last BACK. This requirement is measured as RT1 after the BACK (BACK-BACK Response Time).

```
FOR (100 iterations)
   FOR (msgLen = [1 – 0x15] )
      SEND <5 preambles>, testCmd[0] – testCmd[msgLen]
      Record BACK-BACK Response Time
      IF (Collision)
         THEN Test result is FAIL                                 (840)
      END IF
      IF (BACK-BACK Response Time > RT1)
         THEN Test result is FAIL                                 (841)
      END IF
   END FOR
```

```
      SEND testCmd with Parity Error in the third Address Byte
      Record Response Time after transmission until next BACK
      IF (Response Time > (RT1+STO))
         THEN Test result is FAIL                                       (842)
      ELSE IF (Response Time > (RT1+RT2))
         THEN PRINT "WARNING: Long STX to BACK response time            (102)
            threatens bus arbitration integrity. BACK should
            immediately follow RT1 timeout"
      END IF
   END FOR

   SEND Command 109 to take the DUT out of Burst Mode

   END TEST
```

7.29 DLL029 Burst Mode Timeout On Other Slave

A Burst-Mode Slave must wait RT1 after a Host Request to another Field Device, thus allowing it to respond. When the other Field Device does not answer, the Burst-Mode Slave must recover the token and issue the BACK. In other words, the DUT must burst immediately after RT1 lapses.

The test puts the DUT into Burst Mode and sends Long and Short Frame Requests to a non-existent Field Device. The DUT should issue another BACK immediately after RT1 (Primary) time elapses from the Check Byte of the STX. The BACK must occur within STO after RT1. However this is not recommended as it can cause arbitration problems under some (albeit infrequent) conditions.

Table 15 Non-Existent Field Device Test Vectors

	Test Vectors (In Hexadecimal Bytes)
1	`FF FF, …, FF FF 82, AF, FA, 12, 34, 56, 80, 20, 01, 02, 03, ... , 1E, 1F, 20, <check byte>`
2	`FF FF, …, FF FF 82, AF, FA, 12, 34, 56, 3, 0, <check byte>`
3	`FF FF, …, FF FF 02, <shortAddress>, 3, 0, <check byte>`

References:

Specification	**Rev.**	**Sections**
Data Link Layer Specification	8.0	7.2.3

Test Procedure

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode

```
CALL CheckReadyForBurst()
```

Use a Command that the DUT does not think it needs to answer. Use the normal preamble length indicated by NUMBER_REQUEST_PREAMBLES

```
SET shortAddress = 0x3F exclusive or'd with POLL_ADDRESS

SEND Command 109 to put the DUT into Burst Mode

FOR (100 iterations)
   FOR (each test vector)
      SEND test vector
      Record Response Time after transmission until next BACK
      IF (Response Time > (STO+RT1))
         THEN Test result is FAIL                     (842 + TEST_VECTOR)
      ELSE IF (Response Time > (RT1+RT2))
         THEN PRINT "WARNING: Long STX to BACK response              (103)
              time threatens bus arbitration integrity. BACK
              should immediately follow RT1 timeout"
      ELSE IF (Response Time < RT1 )
         THEN Test result is FAIL                     (845 + TEST_VECTOR)
      END IF
   END FOR
END FOR

SEND Command 109 to take the DUT out of Burst Mode

If (number of message cycles < 300)
      THEN Test result is FAIL                                      (396)
END IF

END TEST
```

7.30 DLL030 Burst After Response From Other Slave

The DUT should begin a BACK immediately after the Check Byte of an ACK from another Field Device. Under no circumstances can the BACK begin later than STO.

The test puts the DUT into Burst Mode and then simulates message traffic to another Field Device verifying the BACK time after each ACK.

> Note RT2 is the maximum amount of time required to detect the immediate transmission of a message (see Annex C of the *Data Link Layer Specification*). As a result it is used to confirm that the BACK started immediately after the ACK

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	
Command Summary Specification	8.0	10.5

Test Procedure

See if the DUT supports burst-mode

```
CALL CheckReadyForBurst()
```

Use a Command that the DUT does not think it needs to answer.

```
SET testReq[] = 0x82, 0xAF, 0xFA, 0x12, 0x34, 0x56,
      0x03, 0x00, <check byte = 0xA4>]
SET testRsp[] = 0x86, 0xAF, 0xFA, 0x12, 0x34, 0x56,
      0x03, 0x1A, 0x00, 0x40, 0x40, 0xA3, 0x26, 0x44,
      0x7B, 0x42, 0x37, 0xEB, 0x85, 0x01, 0x43, 0xED, 0x80, 0x00,
      0x06, 0x42, 0xFC, 0xA8, 0xF6, 0x33, 0x42, 0x2C, 0xCC, 0xCD,
      <check byte>]

SEND Command 109 to put the DUT into Burst Mode

FOR (100 iterations)

      SEND <5 preambles>, testReq
      SEND <5 preambles>, testRsp
      Record Response Time after transmission until next BACK
      IF (Response Time > STO)
            THEN Test result is FAIL                             (849)
      ELSE IF (Response Time > RT2)
            THEN PRINT "WARNING: Long ACK to BACK response time      (104)
                  threatens bus arbitration integrity. BACK should
                  immediately follow any ACK"
      END IF
END FOR
SEND Command 109 to take the DUT out of Burst Mode

END TEST

Note:  If fewer than 200 BACK's detected
      THEN Test result is FAIL                                   (349)
```

7.31 DLL031 (Reserved)

In previous versions of this document, DLL031 confirmed Field Device operation when two Burst-Mode Slave are on the same network. This condition is not allowed by the Protocol and, thus, no longer tested.

7.32 DLL032 Read Unique Identifier (Command 0)

This test checks the following for Command 0:

1. The device accepts short and long frame formats.
2. The device responds without communications or command specific errors.
3. The first data byte in the response is 254.
4. The number of preambles sent back by the device is in the allowed range.

Table 16 Command 0 Pass / Fail Criteria

Data Item	Requirement	
First Response Data byte	PASS	= 254
Request Preambles Note: DUTs supporting Burst Mode must have request preambles set to 5 to ensure proper bus arbitration.	FAIL FAIL WARNING FAIL	>20 < 5 (HART 6 or later) < 5 (HART 5) < 2 (HART 5)
HART Universal Command Revision	PASS WARNING FAIL FAIL	= 7 < 7 < 5 > 7
Device Rev.	PASS	< 250
Software Rev.	PASS	< 250
Hardware Rev	PASS	< 31
EEPROM Control (Flags Byte)	IF set THEN WARNING	
Response Preambles	WARNING	> 5
Max. Device Variables	PASS	< 240
Device ID	PASS PASS	!= 0xFFFFFF != 0x000000
Device Profile	PASS	Legal Values in Common Table 57

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	
Universal Command Specification	7.0	6.1

Test Procedure

```
Set NUMBER_REQUEST_PREAMBLES to 15
```

The DUT must answer Command 0 with no errors at one valid poll address.

```
pollAddress = 0, numDevices = 0
While ( pollAddress < 63 )
      SEND short frame Command 0 using POLL_ADDRESS = pollAddress
      IF   ( COMMUNICATIONS_ERROR == "No Response" )
            THEN increment pollAddress
      ELSE  IF ( COMMUNICATIONS_ERROR )
            THEN Test result is FAIL                                       (850)
      ELSE  IF ((RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Busy"))
            THEN Test result is FAIL                                       (851)
      ELSE
            increment numDevices
      END IF
END WHILE

IF (numDevices != 1)
      THEN Test result is FAIL                                             (852)
END IF
IF ((UNIV__REVISION == 7) AND (BYTE_COUNT != 24)) OR
  (UNIV__REVISION == 6) AND (BYTE_COUNT != 19)) OR
  (UNIV__REVISION == 5) AND (BYTE_COUNT != 14))
      THEN Test result is FAIL                                             (853)
END IF

FOR (each data field)
      IF (data field value invalid)
            THEN Test result is FAIL                                       (854)
      END IF
      IF (data field value not recommended)
            THEN PRINT warning                                             (105)
      END IF
END FOR

SEND Command 109 with no data bytes
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                             (412)
END IF
IF (RESPONSE_CODE == "Too Few Data Bytes") THEN
      IF ("Request Preambles" != 5)
            THEN Test result is FAIL                                       (413)
      END IF
      IF ((UNIV_REVISION >= 6) && ("Response Preambles" != 5))
            THEN Test result is FAIL                                       (414)
      END IF
END IF

END TEST

Note:  If fewer than 64 Command 0 STX's detected
      THEN Test result is FAIL                                             (348)
```

7.33 DLL033 Write Polling Address (Command 6)

The use of short frame Command 0 to automatically identify the connected Field Devices is fundamental to the Protocol. The test checks that polling addresses can be set correctly by a master. This test checks the following:

- The DUT accepts all valid polling addresses and only responds to the address set.
- The DUT returns Response Code "Invalid Selection" to a request with an invalid Polling Address.
- The DUT returns Response Code "Too Few Data Bytes Received" to a request with zero Request Data Bytes.
- The DUT properly supports the "Loop Current Signaling" property.
- The DUT answers master requests containing a valid Polling Address and only one Request Data Byte.

These requirements are verified using the three test cases below.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	5.3.4
Universal Command Specification	8.0	7.5.2

Test Case A: Test All Polling Addresses

Check DUT accepts all valid Polling Addresses and responds only to the correct one.

```
CALL IdentifyDevice

IF (UNIV__REVISION < 6)
      THEN maxPollAddress = 15
      ELSE maxPollAddress = 63
END IF

CALL VerifyNotWriteProtected()
FOR (pollAddress = [0 to maxPollAddress] )
      SEND Command 6 with ((one data byte) AND (POLL_ADDRESS=pollAddress))
      IF (RESPONSE_CODE != 0)
            THEN Test result is FAIL                                (855)
      ELSE IF (UNIV__REVISION > 5) AND (BYTE_COUNT !=4)
            THEN Test result is FAIL                                (856)
      END IF

      FOR (checkAddress = [0 to maxPollAddress] )
            SEND short frame Command 0 using checkAddress
            IF (checkAddress = pollAddress) THEN
                  IF ( COMMUNICATIONS_ERROR = "No Response" )
                        THEN Test result is FAIL                    (857)
                  END IF
                  IF ( Response returned long frame address)
                        THEN Test result is FAIL                    (859)
                  END IF
            ELSE  IF ( COMMUNICATIONS_ERROR != "No Response" )
                  THEN Test result is FAIL                          (858)
            END IF
      END FOR
END FOR
SEND Command 6 with ((one data byte) AND (POLL_ADDRESS=0))
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                      (398)
END IF
END TEST CASE
```

Test Case B: Test Invalid Data Fields

```
CALL IdentifyDevice
```

Check invalid Polling Address is not accepted

```
IF (UNIV__REVISION < 6)
      THEN maxPollAddress = 15
      ELSE maxPollAddress = 63
END IF
CALL VerifyNotWriteProtected()
SEND Command 6 with POLL_ADDRESS = (maxPollAddress +1)
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                      (399)
END IF
IF (RESPONSE_CODE != "Invalid Selection")
      THEN Test result is FAIL                                      (860)
END IF
```

Check command with too few data bytes is not accepted

```
SEND Command 6 with BYTE_COUNT = 0
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                      (400)
END IF
IF (RESPONSE_CODE != "Too Few Data Bytes" )
      THEN Test result is FAIL                                      (861)
END IF
```

Check command with too many data bytes is accepted

```
SEND Command 6 with BYTE_COUNT = 3, (POLL_ADDRESS=1) AND
                  (LOOP_CURRENT="Disabled") AND (EXTRA_BYTE = 0)
IF (RESPONSE_CODE != 0 )
      THEN Test result is FAIL                                      (862)
END IF
IF UNIV__REVISION > 5 THEN
      IF (BYTE_COUNT != 4)
            THEN Test result is FAIL                                (410)
      END IF
ELSE
      IF (BYTE_COUNT != 3)
            THEN Test result is FAIL                                (411)
      END IF
END IF
```

Set Polling Address back to 0

```
SEND Command 6 with POLL_ADDRESS=0
IF (RESPONSE_CODE != 0)
      THEN Test result is FAIL                                      (863)
END IF
END TEST CASE
```

Test Case C: Test Loop Current Signaling

The DUT must enable and disable loop current signaling correctly.

```
CALL IdentifyDevice
CALL VerifyNotWriteProtected()
IF UNIV__REVISION > 5 THEN
      SEND Command 6 with ((POLL_ADDRESS=1) AND (LOOP_CURRENT="Enabled"))
      RECORD Command 6 response
      IF ((RESPONSE_CODE != 0) AND NOT WIRELESS )
            THEN Test result is FAIL                                  (865)
      END IF
      IF ((DEVICE_STATUS == "Loop Current Fixed") AND NOT WIRELESS )
            THEN Test result is FAIL                                  (866)
      END IF
      SEND Command 7
      IF (RESPONSE_CODE != 0)
            THEN Test result is FAIL                                  (867)
      END IF
      IF (Command 6 response != Command 7 response)
            THEN Test result is FAIL                                  (864)
      END IF

      SEND Command 6 with ((POLL_ADDRESS=0) AND (LOOP_CURRENT="Disabled"))
      IF (RESPONSE_CODE != 0)
            THEN Test result is FAIL                                  (868)
      END IF
      IF (DEVICE_STATUS != "Loop Current Fixed")
            THEN Test result is FAIL                                  (869)
      END IF
END IF
```

Test Backward Compatibility

```
SEND Command 6 with ((one data byte) AND (POLL_ADDRESS = 1))
IF (RESPONSE_CODE != 0)
      THEN Test result is FAIL                                        (870)
END IF
IF (DEVICE_STATUS != "Loop Current Fixed")
      THEN Test result is FAIL                                        (871)
END IF
IF UNIV__REVISION > 5 THEN
      IF (BYTE_COUNT != 4)
            THEN Test result is FAIL                                  (872)
      END IF
ELSE
      IF (BYTE_COUNT != 3)
            THEN Test result is FAIL                                  (903)
      END IF
ENDIF
SEND Command 6 with ((one data byte) AND (POLL_ADDRESS = 0))
IF (RESPONSE_CODE != 0)
      THEN Test result is FAIL                                        (873)
END IF
IF ((DEVICE_STATUS == "Loop Current Fixed") AND NOT WIRELESS)
      THEN Test result is FAIL                                        (874)
END IF
END TEST CASE
```

7.34 DLL034 Read Unique Identifier with Tag (Command 11)

Command 11 is one of two commands that support Broadcast Addresses. A Broadcast Address consists of all zeroes in place of the Slave Address. In Command 11, the 6 byte Tag (found in the Request Data) is used as the Field Device's address. As a result of the addressing issues associated with this command, its implementation is validated in this Test Specification

Command 11 returns the same Response Data as Command 0.

The following test cases are used.

Table 17 Command 11 Test Cases

Case	Address Format	Request Data	Response Expected
1	Broadcast Address	Valid TAG	Yes
2	Broadcast Address	Invalid TAG	No
3	Broadcast Address	Too Few Data Bytes	No
4	Broadcast Address	Too Many Data Bytes (with valid TAG)	Yes
5	DUT Address	Valid TAG	Yes
6	DUT Address	Invalid TAG	No
7	DUT Address	Too Few Data Bytes	No
8	DUT Address	Too Many Data Bytes (with valid TAG)	Yes

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	
Universal Command Specification	6.0	6.11
Command Summary Specification	8.0	10.1

Test Procedure

```
CALL IdentifyDevice

SEND Command 0
IF ( COMMUNICATIONS_ERROR OR (RESPONSE_CODE != 0))
      THEN Test result is FAIL                                          (250)
END IF
Record Command 0 response

SEND Command 13
IF ( COMMUNICATIONS_ERROR OR (RESPONSE_CODE != 0))
      THEN Test result is FAIL                                          (251)
END IF

FOR each TEST_CASE
      SEND Command 11

      SWITCH on (TEST_CASE)
      CASE TEST_CASE = [1, 4, 5, or 8]
            IF ( COMMUNICATIONS_ERROR OR (RESPONSE_CODE != 0))
                  THEN Test result is FAIL                (255 + TEST_CASE)
            END IF
            IF ( Command 11 response != Command 0 Response))
                  THEN Test result is FAIL                              (252)
            END IF
      CASE TEST_CASE = [2, 3, 6, or 7]
            IF ( COMMUNICATIONS_ERROR != "No Response")
                  THEN Test result is FAIL                (255 + TEST_CASE)
            END IF
      END SWITCH
      CALL CheckDeviceAlive
END FOR

END TEST
```

7.35 DLL035 Write Number Of Response Preambles (Command 59)

This test checks the following for Command 59:

1. The DUT accepts Number of Response preambles between its defined minimum and maximum limits only.
2. The DUT responds with the number of preambles set by Command 59.

Test assumes that one Preamble in response from device may be lost in the modem.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	
Common Practice Command Specification	8.0	7.27

Test Procedure

```
CALL IdentifyDevice
```

Does the DUT support Command 59?

```
SEND Command 59 with no data bytes
IF (RESPONSE_CODE = "Command not Implemented")
      THEN Abort Test (i.e., test is not applicable to this device)   (111)
END IF
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                        (875)
END IF
IF (RESPONSE CODE != "Too Few Data Bytes")
      THEN Test result is FAIL                                        (876)
END IF
CALL VerifyNotWriteProtected()
```

Check response to Command 59 with two data bytes

```
SEND Command 59 with two data bytes and PreambleCnt = 5
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                        (877)
END IF
IF (RESPONSE CODE != 0 )
      THEN Test result is FAIL                                        (878)
END IF
```

Verify that the DUT is sending the right number of preamble characters

```
CheckNoPreambles ( 5, 879 )                                       (879-881)
```

Test against legal and illegal preamble counts

```
FOR PreambleCnt = [1, 5, 10, 18, 21]
      SEND Command 59 with PreambleCnt
      IF (COMMUNICATIONS_ERROR)
            THEN Test result is FAIL                                  (882)
      END IF
      SWITCH on (PreambleCnt)
      CASE PreambleCnt = 1
            IF (RESPONSE_CODE != "Passed Parameter Too Small")
                  IF (RESPONSE_CODE == "Set To Nearest Possible Value")
                        IF (returned NUMBER_RESPONSE_PREAMBLES != 5)
                              THEN Test result is FAIL                (888)
                        END IF
                        IF ( Byte Count is not 2 )
                              THEN Test result is FAIL                (280)
                        END IF
                  ELSE
                        Test result is FAIL                           (883)
                  END IF
            ELSE
                  IF ( Byte Count is not 2 )
                        THEN Test result is FAIL                      (281)
                  END IF
            END IF
            CheckNoPreambles ( 5, 283 )                           (283-285)
      CASE PreambleCnt = [5, 10, or 18]
            IF (RESPONSE_CODE != 0)
                  THEN Test result is FAIL                  (884, 885, 886)
            END IF

            IF UNIV__REVISION > 5
                  SEND Command 0
                  IF (COMMUNICATIONS_ERROR)
                        THEN Test result is FAIL                      (357)
                  END IF
                  IF ( NUMBER_RESPONSE_PREAMBLES from Command 0 !=
                    PreambleCnt)
                        THEN Test result is FAIL                      (358)
                  END IF
            END IF
            CheckNoPreambles ( PreambleCnt, 286 )                 (286-288)
      CASE PreambleCnt = 21
            IF (RESPONSE_CODE != "Passed Parameter Too Large")
                  IF (RESPONSE_CODE == "Set To Nearest Possible Value")
                        IF (returned NUMBER_RESPONSE_PREAMBLES != 20)
                              THEN Test result is FAIL                (889)
                        ELSE
                              Test result is FAIL                     (887)
                        END IF
                  CheckNoPreambles ( 20, 289 )                    (289-291)
            ELSE
                  IF ( Byte Count is not 2 )
                        THEN Test result is FAIL                      (282)
                  END IF
                  CheckNoPreambles ( 18, 292 )                    (292-294)
            END IF
      END SWITCH
END FOR

SEND Command 59 with PreambleCnt = 5
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                        (359)
END IF
```

CheckNoPreambles (noPre, FAILUREPOINT)

The following procedure is used to validate that the number of preambles are being set correctly.

```
PROCEDURE CheckNoPreambles ( noPre, FAILUREPOINT )
FOR (100 iterations)
      Send Command 3
Record Number Response Preambles
IF (No Response)
      THEN Test result is FAIL                               (FAILUREPOINT)
END IF
IF (RESPONSE_CODE != [0 or 8])
      THEN Test result is FAIL                             (FAILUREPOINT+1)
END IF
IF (Response Preambles > noPre)
      THEN Test result is FAIL                             (FAILUREPOINT+2)
END IF
END FOR

Note:  If fewer than 100 STX's detected
      THEN Test result is FAIL                                        (354)

PROCEDURE END

END TEST
```

7.36 DLL036 Write Burst Mode Command Number (Command 108)

This test checks the following for Command 108:

1. Device accepts Burst Mode Command Numbers 1, 2, 3 and 9 and responds with new command in bursts.
2. Device rejects Burst Mode Command Number of 4 (reserved command).
3. Device responds with Response Code 5 when too few Data Bytes sent in request (HART 5 and 6).
4. HART 7 device only responds with Response Code 5 when no data bytes are received. Otherwise, the HART 7 device operates in backward compatibility mode.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	
Common Practice Command Specification	8.0	7.53

Test Case A: Verify Mandatory Burst Commands

Check that the DUT accepts all commands that must be supported in burst mode (e.g., 1, 2, 3, 9, 48 and 33)

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode

```
CALL CheckReadyForAnyBurst

IF (UNIV__REVISION > 6) THEN
   burstCommand = [1, 2, 3, 4, 9, 33, 48]
ELSE IF (UNIV__REVISION > 5)
   THEN burstCommand = [1, 2, 3, 4, 9, 33]
   ELSE burstCommand = [1, 2, 3, 4, 33]
END IF
```

See if the DUT supports burst-mode and is not in write protect.

```
SEND Command 108 (with command = 255)
IF (COMMUNICATIONS_ERROR)
   THEN Test result is FAIL                                    (245)
END IF
IF (RESPONSE_CODE != "Invalid Selection")
   THEN Test Result is FAIL                                    (246)
END IF
```

Check the commands the DUT supports with Command 108

```
FOR each burstCommand
```

All devices must accept a one-byte request for burst command number.

```
   SEND Command 108 with 8-bit burstCommand only
   IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                 (892)
   END IF

   SWITCH on (burstCommand)
   CASE burstCommand = [1, 2, 3, 48, or 9]
      IF (RESPONSE_CODE != 0)
         THEN Test result is FAIL                         (893 + Iteration)
      END IF
   CASE burstCommand = 4
      IF (RESPONSE_CODE != "Invalid Selection")
         THEN Test result is FAIL                         (893 + Iteration)
      END IF
   CASE burstCommand = 33
      IF (RESPONSE_CODE = "Invalid Selection ") THEN
         PRINT "WARNING: Command 108 is recommended to
               support Command 33"                                    (106)
      ELSE IF (RESPONSE_CODE != 0)
         THEN Test result is FAIL                         (893 + Iteration)
      END IF
   END SWITCH
```

Read the setting back to verify that the command number changed.

```
   SEND Command 105 with no data bytes
   IF UNIV__REVISION > 5) THEN
      SET bc = 8
      IF UNIV__REVISION > 6) THEN
         SET bc = 31
      END IF
      CALL VerifyResponseAndByteCount ( 0, bc )

      CmdCheck = burstCommand
      IF burstCommand = 4 THEN
         SET CmdCheck = 3
      END IF
      IF "Command Number Expansion Flag" != CmdCheck
         THEN Test Result is FAIL                                     (905)
      END IF

      IF UNIV__REVISION > 6) THEN
         IF "Extended Command Number" != CmdCheck
            THEN Test Result is FAIL                                  (906)
         END IF
         IF "Burst Message" != 0
            THEN Test Result is FAIL                                  (907)
         END IF
      END IF

   END IF


END FOR
END TEST CASE
```

Test Case B: BACK Changes Command Response In Burst Mode

Checks that the DUT changes bursting command response while in burst-mode and does so within one BACK after receiving Command 108.

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode and is not in write protect.

```
CALL CheckReadyForAnyBurst()

SEND Command 109 to put the DUT into Burst Mode

CALL WriteBurstCommand ( 1, 0 )
```

Verify that the DUT switches burst response within one BACK

```
MONITOR BACKs
IF (BACK != Command 1)
   THEN Test Result is FAIL                                              (200)
END IF

CALL WriteBurstCommand ( 3, 0 )

Wait one BACK
IF (second BACK != Command 2)
   THEN Test Result is FAIL                                              (202)
END IF

SEND Command 109 to take the DUT out of Burst Mode

END TEST CASE
```

Test Case C: Support for 16-Bit Burst Command Numbers

Verifies proper DUT operation using 16-Bit Burst Command numbers .

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode and is not in write protect.

```
IF UNIV_REVISION < 7 THEN
   PRINT "WARNING: Implementation of HART 7 is strongly recommended."
   Abort Test                                                            (325)
END IF

CALL CheckReadyForAnyBurst()

CALL WriteBurstCommand ( 3, 0 )
CALL WriteBurstCommand ( 48, 0 )
CALL WriteBurstCommand ( 9, 0 )

END TEST CASE
```

Test Case D: Support for Multiple Burst Messages

Verifies that the DUT support required number of Burst Messages. Field Devices must support 3, Adapters must support 5.

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode and is not in write protect.

```
IF UNIV_REVISION < 7 THEN
   PRINT "WARNING: Implementation of HART 7 is strongly recommended."
   Abort Test                                                              (330)
END IF
CALL CheckReadyForAnyBurst()

SEND Command 105 with bmsg = 0
CALL VerifyResponseAndByteCount ( 0, 31 )

IF MAX_BURST_MSGS < 3
   THEN Test Result is FAIL                                                (331)
END IF
IF DEVICE_PROFILE_CODE is 141 or 142
   IF MAX_BURST_MSGS < 5
      THEN Test Result is FAIL                                             (332)
   END IF
END IF
```

Set command numbers into all the burst messages.

```
burstCommand = [1, 2, 3, 9, 48, 1, 2, 3, 9, 48, 1, 2, 3, 9, 48]
FOR bmsg = 0 to (MAX_BURST_MSGS-1)
   CALL WriteBurstCommand ( burstCommand[bmsg], bmsg )
END FOR
SEND Command 108 with Cmd = 1, BURST_MESSAGE = MAX_BURST_MSGS
IF
IF (RESPONSE_CODE != "Invalid Burst Message")
   THEN Test result is FAIL                                                (333)
END IF
IF (BYTE_COUNT != 2)
   THEN Test result is FAIL                                                (334)
END IF
```

Read back the burst messages to make sure then are still set correctly.

```
FOR bmsg = 0 to (MAX_BURST_MSGS-1)
   SEND Command 105 with bmsg
   CALL VerifyResponseAndByteCount ( 0, 31 )
   IF "Command Number Expansion Flag" != 31
      THEN Test Result is FAIL                                             (335)
   END IF
   IF "Extended Command Number" != burstCommand[bmsg]
      THEN Test Result is FAIL                                             (336)
   END IF
   IF "Burst Message" != bmsg
      THEN Test Result is FAIL                                             (337)
   END IF

END FOR

END TEST CASE
```

WriteBurstCommand (cmd, bmsg)

Writes a Burst Command number and verifies it using Command 105

```
PROCEDURE WriteBurstCommand (cmd, bmsg)

SET bc = 5
IF UNIV_REVISION < 7 THEN
   SET bc = 3
   SET cmd = cmd right shifted 8 bits.
END IF
```

HART 5 and 6 will ignore the extra 16 bits

```
SEND Command 108 with cmd, bmsg
CALL VerifyResponseAndByteCount ( 0, bc )
```

Read back the settings using Command 105

```
IF UNIV_REVISION > 5 THEN
   SET bc = 31
   SET CmdExp = 31
   IF UNIV_REVISION < 7 THEN
      SET bc = 8
      SET CmdExp = MS 8-Bits of cmd
   END IF

   SEND Command 105 with bmsg
   CALL VerifyResponseAndByteCount ( 0, bc )
   IF "Command Number Expansion Flag" != CmdExp
      THEN Test Result is FAIL                                    (520)
   END IF

   IF UNIV_REVISION > 6 THEN
      IF "Extended Command Number" != cmd
         THEN Test Result is FAIL                                 (521)
      END IF
      IF "Burst Message" != bmsg
         THEN Test Result is FAIL                                 (522)
      END IF
   END IF
END IF

END PROCEDURE
```

7.37 DLL037 Burst Mode Control (Command 109)

This test checks the following for Command 109:

- DUT enters and exits Burst Mode with correct Burst Mode Select Codes.
- DUT does not enter and exit Burst Mode with invalid Burst Mode Select Code.
- DUT responds with Response Code 5 when too few Data Bytes sent in request.

If the device supports more than one Data Link Layer, the operations of the token-passing Data Link Layer are verified. TDMA devices with maintenance ports are not required to support burst mode on the non-TDMA data link layer.

Network Type	**Codes**	**Non-TDMA**	**TDMA**
Non-TDMA	0,1	BACK	No BACK
TDMA without support of burst on non-TDMA	0,2	No BACK	BACK
TDMA and non-TDMA burst-mode support	0,1,2,3	BACK	BACK

Unless otherwise noted, these test must be repeated for each Data-Link supported by the DUT.

References:

Specification	**Rev.**	**Sections**
Data Link Layer Specification	8.0	
Common Practice Command Specification	8.0	7.52

Test Case A: Enable/Disable Burst Mode.

Check basic burst mode operation (i.e., can we turn burst mode on and off).

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode. CheckReadyForBurst check the too few data bytes case. It also sets the burst command to 3.

```
CheckReadyForAnyBurst()
```

Put into Burst Mode

```
SEND Command 109 to put the DUT into Burst Mode

MONITOR BACKs
IF (no BACKS detected)
      THEN Test Result is FAIL                                    (206)
END IF
```

Check Response Code to invalid Burst Mode Select Code

```
SEND Command 109 with Burst Mode Control = 255
IF (RESPONSE_CODE != "Invalid Selection )
      THEN Test Result is FAIL                                    (207)
END IF
IF (no BACKS detected)
      THEN Test Result is FAIL                                    (208)
END IF
```

Take DUT out of burst mode

```
SEND Command 109 to take the DUT out of Burst Mode
IF (COMMUNICATIONS_ERROR) OR (RESPONSE_CODE != 0 )
      THEN Test Result is FAIL                                        (209)
END IF

IF (any BACKS detected)
      THEN Test Result is FAIL                                        (210)
END IF
```

Check Response Code to invalid Burst Mode Select Code

```
SEND Command 109 with Burst Mode Control = 255
IF (RESPONSE_CODE != "Invalid Selection )
      THEN Test Result is FAIL                                        (211)
END IF

IF (any BACKS detected)
      THEN Test Result is FAIL                                        (212)
END IF

END TEST CASE
```

Test Case B: Verify Burst mode through power cycles, self test, reset.

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode.

```
CALL CheckReadyForAnyBurst

SEND Command 109 to put the DUT into Burst Mode

MONITOR BACKs
IF (no BACKS detected)
      THEN Test Result is FAIL                                        (910)
END IF
```

Check that the DUT supports Burst Mode through power cycle

Prompt the user to power down the devices.

```
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")
```

Check that the DUT supports Burst Mode through Device Reset

```
IF UNIV_REVISION > 5
   SEND Command 105 (with no data bytes) to read Burst Mode Control Code
   IF UNIV_REVISION > 6 THEN
      CALL VerifyResponseAndByteCount ( 0, 8 )
   ELSE
      CALL VerifyResponseAndByteCount ( 0, 31 )
   ENDIF
   IF Burst Mode Control != Value sent in Command 109
      THEN Test result is FAIL                                       (911)
   END IF
ENDIF

MONITOR BACKs
IF (no BACKS detected)
      THEN Test Result is FAIL                                       (912)
END IF
```

Check that the DUT supports Burst Mode through Device Reset

```
SEND Command 42
IF (RESPONSE_CODE == "Command Not Implemented") THEN
   IF DEVICE_PROFILE_CODE > 128
         THEN Test Result is FAIL                                    (913)
   END IF
ELSE
   IF (RESPONSE_CODE != "Success")
      THEN Test result is FAIL                                       (914)
   END IF

   DO
      SEND Command 0
   WHILE (COMMUNICATIONS_ERROR == "No Response")

   MONITOR BACKs
   IF (no BACKS detected)
      THEN Test Result is FAIL                                       (915)
   END IF

END IF
```

Check that the DUT supports Burst Mode through Self-test

```
SEND Command 41
IF (RESPONSE_CODE == "Command Not Implemented") THEN
   IF DEVICE_PROFILE_CODE > 128
         THEN Test Result is FAIL                                    (916)
   END IF
ELSE
   IF (RESPONSE_CODE != "Success")
      THEN Test result is FAIL                                       (917)
   END IF

   DO
      SEND Command 0
   WHILE (COMMUNICATIONS_ERROR == "No Response")
```

```
      MONITOR BACKs
      IF (no BACKS detected)
         THEN Test Result is FAIL                                            (918)
      END IF

   END IF

   SEND Command 109 to take the DUT out of Burst Mode

   END TEST CASE
```

Test Case C: Verify Supported Burst Mode Control Codes

```
   CALL IdentifyDevice
```

See if the DUT supports burst-mode.

```
   CALL CheckReadyForAnyBurst

   IF UNIV_REVISION > 6 THEN

      SET TPMode = FALSE
      SET TDMAMode = TRUE

      SEND Command 109 with 1 ("Token-Passing Burst Mode"); Burst Message = 0
      IF (RESPONSE_CODE != "Invalid Selection")
         SET TPMode = TRUE
      END IF

      SEND Command 109 with 2 ("TDMA Burst Mode"); Burst Message = 0
      IF (RESPONSE_CODE == "Invalid Selection")
         SET TDMAMode = FALSE
         IF PROFILE > 128
            THEN Test Result is FAIL                                         (925)
         END IF
      END IF

      SEND Command 109 with 3; Burst Message = 0
      IF (RESPONSE_CODE == "Invalid Selection")
         IF TPMode AND TDMAMode
            THEN Test Result is FAIL                                         (926)
         END IF
      END IF

      IF (RESPONSE_CODE != "Invalid Selection") AND (PROFILE < 128)
         THEN Test Result is FAIL                                            (927)
      END IF

      Record Modes Supported in Test Report

      SEND Command 109 to take the DUT out of Burst Mode (Burst Message = 0)

   END IF

   END TEST CASE
```

Test Case D: Support for Multiple Burst Messages.

Verifies DUT support required number of Burst Messages. Field Devices must suport 3, Adapters must support 5.

```
CALL IdentifyDevice
```

See if the DUT supports burst-mode.

```
IF UNIV_REVISION < 7 THEN
   PRINT "WARNING: Implementation of HART 7 is strongly recommended."
   Abort Test                                                           (930)
END IF
CALL CheckReadyForAnyBurst()

SEND Command 105 with bmsg = 0
CALL VerifyResponseAndByteCount ( 0, 31 )

IF MAX_BURST_MSGS < 3
   THEN Test Result is FAIL                                             (931)
END IF
IF DEVICE_PROFILE_CODE is 141 or 142
   IF MAX_BURST_MSGS < 5
      THEN Test Result is FAIL                                          (932)
   END IF
END IF
```

Turn each burst message on and off

```
FOR bmsg = 0 to (MAX_BURST_MSGS-1)
   SEND Command 109 to put bmsg in burst mode
   MONITOR BACKs
   IF (no BACKS detected)
      THEN Test Result is FAIL                                          (933)
   END IF

   SEND Command 109 to take bmsg out of burst mode
   MONITOR BACKs
   IF (BACKS detected)
      THEN Test Result is FAIL                                          (934)
   END IF

   FOR bmsg1 = 0 to (MAX_BURST_MSGS-1)
      SEND Command 105 with bmsg1
      CALL VerifyResponseAndByteCount ( 0, 31 )

      IF (bmsg1 = bmsg) AND (Burst Message Control == 0)
         THEN Test Result is FAIL                                       (935)
      ELSE IF Burst Message Control != 0
         THEN Test Result is FAIL                                       (936)
      END IF

   END FOR
END FOR
```

Read back the burst messages to make sure then are still set correctly.

```
SEND Command 109 to put bmsg = 1 in burst mode
MONITOR BACKs
IF (no BACKS detected)
   THEN Test Result is FAIL                                          (937)
END IF

SEND Command 109 to take bmsg = 2 out of burst mode
MONITOR BACKs
IF (no BACKS detected)
   THEN Test Result is FAIL                                          (938)
END IF


SEND Command 109 to take bmsg = 1 out of burst mode
MONITOR BACKs
IF (BACKS detected)
   THEN Test Result is FAIL                                          (939)
END IF

END TEST CASE
```

7.38 DLL038 Read Unique Identifier With Long Tag (Command 21)

Command 21 is one of two Universal Commands that support Broadcast Addresses. A Broadcast Address consists of all zeroes in place of the Slave Address. In Command 21, the 32 byte Long Tag (found in the Request Data) is used as the Field Device's address. As a result of the addressing issues associated with this command, its implementation is validated in this Test Specification.

Command 21 returns the same Response Data as Command 0.

The following test cases are used.

Table 18 Command 21 Test Cases

Case	Address Format	Request Data	Response Expected
1	Broadcast Address	Valid TAG	Yes
2	Broadcast Address	Invalid TAG	No
3	Broadcast Address	Too Few Data Bytes	No
4	Broadcast Address	Too Many Data Bytes (with valid TAG)	Yes
5	DUT Address	Valid TAG	Yes
6	DUT Address	Invalid TAG	No
7	DUT Address	Too Few Data Bytes	No
8	DUT Address	Too Many Data Bytes (with valid TAG)	Yes

References:

Specification	Rev.	Sections
Universal Command Specification	6.0	5.4, 6.21

Test Procedure

```
CALL IdentifyDevice
IF UNIV__REVISION < 6
      THEN Abort Test (i.e., test is not applicable to this device)
END IF

SEND Command 0
IF ( COMMUNICATIONS_ERROR OR (RESPONSE_CODE != 0))
      THEN Test result is FAIL                                        (213)
END IF
Record Command 0 response

SEND Command 20
IF ( COMMUNICATIONS_ERROR OR (RESPONSE_CODE != 0))
      THEN Test result is FAIL                                        (214)
END IF

FOR each TEST_CASE
      SEND Command 21

      SWITCH on (TEST_CASE)
      CASE TEST_CASE = [1, or 5]
            IF ( COMMUNICATIONS_ERROR OR (RESPONSE_CODE != 0))
                  THEN Test result is FAIL                (216 + TEST_CASE)
            END IF
            IF ( Command 21 response != Command 0 Response))
                  THEN Test result is FAIL                            (215)
            END IF
      CASE TEST_CASE = [4, or 8]
            IF ( COMMUNICATIONS_ERROR != "Buffer Overflow")
                  IF (COMMUNICATIONS_ERROR OR (RESPONSE_CODE != 0))
                        THEN Test result is FAIL          (216 + TEST_CASE)
                  END IF
                  IF ( Command 21 response != Command 0 Response))
                        THEN Test result is FAIL                      (216)
                  END IF
            END IF
      CASE TEST_CASE = [2, 3, 6, or 7]
            IF ( COMMUNICATIONS_ERROR != "No Response")
                  THEN Test result is FAIL                (216 + TEST_CASE)
            END IF
      END SWITCH
      CALL CheckDeviceAlive
END FOR

END TEST
```

7.39 DLL039 Slave Time-Out Stress Test

The Data Link Layer Specification requires any message addressed to the slave device to be answered within STO. Test DLL024 verifies basic conformance to this requirement. This test is designed to confirm compliance over a longer test interval and determine the stability of the DUT's Data Link Layer.

Since this test is performed under "ideal conditions" (i.e., a noise free laboratory environment), the Data Link Layer Specifications requirements are that the slave device answer every message. However this test is more forgiving. This test allows a device to not answer 1 message in 10^6. A warning is generated if the error rate is as low as 1 message in 10^5 (about 2 failures per day) and the device is still considered to comply.

This tests DUT operation with a master acquiring secondary variable data using Command 9 (or Command 3 if the slave is HART 5).

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	

Test Case A: Verify Cyclical Data Access

Warning: This test can take more than 2 weeks to execute.

```
CALL IdentifyDevice
Msg_Cnt = 0, Retry_Cnt = 0, Total_Errs = 0
IF UNIV__REVISION < 6 THEN
      Command = 3
ELSE
      Command = 9 (with slots 0, 1, 2, and 3)
END IF
maintenancePort = FALSE

PROMPT: Is the test system connected to a maintenance port?
IF Yes THEN
      maintenancePort = TRUE
END IF
IF maintenancePort
      THEN Max_msg = 20,000
ELSE
      Max_msg = 2,000,000
END IF
WHILE Msg_Cnt < Max_msg
      SEND Command
      Increment Msg_Cnt
      IF (COMMUNICATIONS_ERROR) OR
        ( (RESPONSE_CODE != 0) AND (RESPONSE_CODE != 8) ) THEN
            Increment Total_Errs, Retry_Cnt
            IF (Retry_Cnt > 3)
                  THEN Test result is FAIL                          (225)
            END IF
      ELSE
            Retry_Cnt = 0
      END IF
      IF (number of response preamble bytes > 20)
            THEN Print "WARNING: Too long of a preamble detected"      (107)
      END IF

      IF (RESPONSE_CODE = "Busy") THEN
            IF (UNIV_COMMAND_REVISION > 5)
                  THEN Test result is FAIL                            (226)
            END IF
            SEND Short Frame Command 0
            Increment Msg_Cnt
            IF (No Response) OR (RESPONSE_CODE != 0)
                  THEN Test result is FAIL                            (227)
            END IF
      END IF
END WHILE

IF (Total_Errs > 2)
      IF (Total_Errs > 20)  THEN
            Test result is FAIL                                       (228)
      ELSE
            PRINT "WARNING: <Total_Errs> Errors Detected,             (108)
                  Excessive Communications Errors should be corrected"
      END IF
END IF

END TEST CASE
```

Test Case B: Verify Write Commands

This is a brief stress test using write commands. This test case confirms adherence to STO response time requirements even when a write command is sent. The writes in this test are distributed across the Command 17 (Write Message), 18 (Write Tag, Descriptor and Date) and 19 (Write Final Assembly Number).

```
CALL IdentifyDevice
CALL VerifyNotWriteProtected()

IF UNIV__REVISION < 6 THEN
   ReadCommand = 3
ELSE
   ReadCommand = 9 (with slots 0, 1, and 2)
END IF

SEND Command 12 to read msg0
IF (COMMUNICATION_ERROR) OR (RESPONSE_CODE != 0)
   THEN Test result is FAIL                                           (310)
END IF

SEND Command 13 to read tag0, desc0, date0
IF (COMMUNICATION_ERROR) OR (RESPONSE_CODE != 0)
   THEN Test result is FAIL                                           (311)
END IF

SEND Command 16 to read fan0
IF (COMMUNICATION_ERROR) OR (RESPONSE_CODE != 0)
   THEN Test result is FAIL                                           (312)
END IF
```

```
SET all bytes of msg1, tag1, desc1, fan1 to zero.
SET date1 to 01 Jan, 1900.
WHILE Msg_Cnt < 200
   FOR WriteCommand = [17, 18, and 19]
      Increment Msg_Cnt
      DO
         SEND WriteCommand
         IF (COMMUNICATIONS_ERROR)
            THEN Test result is FAIL                                  (313)
         END IF
         IF (RESPONSE_CODE != 0)
            IF (RESPONSE_CODE == "Busy") THEN
               SEND Short Frame Command 0
               IF (No Response) OR (RESPONSE_CODE != 0)
                  THEN Test result is FAIL                            (314)
               END IF
            ELSE
               Test result is FAIL                                    (315)
            END IF
         END IF
      WHILE (WriteCommand's RESPONSE_CODE == "Busy")

      SEND ReadCommand
      Increment Msg_Cnt
      IF (ReadCommand's RESPONSE_CODE == "Busy") THEN
         IF (UNIV_COMMAND_REVISION > 5)
            THEN Test result is FAIL                                  (316)
         END IF
         SEND Short Frame Command 0
         IF (COMMUNICATION_ERROR) OR (RESPONSE_CODE != 0)
            THEN Test result is FAIL                                  (317)
         END IF
         IF (number of response preamble bytes > 20)
            THEN Print "WARNING: Too long of a preamble detected"     (115)
         END IF
      END IF
   END FOR
   INCREMENT each byte of msg1, tag1, desc1, fan1.
   INCREMENT year in date1.
END WHILE

SEND Command 17 to write msg0
IF (COMMUNICATION_ERROR) OR (RESPONSE_CODE != 0)
   THEN Test result is FAIL                                           (319)
END IF

SEND Command 18 to write tag0, desc0, date0
IF (COMMUNICATION_ERROR) OR (RESPONSE_CODE != 0)
   THEN Test result is FAIL                                           (320)
END IF

SEND Command 19 to write fan0
IF (COMMUNICATION_ERROR) OR (RESPONSE_CODE != 0)
   THEN Test result is FAIL                                           (321)
END IF

END TEST CASE
```

7.40 DLL040 Unique Address Test

Each device must have a unique device ID. This multi-drop test ensures that the unique device ID is functioning properly. The test also checks that the unique device ID is stored in non-volatile memory.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	

Test Procedure

```
Prompt the user to place two production slave devices in multi-drop mode at
      different polling address on the network and cycle the power on the
      devices.  The poll addresses must be both less than 16.

Set NUMBER_REQUEST_PREAMBLES in request to 15
Device_Number = 0
FOR (Poll_Address = 0 to 15)

   SEND Short Frame Command 0 to identify the slave device
   IF (COMMUNICATIONS_ERROR != "No Response")
      IF (COMMUNICATIONS_ERROR)
         THEN Test result is FAIL                                     (230)
      END IF

      IF UNIV_REVISION < 5
         THEN Abort Test (i.e., test is not applicable)
      END IF

      IF (DEVICE_STATUS != "Cold Start")
         THEN Test result is FAIL                                     (231)
      END IF

      Device_Info[Device_Number] = data from this response
      Increment Device_Number

   END IF
END FOR

IF Device_Number < 2
   THEN Test result is FAIL                                           (232)
END IF

Compare Device_Info
IF (Any Manfacturer_IDs different)
   OR (Any Device_Types different)
   OR (Any Device_IDs identical)
      THEN Test result is FAIL                                        (233)
END IF
```

Prompt the user to power down the devices.

```
PRINT: "Please power down devices, including removal of any batteries (if
   applicable)."
```

Verify that the devices are powered down.

```
FOR (Poll_Address = 0 to 15)
   SEND Command 0
   IF (COMMUNICATIONS_ERROR != "No Response")
      THEN Test result is FAIL                                  (270)
   END IF
END FOR
```

Prompt the user to power up the devices.

```
PRINT: "Please re-apply power to devices, including connecting any
   batteries (if applicable)."
```

First command after a Device Reset, Cold Start must be set.

```
FOR (Poll_Address = 0 to 15)
   SEND Short Frame Command 0 to identify the slave device
   IF (COMMUNICATIONS_ERROR != "No Response")
      IF (COMMUNICATIONS_ERROR)
         THEN Test result is FAIL                               (271)
      END IF

      IF (DEVICE_STATUS != "Cold Start")
         THEN Test Result is FAIL                               (272)
      END IF

      Device_Info2[Device_Number] = data from this response
      Increment Device_Number2
   END IF
END FOR
```

Verify that the same number of devices are found as before.

```
IF Device_Number2 != Device_Number
   THEN Test result is FAIL                                     (273)
END IF
```

Verify no device info has changed across power cycle (e.g., revision info, config changed counters, device IDs, etc.)

```
FOR (devNo = 0 to Device_Number)
   IF (Device_Info[Device_Number] != _Device_Info2[Device_Number])
      THEN Test result is FAIL                                  (274)
   END IF
END FOR

END TEST
```

7.41 DLL041 Framing Successive Messages

A common slave programming shortcut waits for the line to go idle, then watches for a carrier. Devices using this shortcut will miss a frame that closely follows another. In this test, the master will place three frames on the network as a block (i.e., with no gaps or carrier transitions):

- A transaction to another non-existent transmitter
- A contrived response from the non-existent transmitter
- A command to the DUT

By placing these three frames on the network, the host simulates heavy network traffic where other devices are framing each message correctly and do not wait for loss of carrier to transmit their messages. If the DUT waits for the line to go idle to begin framing the next message, it will miss the third message in the block and fail to respond.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	
Common Practice Command Specification	8.0	7.52

Test Procedure

```
CALL IdentifyDevice
```

Send a fake command 1 request, fake command 1 response, and a command 1 request to the DUT - all as a single transaction (i.e., with no gaps, no carrier transitions, etc)

```
Issue one  network transaction with three consecutive messages:
Primary Master Command 1 to other device
Command 1 response from other device
Secondary Master Command 2 to DUT

IF (No Response)
      THEN Test result is FAIL                                   (235)
END IF
IF (RESPONSE_CODE != "Success")
      THEN Test result is FAIL                                   (236)
END IF
IF (Response Time > STO)
      THEN Test result is FAIL                                   (237)
END IF
IF (Response != Command 2)
      THEN Test result is FAIL                                   (238)
END IF

END TEST
```

7.42 DLL042 Command Number Expansion

This test sends command 31 to the DUT. If the DUT does not support any expanded commands then it must answer command 31 with "Command Not Implemented". Otherwise, it must answer with "Too Few Data Bytes". Upon confirmation of command expansion support, proper operation of the command expansion mechanism is confirmed.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	
Common Practice Command Specification	8.0	

Test Procedure

```
CALL IdentifyDevice

SEND Command 31 with no data bytes
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                        (365)
END IF
IF (RESPONSE_CODE = "Command not Implemented")
      THEN Abort Test (i.e., test is not applicable to this device)
END IF

IF (RESPONSE_CODE != "Too few data byes received")
      THEN test result is FAIL                                        (240)
END IF

SEND Command 31 with Data = 0xFE
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                        (366)
END IF
IF (RESPONSE_CODE != "Too few data byes received")
      THEN test result is FAIL                                        (241)
END IF

SEND Command 31 with Data = 0xFE, 0x00
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                        (367)
END IF
IF (RESPONSE_CODE != "Command not Implemented")
      THEN test result is FAIL                                        (242)
END IF

SEND Command 31 with Data = 0x00, 0x03
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                        (368)
END IF
IF (RESPONSE_CODE != "Invalid Extended Command Number")
  AND (RESPONSE_CODE != "SUCCESS")
      THEN test result is FAIL                                        (243)
END IF

END TEST
```

7.43 DLL043 Write Burst Device Variables

Verifies that the DUT responds properly to Command 107. Checks Addresses, Command Number, Response Code and Byte Count for Command 107, Write Burst Device Variable. The following conditions are evaluated.

- The DUT must support Commands 105, 108, and 109 if Command 107 is supported.
- The DUT is placed in Burst-Mode transmitting Command 9 (verified using Command 105).
- A Valid Device Variable is identified using Command 9 or 33.
- Varying the length of Command 107, 1-4 slots are configured (verified using Command 105).
- A 5 byte Command 107 is sent (a normal response is required).
- The DUT is taken out of Burst-Mode.

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

References:

Specification	**Rev.**	**Sections**
Common Practice Command Specification	7.0	6.9, 7.52

Test Case A: Basic Command 107 Support.

This test case checks that command 107 can be read and written. This test is performed using 4 device variables and no burst message field. HART 7 or later devices must operate in backward compatible mode and assume we are provisioning burst message 0.

```
CALL IdentifyDevice
```

Verify that Burst Mode is supported on wired connection of device under test.

```
CheckReadyForAnyBurst()
CALL CheckCommandImplemented(107)
```

The burst command will vary based on revision supported.

```
SWITCH UNIV_REVISION
CASE 5
      SET Cmd105BC = NULL
      SET BurstCmd = 33
CASE 6
      SET Cmd105BC = 8
      SET BurstCmd = 9
DEFAULT
      SET Cmd105BC = 31
      SET BurstCmd = 9
END SWITCH
```

There must be at least one valid Device Variable

```
dVar = -1
IF (FindNextDeviceVariable(dVar) == "No More Device Variables")
      THEN Test result is FAIL                                    (1030)
END IF
```

Put into Burst Mode

```
SEND Command 108 (with BurstCmd)
SEND Command 109 to put the DUT into Burst Mode

MONITOR BACKs
IF (no BACKS detected)
      THEN Test Result is FAIL                                       (1031)
END IF
IF (no BACK is not BurstCmd)
      THEN Test Result is FAIL                                       (1032)
END IF
```

Test response burst variable settings

```
FOR n = 1 to 4
   SEND Command 107 with n data bytes of dVar
   CALL TestValidFrame
   IF (UNIV_REVISION > 6)
      CALL VerifyResponseAndByteCount(0, 11)
   ELSE
      CALL VerifyResponseAndByteCount(0, n+2)
   END IF
   IF data bytes  0 through n-1 are not all dVar
      THEN Test result is FAIL                                       (1033)
   END IF
```

Verify slot settings

```
   IF (BACK does not have n slots all set to dVar)
      THEN Test Result is FAIL                                       (1035)
   END IF
   IF (UNIV_REVISION >= 6)
      SEND Command 105
      CALL VerifyResponseAndByteCount(0, Cmd105BC)
      IF Slot [0 - (n-1) ] are not all dVar
         THEN Test result is FAIL                                    (1036)
      END IF
      IF Slot [n to 8] are not all 250
         THEN Test result is FAIL                                    (1037)
      END IF
      IF BurstMessage != 0
         THEN Test result is FAIL                                    (1038)
      END IF
   END IF
END FOR
SEND Command 109 to take the DUT out of Burst Mode
IF (any BACKS detected)
   THEN Test Result is FAIL                                          (1040)
END IF

SEND Command 107 with maxVars+1 data bytes of dVar
CALL TestValidFrame
IF (UNIV_REVISION > 6)
   CALL VerifyResponseAndByteCount(0, 11)
ELSE
   CALL VerifyResponseAndByteCount(0, 6)
END IF

END TEST CASE
```

Test Case B: Command 107 Support Across Burst Messages.

This test case checks that command 107 can be read and written differently for each burst messsage.

```
CALL IdentifyDevice
IF UNIV_REVISION < 7 THEN
   PRINT "WARNING: Implementation of HART 7 is strongly recommended."
   Abort Test                                                    (1045)
END IF
```

Verify that Burst Mode is supported on wired connection of device under test.

```
CheckReadyForAnyBurst()
```

Make sure there are enough burst messages

```
SEND Command 105 with bmsg = 0
CALL VerifyResponseAndByteCount ( 0, 31 )

IF MAX_BURST_MSGS < 3
   THEN Test Result is FAIL                                      (1046)
END IF
IF DEVICE_PROFILE_CODE is 141 or 142
   IF MAX_BURST_MSGS < 5
      THEN Test Result is FAIL                                   (1047)
   END IF
END IF
```

Find supported Device Variables

```
SET dVar = -1
SET dVarList[] = NULL
SET dVarCnt = 0

While (FindNextDeviceVariable(dVar) != "No More Device Variables")
   ADD dVar to dVarList
   INCREMENT dVarCnt
END WHILE
```

If the DUT does not expose enough Device Variables, then add PV, SV, TV, QV to dVarList

```
WHILE (dVarCnt < MAX_BURST_MSGS)
   SET dVarCnt = dVarCnt + 4
   ADD 246,247,248,249 to dVarList
END WHILE
```

Test response burst variable settings

```
FOR n = 1 to MAX_BURST_MSGS
   SEND Command 107 and 8 slots of dVarList[n] and Burst Message = n
   CALL TestValidFrame
   CALL VerifyResponseAndByteCount(0, 11)
END FOR
```

Verify slot settings

```
FOR n = 1 to MAX_BURST_MSGS
   SEND Command 105 with Burst message = n
   CALL VerifyResponseAndByteCount(0, 31)
   IF Slot [0 - (n-1) ] are not all dVarList[n]
      THEN Test result is FAIL                                   (1048)
   END IF
END FOR

END TEST CASE
```

7.44 DLL044 Burst Mode Mixed Operations

This test verifies that each Burst Message can support a different configuration. Proper operation of the DUT is confirmed as the burst message setting are modified while Burst Mode is enabled.

References:

Specification	Rev.	Sections
Data Link Layer Specification	8.0	
Common Practice Command Specification	8.0	7.53

Test Case A: Verify support for at least 3 Burst Messages

Verify that the device supports the required number of burst messages and turn them on each with a differernt setting.

```
CALL IdentifyDevice
IF UNIV_REVISION < 7 THEN
   PRINT "WARNING: Implementation of HART 7 is strongly recommended."
   Abort Test                                                          (940)
END IF
```

Verify that Burst Mode is supported on wired connection of device under test.

```
CheckReadyForAnyBurst()
```

Make sure there are enough burst messages

```
SEND Command 105 with bmsg = 0
CALL VerifyResponseAndByteCount ( 0, 31 )

IF MAX_BURST_MSGS < 3
   THEN Test Result is FAIL                                            (941)
END IF
IF DEVICE_PROFILE_CODE is 141 or 142
   IF MAX_BURST_MSGS < 5
      THEN Test Result is FAIL                                         (942)
   END IF
END IF
```

Find supported Device Variables

```
SET dVar = -1
SET dVarList[] = NULL
SET dVarCnt = 0
While (FindNextDeviceVariable(dVar) != "No More Device Variables")
   ADD dVar to dVarList
   INCREMENT dVarCnt
END WHILE
CALL IdentifyDevice
```

If the DUT does not expose enough Device Variables, then add PV, SV, TV, QV to dVarList

```
WHILE (dVarCnt < 8)
   SET dVarCnt = dVarCnt + 4
   ADD 246,247,248,249 to dVarList
END WHILE
```

OK we can test. First do an easy one - One burst message using Command 3. Set up and Enable Burst mode, make sure we are getting Command 3 BACKs

```
SEND Command 108 (with Command = 3 and BMessage = 2)
```

```
CALL VerifyResponseAndByteCount(0, 5)
SEND Command 103 with BMessage = 2 and both Update Periods set to 1sec.
CALL VerifyResponseAndByteCount(0, 11)
END Command 109 with BMessage = 2 to turn burst on
CALL VerifyResponseAndByteCount(0, 4)

FOR (20 BACKs)
   IF (BACK does NOT contain Command 3)
      THEN Test Result is FAIL                                         (943)
   END IF
   SEND Command 9 with dVarSet = {246, 247, 248, 249}
   IF ( (RESPONSE CODE != "Success")
     AND (RESPONSE CODE != " Update Failure") )
      THEN Test result is FAIL                                         (945)
   END IF
END FOR
```

Now load up the Burst messages. First, configure a burst message 1 for command 9.

```
SEND Command 108 (with command = 9 and burst message = 1)
CALL VerifyResponseAndByteCount(0, 5)
SEND Command 103 with BMessage = 1 and both Update Periods set to 1sec.
CALL VerifyResponseAndByteCount(0, 11)
SEND command 107 with slots assigned from the first 8 entries in dVarList
CALL VerifyResponseAndByteCount(0, 11)
SEND Command 109 with burst enabled and burst message = 1
CALL VerifyResponseAndByteCount(0, 4)
```

Configure a burst message 0 for command 48.

```
SEND Command 108 (with command = 48 and burst message = 0)
CALL VerifyResponseAndByteCount(0, 5)
SEND Command 103 with BMessage = 0 and both Update Periods set to 16sec.
CALL VerifyResponseAndByteCount(0, 11)
SEND Command 109 with burst enabled and burst message = 0
CALL VerifyResponseAndByteCount(0, 4)
```

Device burst messages include command 3, 9, 48.

```
SET Cmd3BackCnt = 0
SET Cmd9BackCnt = 0
SET Cmd48BackCnt = 0

FOR (200 BACKs)
   SWITCH on command in BACK

   CASE Command 3
      INCREMENT Cmd3BackCnt

   CASE Command 9
      INCREMENT Cmd9BackCnt

   CASE Command 48
      INCREMENT Cmd48BackCnt

   CASE DEFAULT
      Test result is FAIL                                              (946)

   END SWITCH
END FOR
```

```
IF (Cmd3BackCnt == 0)
   THEN Test result is FAIL                                          (947)
END IF
IF (Cmd9BackCnt == 0)
   THEN Test result is FAIL                                          (948)
END IF

IF (Cmd48BackCnt == 0)
   THEN Test result is FAIL                                          (950)
END IF

IF (Cmd3BackCnt much larger than Cmd9BackCnt)  OR
  (Cmd9BackCnt much larger than Cmd3BackCnt)
   THEN Test result is FAIL                                          (951)
END IF

IF (Cmd3BackCnt NOT much larger than Cmd48BackCnt)  OR
  (Cmd9BackCnt NOT much larger than Cmd48BackCnt)
    THEN Test result is FAIL                                         (952)
END IF
```

Turn off all the bursts.

```
SEND Command 109 with burst disable and each burst message

END TEST CASE
```

Test Case B: Verify on-the-fly burst configuration changes

Verify the triggers and update periods settings.

```
CALL IdentifyDevice
IF UNIV_REVISION < 7 THEN
   PRINT "WARNING: Implementation of HART 7 is strongly recommended."
   Abort Test                                                        (953)
END IF
```

Verify that Burst Mode is supported on wired connection of device under test.

```
CheckReadyForAnyBurst()
```

Make sure there are enough burst messages

```
SEND Command 105 with bmsg = 0
CALL VerifyResponseAndByteCount ( 0, 31 )

IF MAX_BURST_MSGS < 3
   THEN Test Result is FAIL                                          (955)
END IF
IF DEVICE_PROFILE_CODE is 141 or 142
   IF MAX_BURST_MSGS < 5
      THEN Test Result is FAIL                                       (956)
   END IF
END IF
```

OK we can test. Set up burst message 0

```
    SET bm0VarList = {246, 247, 248, 249, 250, 250, 250, 250}
    SEND Command 107 (with BMessage = 0 and bm0VarList)
    CALL VerifyResponseAndByteCount(0, 11)
    SEND Command 108 (with Command = 0x0009 and BMessage = 0)
    CALL VerifyResponseAndByteCount(0, 5)
    SEND Command 103 with BMessage = 0 and both Update Periods set to 1sec.
    CALL VerifyResponseAndByteCount(0, 11)
    SEND Command 104 with BMessage = 0, "Continuous", and trigger value = NaN.
    CALL VerifyResponseAndByteCount(0, 10)
```

Set up burst message 1

```
    SET bm1VarList = {247, 248, 249, 250, 250, 250, 250, 250}
    SEND Command 107 (with BMessage = 1 and bm1VarList)
    CALL VerifyResponseAndByteCount(0, 11)
    SEND Command 108 (with Command = 0x0009 and BMessage = 1)
    CALL VerifyResponseAndByteCount(0, 5)
    SEND Command 103 with BMessage = 1 and both Update Periods set to 1sec.
    CALL VerifyResponseAndByteCount(0, 11)
    SEND Command 104 with BMessage = 1, "Continuous", and trigger value = NaN.
    CALL VerifyResponseAndByteCount(0, 10)
```

Set up burst message 2

```
    SET bm2VarList = {248, 249, 250, 250, 250, 250, 250, 250}
    SEND Command 107 (with BMessage = 0 and bm2VarList)
    CALL VerifyResponseAndByteCount(0, 11)
    SEND Command 108 (with Command = 0x0009 and BMessage = 2)
    CALL VerifyResponseAndByteCount(0, 5)
    SEND Command 103 with BMessage = 2 and both Update Periods set to 1sec.
    CALL VerifyResponseAndByteCount(0, 11)
    SEND Command 104 with BMessage = 2, "Continuous", and trigger value = NaN.
    CALL VerifyResponseAndByteCount(0, 10)

    SET Cmd1BackCnt = 0
    SET Cmd3BackCnt = 0
    SET Cmd9-246Cnt = 0
    SET Cmd9-247Cnt = 0
    SET Cmd9-248Cnt = 0

    SEND Command 109 with BMessage = 2 to turn burst on
    CALL VerifyResponseAndByteCount(0, 4)
```

As the BACKs roll-in, change the burst configuration; start with only burst message 2 enabled

```
      FOR (1000 BACKs)
         SWITCH on NumBACKs

         CASE 100
            IF (Cmd9-246Cnt != 0)
               THEN Test Result is FAIL                                   (957)
            END IF
            SEND Command 109 with BMessage = 0 to turn burst on
            CALL VerifyResponseAndByteCount(0, 4)

         CASE 200
            IF (Cmd9-247Cnt != 0)
               THEN Test Result is FAIL                                   (958)
            END IF
            SEND Command 109 with BMessage = 1 to turn burst on
            CALL VerifyResponseAndByteCount(0, 4)

         CASE 400
            IF (Cmd3BackCnt != 0)
               THEN Test Result is FAIL                                   (960)
            END IF
            IF (Cmd9-247Cnt == 0)
               THEN Test Result is FAIL                                   (961)
            END IF
            SET Cmd9-247Cnt = 0
            SEND Command 108 (with Command = 0x0003 and BMessage = 1)
            CALL VerifyResponseAndByteCount(0, 5)

         CASE 600
            IF (Cmd1BackCnt != 0)
               THEN Test Result is FAIL                                   (962)
            END IF
            IF (Cmd9-248Cnt == 0)
               THEN Test Result is FAIL                                   (963)
            END IF
            SET Cmd9-248Cnt = 0
            SEND Command 108 (with Command = 0x0001 and BMessage = 2)
            CALL VerifyResponseAndByteCount(0, 5)

         CASE 800
            IF (Cmd1BackCnt == 0)
               THEN Test Result is FAIL                                   (965)
            END IF
            IF (Cmd9-248Cnt != 0)
               THEN Test Result is FAIL                                   (966)
            END IF
            SET Cmd1BackCnt = 0
            SEND Command 108 (with Command = 0x0009 and BMessage = 2)
            CALL VerifyResponseAndByteCount(0, 5)

         END SWITCH
```

Keep count of the burst commands we see.

```
      SWITCH on command in BACK
      CASE Command 1
         INCREMENT Cmd1BackCnt

      CASE Command 3
         INCREMENT Cmd3BackCnt

      CASE Command 9
         SWITCH on Command dVar[0]in BACK
         CASE Command 246
            INCREMENT Cmd9-246Cnt

         CASE Command 247
            INCREMENT Cmd9-247Cnt

         CASE Command 248
            INCREMENT Cmd9-248Cnt

         CASE DEFAULT
            Test result is FAIL                                        (967)

         END SWITCH
      CASE DEFAULT
         Test result is FAIL                                           (968)

      END SWITCH
```

Now do a final check on the command counts.

```
      IF (Cmd1BackCnt != 0)
         THEN Test Result is FAIL                                      (970)
      END IF
      IF (Cmd3BackCnt == 0)
         THEN Test Result is FAIL                                      (971)
      END IF
      IF (Cmd9-246Cnt != 0)
         THEN Test Result is FAIL                                      (972)
      END IF
      IF (Cmd9-247Cnt != 0)
         THEN Test Result is FAIL                                      (973)
      END IF
      IF (Cmd9-248Cnt == 0)
         THEN Test Result is FAIL                                      (974)
      END IF

   END FOR
```

Turn off all the bursts.

```
   SEND Command 109 with burst disable and each burst message

   END TEST CASE
```

7.45 DLL045 Smart Data Publishing

Verifies that the DUT responds properly to Command 103 and 104. Checks Command Number, Response Code and Byte Count for Command 103 and 104.

References:

Specification	Rev.	Sections
Common Practice Command Specification	9.0	7.76, 7.75, 7.71, 7.70

Test Procedure

Verify the triggers and update periods settings.

```
CALL IdentifyDevice
IF UNIV_REVISION < 7 THEN
   PRINT "WARNING: Implementation of HART 7 is strongly recommended."
   Abort Test                                                        (980)
END IF
```

Verify that Burst Mode is supported on wired connection of device under test.

```
CheckReadyForAnyBurst()
```

Make sure there are enough burst messages

```
SEND Command 105 with bmsg = 0
CALL VerifyResponseAndByteCount ( 0, 31 )

IF MAX_BURST_MSGS < 3
   THEN Test Result is FAIL                                          (981)
END IF
IF DEVICE_PROFILE_CODE is 141 or 142
   IF MAX_BURST_MSGS < 5
      THEN Test Result is FAIL                                       (982)
   END IF
END IF
```

Find supported Device Variables

```
SET dVar = -1
SET dVarList[] = NULL
SET dVarCnt = 0
While (FindNextDeviceVariable(dVar) != "No More Device Variables")
   ADD dVar to dVarList
   INCREMENT dVarCnt
END WHILE
CALL IdentifyDevice
```

If the DUT does not expose enough Device Variables, then add PV, SV, TV, QV to dVarList

```
WHILE (dVarCnt < 8)
   SET dVarCnt = dVarCnt + 4
   ADD 246,247,248,249 to dVarList
END WHILE
```

OK we can test. First do an easy one - One burst message using Command 3. Set up and Enable Burst mode, make sure we are getting Command 3 BACKs

```
SEND Command 108 (with Command = 3 and BMessage = 2)
CALL VerifyResponseAndByteCount(0, 5)
SEND Command 103 with BMessage = 2 and both Update Periods set to 1sec.
```

```
CALL VerifyResponseAndByteCount(0, 11)
END Command 109 with BMessage = 2 to turn burst on
CALL VerifyResponseAndByteCount(0, 4)

FOR (20 BACKs)
   IF (BACK does NOT contain Command 3)
      THEN Test Result is FAIL                                     (983)
   END IF
   SEND Command 9 with dVarSet = {246, 247, 248, 249}
   IF ( (RESPONSE CODE != "Success")
     AND (RESPONSE CODE != " Update Failure") )
      THEN Test result is FAIL                                     (985)
   END IF
END FOR
```

Now setup Command 9 as burst message 0. Start by using Command 54 to determine the update rates for the desired Device Variables.

```
SEND Command 54 with Device Variable = dVarList[0]
IF (RESPONSE CODE != "Success")
   THEN Test result is FAIL                                        (986)
END IF

UPDATE_PERIOD = Command 54 response
IF UPDATE_PERIOD < 4 sec
   THEN SET UPDATE_PERIOD = 4 SEC
END IF
MAXIMUM_UPDATE_PERIOD = UPDATE_PERIOD * 4
```

Use Command 103 to set the Update Period and Maximum Update Period for publishing the Burst Message.

```
SEND Command 103 (Burst Message = 0)with UPDATE PERIOD
   and MAXIMUM UPDATE PERIOD
CALL VerifyResponseAndByteCount(0, 11)
SEND Command 108 (with command = 9 and burst message = 0)
CALL VerifyResponseAndByteCount(0, 5)
SEND command 107 with slots assigned from the first 8 entries in dVarList
CALL VerifyResponseAndByteCount(0, 11)
SEND Command 109 with burst enabled and burst message = 0
CALL VerifyResponseAndByteCount(0, 4)
```

Use Command 104 to set the trigger mode to rising and a value > current value.

```
SEND Command 9 with the first 8 entries in dVarList
IF BYTE_COUNT != 71
   THEN Fail                                                       (987)
END IF
IF (RESPONSE CODE != "Success") AND (RESPONSE CODE != "Update Failure")
   THEN Fail                                                       (988)
END IF

SEND Command 104 with trigger = "Rising"; trigger level > Command 9
   slot 0 valueand burst message = 0
CALL VerifyResponseAndByteCount(0, 10)
```

Verify the settings

```
SEND Command 105 with Burst Message = 0
CALL VerifyResponseAndByteCount(0, 31)
```

```
IF burst message !=0
   THEN Test result is FAIL                                    (989)
END IF
IF burst is NOT enabled
   THEN Test result is FAIL                                    (990)
END IF
IF Command expansion flag != 31
   THEN Test result is FAIL                                    (991)
END IF
IF device variables != first 8 entries in dVarList
   THEN Test result is FAIL                                    (992)
END IF
IF extended command number != 9
   THEN Test result is FAIL                                    (993)
END IF
IF update time != UPDATE PERIOD
   THEN Test result is FAIL                                    (994)
END IF
IF max update time != MAXIMUM UPDATE PERIOD
   THEN Test result is FAIL                                    (995)
END IF
IF trigger != "Rising"
   THEN Test result is FAIL                                    (996)
END IF
IF trigger level != value set with Command 104
   THEN Test result is FAIL                                    (997)
END IF
```

Device burst messages include command 3, 9, 48.

```
SET Cmd3BackCnt = 0
SET Cmd9BackCnt = 0
SET Cmd48BackCnt = 0

FOR (500 BACKs)
   SWITCH on command in BACK

   CASE Command 3
      INCREMENT Cmd3BackCnt

   CASE Command 9
      INCREMENT Cmd9BackCnt
      CALCULATE Cmd9AveUpdatePeriod

   CASE Command 48
      INCREMENT Cmd48BackCnt

   CASE DEFAULT
      Test result is FAIL                                      (998)

   END SWITCH
END FOR

IF (Cmd3BackCnt == 0)
   THEN Test result is FAIL                                   (1000)
END IF
IF (Cmd9BackCnt == 0)
   THEN Test result is FAIL                                   (1001)
END IF
```

```
IF (Cmd48BackCnt != 0)
   THEN Test result is FAIL                                           (1002)
END IF

IF (Cmd9AveUpdatePeriod != MAXIMUM_UPDATE_PERIOD)
   THEN Test result is FAIL                                           (1003)
END IF
```

Use Command 104 to set the trigger mode to rising and a value < current value.

```
SEND Command 9 with the first 8 entries in dVarList
SEND Command 104 with trigger = "Rising"; trigger level < Command 9
  slot 0 valueand burst message = 0
CALL VerifyResponseAndByteCount(0, 10)
```

Verify the settings

```
SEND Command 105 with Burst Message = 0
CALL VerifyResponseAndByteCount(0, 31)
IF trigger != "Rising"
   THEN Test result is FAIL                                           (1005)
END IF
IF trigger level != value set with Command 104
   THEN Test result is FAIL                                           (1006)
END IF
```

Device burst messages include command 3, 9, 48.

```
SET Cmd3BackCnt = 0
SET Cmd9BackCnt = 0
FOR (500 BACKs)
   SWITCH on command in BACK
   CASE Command 3
      INCREMENT Cmd3BackCnt
   CASE Command 9
      INCREMENT Cmd9BackCnt
      CALCULATE Cmd9AveUpdatePeriod
   CASE DEFAULT
      Test result is FAIL                                             (1007)

   END SWITCH
END FOR

IF (Cmd3BackCnt == 0)
   THEN Test result is FAIL                                           (1008)
END IF
IF (Cmd9BackCnt == 0)
   THEN Test result is FAIL                                           (1010)
END IF
IF (Cmd9AveUpdatePeriod != UPDATE_PERIOD)
   THEN Test result is FAIL                                           (1011)
END IF
```

Turn off all the bursts.

```
SEND Command 109 with burst disable and each burst message

END TEST
```

ANNEX A. COMMON TEST MACROS AND DEFINITIONS

The procedures in this appendix are used in two or more of the UAL test definitions. They are presented here as reusable procedures to remove redundancy in the Test Body.

A1 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 )
      THEN Test result is FAIL      (500)
ELSE  IF ((RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Busy"))
            THEN Test result is FAIL                                  (501)
      ELSE
            deviceFound = TRUE
      END IF
END WHILE

IF (!deviceFound)
      THEN Test result is FAIL                                        (502)
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)   (503)
END IF
IF UNIV__REVISION > 6
      THEN Abort Test (i.e., test is not applicable to this device)   (507)
END IF

PROCEDURE END
```

A2 CheckDeviceAlive ( )

Make sure the device is still working

```
PROCEDURE CheckDeviceAlive()

SEND Command 1
IF (COMMUNICATIONS_ERROR)
      THEN Test result is FAIL                                        (504)

IF (UNIV__REVISION > 5) THEN
IF (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Update Failure")
      THEN Test result is FAIL      (505)
      END IF
ELSE
      IF (RESPONSE_CODE != 0) AND (RESPONSE_CODE != "Busy")
        AND (RESPONSE_CODE != "Update Failure")
            THEN Test result is FAIL                                  (506)
      END IF
END IF

PROCEDURE END
```

A3 CheckSlaveSTO (command)

The following procedure is used to validate that the DUT ACK is received within STO.

```
PROCEDURE CheckSlaveSTO(Command)

SEND Command
Record Slave Response Time
IF (No Response)
      THEN Test result is FAIL                                        (516)
END IF
IF (RESPONSE_CODE != 0)
      IF ((Command != [1, 2, or 3])
        OR (RESPONSE_CODE != "Update Failure"))
            THEN Test result is FAIL                                  (517)
      END IF
END IF
IF (RESPONSE_CODE != "Command not Implemented")
      IF ((Command != [4; 5; 127; 33,792-64,511;
         64,766-64,767; or 65,022-65,535])
            THEN Test result is FAIL                                  (519)
      END IF
END IF
IF (Response Time > STO)
      THEN Test result is FAIL                                        (518)
END IF
PROCEDURE END
```

A4 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      (510)
END IF
WHILE (RESPONSE_CODE == "Busy" )
IF (RESPONSE_CODE != 0)
      THEN Test result is FAIL                                         (511)
END IF
```

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                                         (509)
END IF

IF (DUT is in "Write Protect")
      THEN Test result is FAIL                                         (512)
END IF
PROCEDURE END
```

A5 CheckBurstCommands ( )

The following procedure is used to verify that the DUT supports all the applicable Burst Mode commands. This procedure assumes Command 109 has already been used to confirm that the DUT supports Burst Mode (see CheckReadyForBurst and CheckReadyForAnyBurst). All commands are sent with no data bytes. Table 19 indicates the expected reponses from the DUT for each Command. Aborts the test if burst mode not supported on Token-Passing Data-Link.

Table 19 Burst Mode Command Requirements

Cmd	RC	BC	Requirement
101	5 64	2 2	HART 7 or later I/O Systems only (Profile Code == 141,142) For DUT not an I/O System
102	5 64	2 2	HART 7 or later I/O Systems only (Profile Code == 141,142) For DUT not an I/O System
103	5 64	2 2	HART 7 or later. HART 5 or 6
104	5 64	2 2	HART 7 or later. HART 5 or 6
105	0 0 64	31 8 2	HART 7 or later HART 6 HART 5
107	5 5,64	2 2	HART 6 or later HART 5 (recommended: print warning if not suppoerted)
108	5	2	HART 5 or later

```
PROCEDURE CheckBurstCommands ()
```

Sequentially send Commands in Table 19 with zero data bytes.

```
FOR each TEST_VECTOR in Table 19
      SEND Cmd with zero data bytes
      IF there is no response
            THEN Test result is FAIL          (570+TEST_VECTOR_NUMBER)
      END IF
      CALL TestValidFrame()
      IF (RESPONSE_CODE is not in list)
            THEN Test Result is FAIL          (580+TEST_VECTOR_NUMBER)
      END IF
      IF (BYTE_COUNT is not in list)
            THEN Test Result is FAIL          (590+TEST_VECTOR_NUMBER)
      END IF
END FOR

PROCEDURE END
```

A6 CheckReadyForBurst ( )

The following procedure is used to verify that the DUT supports Burst mode, is not in Burst Mode and is not Write Protected. Aborts the test if burst mode not supported on Token-Passing Data-Link.

```
PROCEDURE CheckReadyForBurst()

CALL VerifyNotWriteProtected()
SEND Command 109 with no data bytes
IF (RESPONSE_CODE == "Command not Implemented") THEN
   IF IF DEVICE_PROFILE_CODE > 128
      THEN Test Result is FAIL                                            (510)
   ELSE
      PRINT "WARNING: Implementation of Burst Mode is
         strongly recommended."
      Abort Test                                                          (513)
END IF
VerifyResponseAndByteCount ("Too Few Data Bytes", 2 )

IF UNIV_REVISION > 6 THEN
   IF DEVICE_PROFILE_CODE > 128 THEN
      SEND Command 109 with CONTROL = 1; BURST_MESSAGE = 0
      IF (RESPONSE_CODE != "Success")
         THEN Abort Test                                                  (514)
      END IF
   END IF
END IF

SEND Command 109 to take the DUT out of Burst Mode
CALL CheckBurstCommands ()
```

Make sure the device is set up for bursting Command 3.

```
SEND Command 108 to burst command 3 (8-bit Command Number)
VerifyResponseAndByteCount(0, 3)

PROCEDURE END
```

A7 CheckReadyForAnyBurst ( )

The following procedure is used to verify that the DUT supports Burst mode, is not in Burst Mode and is not Write Protected.

```
PROCEDURE CheckReadyForAnyBurst()

CALL VerifyNotWriteProtected()
SEND Command 109 with no data bytes

IF (RESPONSE_CODE == "Command not Implemented") THEN
   IF DEVICE_PROFILE_CODE > 128
      THEN Test Result is FAIL                                     (525)
   ELSE
      PRINT "WARNING: Implementation of Burst Mode is
         strongly recommended."
      Abort Test                                                   (526)
END IF
VerifyResponseAndByteCount ("Too Few Data Bytes", 2 )
SEND Command 109 to take the DUT out of Burst Mode
CALL CheckBurstCommands ()
```

Make sure the device is set up for bursting Command 3.

```
SEND Command 108 to burst command 3 (8-bit Command Number)
VerifyResponseAndByteCount(0, 3)

PROCEDURE END
```

A8 FindNextDeviceVariable (dVar)

Find a supported Device Variable using Command 9. This was adapted from *Slave Common Practice Command, Test Specification* (HCF_TEST-4) and, thus, uses the same Failure Point Codes.

```
PROCEDURE FindNextDeviceVariable(dVar)

SWITCH UNIV_REVISION
CASE 5
      SET Cmd = 33
      SET CmdBC = 8
CASE 6
      SET Cmd = 9
      SET CmdBC = 11
DEFAULT
      SET Cmd = 9
      SET CmdBC = 15
END SWITCH
DO
      SET dVarFound = FALSE;
      INCREMENT dVar
      IF (dVar > 239) THEN
            RETURN "No More Device Variables"
      END IF
      SEND Command Cmd 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 != CmdBC)
            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 >5
      SEND Command 0 to read maxDeviceVars
      IF (dVar > maxDeviceVars)
            THEN Test result is FAIL                                 (5146)
      END IF
END IF
RETURN "Device Variable Found"
PROCEDURE END
```

A9 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
```

A10 VerifyResponseAndByteCount (rc, bc)

Verify that the reply to a command matches 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
```

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
2000	UAL000	x	x	x	x	x	x	x	x	x	x
100	DLL026	x									
100	DLL027		x								
100	DLL028			x							
100	DLL029				x						
100	DLL030					x					
100	DLL032						x				
100	DLL036							x			
100	DLL039								x	x	
110	DLL035		x								
110	DLL026			x							
110	DLL022					x					
110	DLL039						x				
200	DLL036	x		x							
200	DLL037							x	x	x	x
210	DLL037	x	x	x							
210	DLL038				x	x	x	x	x	x	x
220	DLL038	x	x	x	x	x					
220	DLL039						x	x	x	x	
230	DLL040	x	x	x	x						
230	DLL041						x	x	x	x	
240	DLL036						x	x			
240	DLL042	x	x	x	x						
250	DLL034	x	x	x				x	x	x	x
270	DLL040	x	x	x	x	x					
270	DLL018							x	x	x	
280	DLL035	x	x	x	x	x	x	x	x	x	x
290	DLL035	x	x	x	x	x					
300											
310	DLL039	x	x	x	x	x	x	x	x		x
320	DLL039	x	x								
320	DLL036						x				
330	DLL036	x	x	x	x	x	x	x	x		

FP Codes	Test	0	1	2	3	4	5	6	7	8	9
340	DLL030										x
340	DLL032									x	
350	DLL035					x			x	x	x
360	DLL042						x	x	x	x	
370	DLL023				x						
370	DLL027					x					
370	DLL022						x	x			
390	DLL016		x								
390	DLL029							x			
390	DLL033									x	x
400	DLL033	x									
400	DLL006		x								
400	DLL012			x							
400	DLL022					x					
400	DLL026						x	x	x	x	x
410	DLL033	x	x								
410	DLL032			x	x	x					
410	DLL025						x	x			
420	DLL022			x							
420	DLL026				x	x	x				
430	DLL026		x								
430	DLL024						x	x	x		
440											
450	DLL013		x	x	x	x	x				
460	DLL013		x	x	x	x	x				
470	DLL013		x	x	x	x	x				
480	DLL013		x	x	x	x	x				
490	DLL013	x	x	x							
500	Annex A		x	x	x	x		x	x		x
510	Annex A	x	x	x	x	x		x	x	x	x
520	Annex A						x	x			
520	DLL036	x	x	x							
570	Annex A	x	x	x	x	x	x	x			

FP Codes	Test	0	1	2	3	4	5	6	7	8	9
580	**Annex A**	x	x	x	x	x	x	x			
590	**Annex A**	x	x	x	x	x	x	x			
600	**DLL001**	x	x	x	x	x	x	x	x	x	x
610	**DLL002**			x	x	x		x	x	x	
620	**DLL002**	x	x	x	x	x	x	x	x	x	x
630											
640	**DLL003**		x	x	x						
650	**DLL004**	x									
660	**DLL005**	x	x	x	x	x	x	x	x		
670	**DLL006**	x	x	x	x		x	x	x	x	
680	**DLL007**	x	x	x	x	x					
690											
700	**DLL009**	x	x	x	x						
710	**DLL010**	x	x	x	x	x	x	x	x	x	
720	**DLL011**	x	x	x	x	x	x	x	x	x	
730	**DLL012**	x	x								
740											
750	**DLL014**	x	x								
760	**DLL015**		x	x	x	x	x	x	x	x	
770	**DLL016**	x	x								
780	**DLL017**	x	x	x	x	x	x				
790	**DLL018**	x	x	x	x	x	x	x			
790	**DLL019**										x
800	**DLL020**	x	x	x	x	x	x				
810	**DLL021**	x	x								
810	**DLL022**			x	x	x	x	x	x	x	
820	**DLL023**	x	x	x	x						
820	**DLL025**						x	x	x	x	x
830	**DLL025**	x	x	x	x						
830	**DLL026**						x				
830	**DLL027**									x	x
840	**DLL028**	x	x	x							

FP Codes	Test	0	1	2	3	4	5	6	7	8	9
840	**DLL029**				x	x	x	x	x	x	
840	**DLL030**										x
850	**DLL032**	x	x	x	x	x					
850	**DLL033**						x	x	x	x	x
860	**DLL033**	x	x	x	x	x	x	x	x	x	x
870	**DLL033**	x	x	x	x	x					
870	**DLL035**						x	x	x	x	x
880	**DLL035**	x	x	x	x	x	x	x	x	x	x
890	**DLL036**			x	x	x	x	x	x	x	x
900	**DLL036**						x	x	x		
900	**DLL033**				x						
910	**DLL037**	x	x	x	x	x	x	x	x	x	
920	**DLL037**						x	x	x		
930	**DLL037**	x	x	x	x	x	x	x	x	x	x
940	**DLL044**	x	x	x	x		x	x	x	x	
950	**DLL044**	x	x	x	x		x	x	x	x	
960	**DLL044**	x	x	x	x		x	x	x	x	
970	**DLL044**	x	x	x	x	x					
980	**DLL045**	x	x	x	x		x	x	x	x	x
990	**DLL045**	x	x	x	x	x	x	x	x	x	
1000	**DLL045**	x	x	x	x		x	x	x	x	
1010	**DLL045**	x	x								
1020											
1030	**DLL043**	x	x	x	x		x	x	x	x	
1040	**DLL043**	x					x	x	x	x	
1050	**DLL043**										
1060	**DLL043**										
5110	**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
DLL001 FSK Preamble Check	□ Pass □ Fail
DLL002 Delimiter Check	□ Pass □ Fail □ With Frame Exp.
DLL003 Frame Expansion Check	□ Pass □ Fail □ With Frame Exp.
DLL004 Short Frame Check	□ Pass □ Fail
DLL005 Master Address Bit Check	□ Pass □ Fail
DLL006 Burst Mode Bit Check	□ Pass □ Fail
DLL007 Long Frame Address Check	□ Pass □ Fail
DLL009 Incorrect Byte Count Check	□ Pass □ Fail
DLL010 Vertical Parity Check	□ Pass □ Fail
DLL011 Framing Error Check	□ Pass □ Fail
DLL012 Check Byte Test	□ Pass □ Fail
DLL013 FSK Gap Receive Timeout Test	□ Pass □ Fail
DLL014 Long Message Test	□ Pass □ Fail
DLL015 Start Of Message In Data Field Check	□ Pass □ Fail
DLL016 Preamble Check For BACK Frames	□ Pass □ Fail □ Not Applicable
DLL017 Preamble Check For ACK Frames	□ Pass □ Fail
DLL018 Gap Errors in ACK Frames Check	□ Pass □ Fail
	Device Variables (List) ________________________________
DLL019 Gap Check For BACK Frames	□ Pass □ Fail □ Not Applicable
DLL020 Dribble Byte Check For ACK Frames	□ Pass □ Fail
DLL021 Dribble Byte Test For BACK Frames	□ Pass □ Fail □ Not Applicable
DLL022 Test Host Address Bit For BACK Frames	□ Pass □ Fail □ Not Applicable
DLL023 Test Burst Mode Bit Of Burst-Mode Slave Frames	□ Pass □ Fail □ Not Applicable

Test	Result
DLL024 Test Slave Responds Within STO	□ Pass □ Fail
DLL025 Burst Hold During Master Preamble	□ Pass □ Fail □ Not Applicable
DLL026 Test Burst Response Time After a DUT ACK	□ Pass □ Fail □ Not Applicable □ Long Response Time
DLL027 Test Response Time Between Consecutive Bursts	□ Pass □ Fail □ Not Applicable □ Long Response Time
DLL028 BACK Timing with STXs Errors	□ Pass □ Fail □ Not Applicable □ Long Response Time
DLL029 Burst Mode Timeout On Other Slave	□ Pass □ Fail □ Not Applicable □ Long Response Time
DLL030 Burst After Response From Other Slave	□ Pass □ Fail □ Not Applicable □ Long Response Time
DLL032 Read Unique Identifier	□ Pass □ Fail □ >5 Request Preambles Suggested
DLL033 Write Polling Address	□ Pass □ Fail
DLL034 Read Unique Identifier with Tag	□ Pass □ Fail
DLL035 Write Number Of Response Preambles	□ Pass □ Fail □ Not Applicable
DLL036 Write Burst Mode Command Number	□ Pass □ Fail □ Not Applicable □ Command 33 NOT Supported
DLL037 Burst Mode Control	□ Pass □ Fail □ Not Applicable
DLL038 Read Unique Identifier With Long Tag	□ Pass □ Fail
DLL039 Slave Time-Out Stress Test	□ Pass □ Fail □ Long Preamble Sequence Detected Total Non-Responses=

Test	Result
DLL040 Unique Address Test	□ Pass □ Fail
DLL041 Framing Successive Messages	□ Pass □ Fail
DLL042 Command Number Expansion	□ Pass □ Fail
DLL043 Write Burst Device Variables	□ Pass □ Fail □ Not Applicable
DLL044 Burst Mode Mixed Operations	□ Pass □ Fail □ Not Applicable
DLL045 Smart Data Publishing	□ Pass □ Fail □ Not Applicable

ANNEX D. REVISION HISTORY

D1 Changes from Revision 2.1 to 3.0

This document was upgraded to Revision 3.0 since addition of a new test cases (DLL043-DLL045) is a functional change. In addition,

- The cover page and page 2 were upgraded to reflect the new HART logos.
- Created a new classification for Burst Mode related tests in Section 5.
- DLL014 was modified to extend the buffer size for HART 7 devices.
- Added a new test case to DLL024 to confirm DUT response times for extended (i.e., 16-bit) command numbers.
- DLL032 was modified to accommodate the revised identifier information in HART 7 command 0.
- DLL036 was modified to test for the additional burst message support as well as devices that support TDMA and token-passing data link layers. Added macro WriteBurstCommand in DLL036.
- DLL037 was modified to accommodate devices supporting TDMA and token-passing data link layers. In addition, three new test cases were added to DLL037.
- DLL039 was modified to accommodate the maintenance ports of TDMA devices.
- Added a power Off/On cycle to DLL040. After the power cycle, confirm we found as many devices after the power cycle as before it. In addition, a check was added to confirm all of device info after power cycle is unchanged.
- Added DLL043, DLL044, and DLL045.
- DLL043 was moved from the *Common Practice Test Specifications* (HCF_TEST-4) and improved for HART 7 and wireless device operation.
- DLL044 was added to test HART 7 extended and mixed capabilities of Burst Mode including multiple burst messages and changing burst message configuration on-the-fly.
- DLL045 was added to verify operation of Smart Data Publishing features like threshold burst messages and mixed update intervals.
- Imported FindNextDeviceVariable, FindNextDeviceVariable, TestValidFrame, and VerifyResponseAndByteCount test macros from *Common Practice Test Specifications* (HCF_TEST-4).
- Added new test macro, CheckBurstCommands, that does basic sanity check on burst commands. CheckBurstCommands incorporates a large portion of CheckReadyForBurst.
- CheckReadyForBurst simplified with addition of CheckBurstCommands. This macro aborts if Token-Passing Data-Link is not supported. Targets use in frame generation and bus arbitration tests.

- Added new test macro, CheckReadyForAnyBurst, to determine whether burst mode is supported or not. CheckReadyForAnyBurst is designed for use in Burst Mode service test procedures.

D2 Changes from Revision 2.0 to 2.1

This document was upgraded to Revision 2.1 to make minor corrections to the following test specifications:

DLL014 was modified to allow HART 5 Field Devices to utilize 24 byte receive buffers.

DLL018 was modified to identify the device variables before issuing command 9

A minor typo was corrected in DLL022 (the word "NOT" was added in one place).

In DLL023 failure point 373 is now documented.

DLL024B was modified to not send commands 11, 21, 39, 41, 42 or 73.

In DLL029 failure point 396 is now documented.

DLL035 now verifies that the correct number of preambles are sent every time Command 59 is used to change the number of preambles.

DLL039A now considers "update failure" (RESPONSE_CODE=8) a valid response.

Procedure IdentifyDevice() now fails if Universal Revision > 6.

Procedure CALL VerifyNotWriteProtected() now fails if a Write Protect Code other then 0, 1 or 251 is returned by the device.

D3 Changes from Revision 1.2 to 2.0

This document was updated with Revision 2.0 to reflect changes to referenced documents and to reformat certain document elements. In addition to formatting, the document was updated to reflect the following changes:

References to Structured Analysis (LIT-8) were removed from: DLL020, DLL022, DLL024, DLL025, DLL026, DLL027, DLL028, DLL029, DLL030, and DLL031.

References to the HART Specification documents were revised to reflect the renumbering of version 6.

References to the HART 5 to 6 Forward Compatibility Specification were removed.

D4 Changes from Revision 1.1 to 1.2

This document was updated with Revision 1.2 to reflect changes to referenced documents and to reformat certain document elements. In addition to formatting, the document was updated to reflect the following changes:

Trademark information was moved to copyright page.

All references to "Non-Burst Mode" were changed to "normal operating mode".

Where "Caveats" were not applicable (marked NA) the section was removed.

Heading titled "Summary of Changes" was renamed "Revision History".

All occurrences of the phrase "Terminology in this test:" were removed from Test Procedure sections.

All occurrences of the phrase "Data Link Layer" were changed to "Data Link Layer".

The test extensions in sections 8.1 Frame Detection and Recognition Tests, 8.2 Frame Generation, 8.3 Arbitration, and 8.4 Services were removed.

Changes to Tests DT00001 – DT00037

> Recommended Test Tools And Tests: TEST Identity and Post Processing tools were added.

Changes to Test DL00001 – DL00004

> Frame Detection and Recognition: Test Extensions were dropped from column Test Identity.
>
> Frame Generation: Test Extensions were dropped from column Test Identity.
>
> Arbitration: Test Extensions were dropped from column Test Identity.
>
> Services: Test Extensions were dropped from column Test Identity.

Changes to Test DT00002

> Description And Purpose, 3rd paragraph: "0 Hex to FE Hex" changed to "0 Hex to FD Hex." Deleted "Since some of these values can be interpreted by the slave device as indicating the presence of Expanded bytes (extra bytes between the address and command bytes) care must be taken to insert these extra bytes into the frame (between the address and command bytes) during the test. It is assumed the slave device will ignore these bytes."

Changes to Test DT00008, DT00017, DT00020

Test Body: The pseudo code for the sending of short frame commands 1, 2 and 3 deleted.

Changes to Test DT00022

Pass Criteria: In the second paragraph, command "IF The Burst Response immediately after the Slave Response to the Primary Host is not to the Secondary Host." was changed to "IF The Burst Response immediately after the Slave Response to the Primary Host is to the Secondary Host." In the third paragraph, command "IF The Burst Response immediately after the Slave Response to the Secondary Host is not to the Primary Host" was changed to "IF The Burst Response immediately after the Slave Response to the Secondary Host is to the Primary Host."

Changes to Test DT00033

Inclusion of document number in section labeled "Test verifies conformance to HART Specification".

D5Changes from Revision 1.0 to 1.1

The last revision to the document titled 'HART® Slave Data Link Layer, Test Specification' was HCF_TEST-1, Document Revision 1.0. This document has been updated with Revision 1.1 to reflect changes to referenced documents and to reformat certain document elements.

Changes to Test DT00002

Test Body: In the indented section under "FOR (Short Frame)" the second through fourth paragraphs were deleted since revision 5 devices do not support these short frame commands.