STANDARD

Discrete Applications Specification

HCF_SPEC-285, Revision 2.0

Release Date: 11 June, 2012

Release Date: 11 June 2012

Document Distribution / Maintenance Control / Document Approval
To obtain information concerning document distribution control, maintenance control and document approval, please contact the HART Communication Foundation at the address shown below.

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

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

http://www.hartcomm.org

Use of imperatives in HART Specifications
The key words (imperatives) "must", "required", "shall", "should", "recommended", "may", and "optional" when used in this document are to be interpreted as follows:

Must	Must, Shall, or Required denotes an absolute mandatory requirement. For example, "All HART Field Devices must implement all Universal Commands"
Should	Should or Recommended indicates a requirement that, given good cause/reason, can be ignored. However, the consequences of ignoring the requirement must be fully understood and well justified before doing so.
May	May or Optional identifies a requirement that is completely optional and can be supported at the discretion of the implementation. May can be used to identify optional Host Application or Master functionality and, when this is the case, does not imply the function is optional in Field Devices.

Intellectual Property Rights
The HCF does not knowingly use or incorporate any information or data into the HART Specifications 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 this specification 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

Table of Figures

List of Tables

Preface

This Preface is included for informational purposes only.

Adoption of HART and the sale of HART-enabled equipment continue to grow. There are many millions of HART devices installed and plant personnel favor HART due to its: simplicity, low cost, ease of use, and high value. Starting in late 2004, the HCF began evaluating wireless technology and developing a wireless addition to the HART standards. The effort was expanded to include enhancements to Block Data Transfers, Burst Mode, Alerts, Trends, and Discrete measurement points. This specification covers HART-enabled discrete applications.

While the scope of this specification was identified, its completion was deferred until after the initial release of the HART 7 Specifications. In other words, Revision 1.0 was largely empty and a place-holder allowing its later completion.

This Revision (2.0) is a complete specification and contains definitions for a set of commands that must be implemented by a Discrete or Hybrid Field Device and by Discrete Adapters. It also defines optional commands for the support of Discrete Logic Execution Engine (DLEU). Discrete Common Table definitions are specified to support the commands defined in this specification.

The addition of HART support for discrete applications further extends the power and scope of HART Technology. The HART *Discrete Applications Specification* supports products such as field powered electric actuators, discrete on-off motor starters, field switches for detection of abnormal operating conditions, proximity switches, photo sensors, contacts indicating equipment status, field pushbuttons, etc. It is expected that device manufactures will include HART diagnostics that, in turn, will differentiate HART Discrete Field Devices, Hybrid Field Devices and Discrete Adapters.

Introduction

This Introduction is included for informational purposes only.

The popularity of HART Communication has also stimulated industry demands for expanding application of the HART protocol into new domains beyond it historical roots in continuous measurement and control applications. In particular, many end-users have requested HART Technology support for discrete applications.

In most plants in the process industry, discrete applications play an important role in the overall plant control systems and operations. In some industry segments, over 80% of the total input-output count in a control system may be discrete. Traditionally, these discrete requirements have been addressed using discrete input-output cards contained in the control system (or its remote I/O) and using Programmable Logic Controllers (PLCs) interfaced to plant control systems.

The *Discrete Applications Specification* introduces support for three new classes of HART-enabled Products:

- **Discrete Field Devices** supporting, for example, pressure, level, or temperature switches, proximity and limit switches, solenoid valves, motor starters and simple motion/position control;
- **Hybrid Field Devices** like level transmitters that include redundant (backup) level switches or positioner with limit switches providing full-open or full-closed valve position; and
- **Discrete Adapters** that communicate with a connected PLC and allow mini/micro PLCs to be incorporated into HART Networks.

Figure 1 shows some of the features found in Discrete or Hybrid Field Devices and Discrete Adapters. Products enabled by this specification have inputs, outputs or can be connected to a PLC. Central to these products is the Discrete Variable. Discrete Variables are on/off or state-related values and may be inputs from or outputs to the plant. Discrete Variable may also contain (in the case of a Discrete Adapter) a copy of the register values from the connected PLC.

Inputs receive a plant signal and status and convert those signals into a digital value. The conversion process may includes signal conditioning, termination, isolation and/or indication for that signal's state. The input may be a simple Boolean value, push button inputs, binary coded data (BCD). If the input is an on/off or open/close type, such as with a push button or limit switch, the signal can be represented in a single bit. If, on the other hand, the state of the input varies, such as with a blocking valve, where the valve is open, closed, opening, or closing, the state requires a full word.

Output modules transmit single bit or state signals to activate various devices such as actuators, blocking valves, on-off valves, solenoids, and motor starters. The output maintains a target value and may include the actual value as well for a discrete output. Often the output module maintains the status of the output, too (i.e. whether the output is functioning correctly). Signal conditioning, termination, and isolation are also part of the output's function. A host application modifies discrete outputs by writing the target value of the output, and then monitors the transitions to intermediate and/or final states by reading the actual value.

In some cases it is beneficial for Discrete or Hybrid Field Devices to incorporate local control logic capability. This local control capability is run in the (optional) Discrete Logic Execution Unit (DLEU). This local control capability enables discrete field devices to perform a variety of functions including executing a local control action in response to input like field mounted stop/start controls, limit switches, proximity devices. When a DLEU is added to a complex device like a process analyzer (thus making it a Hybrid Field Device) complex calibration cycles can be specified and executed.

Discrete Adapters are (like WirelessHART Adapters) communication bridges. In this case, the sub-device is generally a mini/micro PLC (it could be a RTU, too). The Discrete Adapter reads from and writes to registers and presents those registers as Discrete Variables. This allows the (for example) PLC data to be used within the HART Network the same way as any other HART data. It also allows moderately powerful local control to be added to a HART Network in the form of a remote mounted PLC.

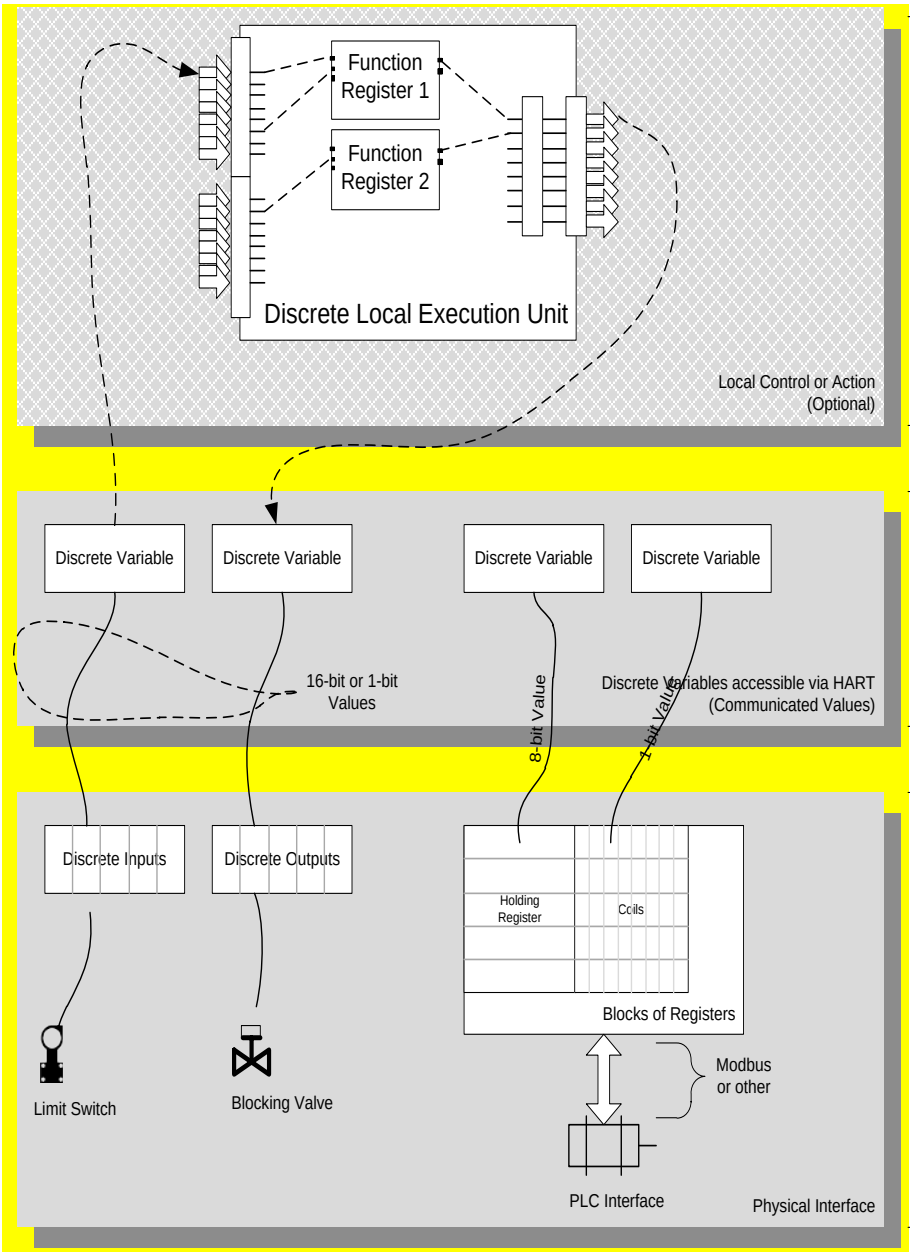


Figure 1. Discrete Variable Interfaces

These extensions to the HART Technology to support Discrete Applications are designed to enhance the information available in HART networks while maintaining ease of use, reliability, and low cost. All HART discrete products must include core mandatory capabilities that allow equivalent device types to be exchanged without compromising system operation. HART Discrete features are backward compatible to HART core technology such as the Device Description Language. Many HART Discrete Application requirements are mandatory and must be universally supported.

The addition of Discrete Applications improves the monitoring and control features of HART and further extends the reach of HART into equipment and process monitoring, asset management, diagnostics and predictive maintenance.

1. SCOPE

This document is an Application Layer specification and, as such, builds on the Application Layer requirements found in the *Command Summary Specification* (HCF_SPEC-99).

This specification defines compliance requirements for Discrete Devices. For compliance purposes, a Discrete Device shall be classified as one of three different product types. These product types are:

- **Discrete Field Devices** are support discrete sensors and/or actuators in the field. These devices support only discrete inputs, outputs, and associated diagnostics. In other words Discrete Field Devices must support Discrete Variables. Discrete Field Devices often include smart, predictive diagnostics about the sensors, actuators or the connected plant equipment.

- **Hybrid Field Devices** include both Continuous and Discrete capabilities. These devices combine both the capabilities of traditional HART Process Automation Field Devices and Discrete Field Devices. Hybrid field devices must support both Process Variables (i.e. Device Variables) and Discrete Variables.

- **Discrete Adapters** that connects a PLC[1] to a HART Network or HART-enabled I/O System. The Discrete Adapter must provide access to the registers inside the PLC and may allow the PLC program to be read/written via the HART Network using Block Data Transfer. Discrete Adapters must support Discrete Variables that are, in turn, mapped to PLC Registers.

Discrete and Hybrid Field Devices may support a Discrete Logic Execution Unit (DLEU). A DLEU is a simple logic engine that allows local control actions to be specified and executed thus avoiding network communication and central controller latency.

This Specification stipulates requirements for each of these device types, commands that they must support and Standard Operating Procedures (SOPs) that must be incorporated/supported. Discrete Devices are HART Devices and must comply with all HART Specifications. Where required, specific requirements (e.g., Common Practice Commands) are incorporated for other HART Specifications.

Document control procedures are also specified.

2. REFERENCES

2.1. HART Field Communications Protocol Specifications

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

HART Field Communications Protocol Specification. HCF_SPEC-13.

Command Summary Specification. HCF_SPEC-99

Common Practice Command Specification. HCF_SPEC-151

Common Tables Specification. HCF_SPEC-183

Block Data Transfer Specification. HCF_SPEC-190

WirelessHART Devices Specification. HCF_SPEC-290

2.2. Related HART Documents

The HART Protocol Specifications frequently reference the manufacturers' device-specific document. Device-specific documents are developed and controlled by the respective manufacturer and should follow the requirements of the following HART Communication Foundation document:

Field Device Specific Specification Template. HCF_LIT-18

[1] A micro-PLC could even be incorporated into the adapter itself.

2.3. Related Documents

The following provides guidelines to programming logic controllers

IEC 61131-3 Programmable Controllers - Programming Languages

Wikipedia, *Ladder logic*, http://en.wikipedia.org/wiki/Ladder_logic

Wikipedia, *Function block diagram*, http://en.wikipedia.org/wiki/Function_block_diagram

The following reference describes the methods for specifying state transition tables used in this document.

Hatley, D., and Pirbhai, I. *Strategies for Real-Time System Specification*. Dorset House, 1987.

3. DEFINITIONS

Definitions for terms can be found in *HART Field Communications Protocol Specification.* Terms used in this document include: Byte, Discrete, Boolean, Discrete Field Device.

Boolean Value	Two values: one and zero (which are equivalent to TRUE and FALSE)
Byte	8-bits. Sometimes called an Octet.
Discrete Adapter	Discrete Adapters connect (typically small) PLCs to a HART communication channel providing access to registers within the PLC and the reading/writing of the PLC Program.
Discrete Device	A Discrete Field Device, Hybrid Field Device or a Discrete Adapter.
Discrete Field Device	A Discrete Field Device supports field-level, discrete (on/off) input and output connections and may be able to perform simple logic, control or plant automation functions.
Discrete Logic Execution Unit	An optional component of a Discrete or Hybrid Field Device that provides the functionality of logic computation to be performed on one or more Discrete Variables.
Discrete Value	The values and status associated with a Discrete Variable.
Discrete Variable	A uniquely defined data item within a Discrete Device that allow access to discrete (e.g., on/off) values. Each Discrete Variable may include both a Target and an Actual discrete value plus status.
Drum	A set of configurable discrete values stored in a device for later retrieval. Drums are typically used for driving predetermined sequence logic.
Fault Behavior	A collection of Discrete Variable attributes defining the its behavior under fault conditions
Function Register	The atomic, programmable element of a DLEU. The Function Register specifies the function to be performed and the inputs and outputs to that function.
Hybrid Field Device	A Hybrid Device has both Process and Discrete Variables.
MUX	Not to be confused with an I/O System Multiplexer, in this document this term refers to a logic function contained within the DLEU that uses an input to select 1 value out of a set of multiple pre-programmed values to be applied to the output.
Process Variable	A uniquely defined floating-point data item within a Field Device that is always associated with cyclical, continuous and (typically) analog process information. A Process Variable's value varies in response to changes and variations in the process and includes Engineering Units and status.
Word	In most PLCs a word is 16bits wide (IEC61131-3).

4. SYMBOLS/ABBREVIATIONS

All Symbols and Abbreviations used in this specification are listed in this section.

DI	Discrete Input
DLEU	Discrete Logic Execution Unit
DO	Discrete Output
FALSE	0x0000
FB	The Fault Behavior of the Function Register.
FID	The enumerated function (i.e. logic) code for the Function Register
FR	Function Register
FSTATUS	The Function Register status indication. FSTATUS is set upon completion of the logic function in this Function Register.
I/O	Input/Output
IN1	Function Register Input 1
IN2	Function Register Input 2
IN3	Function Register Input 3
OUT	Function Register Output
PAC	Programmable Automation Controller
PAM	Plant Asset Management System
PAS	Plant Automation System
PLC	Programmable Logic Controller
SOP	Standard Operating Procedures
TRUE	0xFFFF

5. DOCUMENT CHANGE PROCEDURES

The Discrete Devices Specification is designed to allow the addition of new enumerations or tables to Section 10. These enumerations are applicable to all devices conforming to this specification and will be update, for example, when new state tables are specified. Consequently, Section 10 may be revised without balloting the Discrete Devices Specification by adhering the requirements and procedures in this section. These requirements are designed to:

- Ensure the technical excellence of each table;
- Provide a fair, open and objective basis for the development of additional tables;
- Alignment of the tables with Protocol requirements and practices; and
- Allow access and participation by all interested parties.

Any changes to this specification, other than additions or corrections to Section 10, must follow the normal change procedures as defined in the *HART Field Communications Protocol Specification*. Any additions or corrections must be approved by the Common Tables Committee (see the *Common Tables Specification*).

Discrete Devices Specification Revision 2.0 must be balloted and follow the normal process for HART Specifications as defined in *HART Field Communications Protocol Specification*. Beginning with the release of Revision 2.0 the document control procedures adhered to.

5.1. Revision Numbers

As per normal Protocol requirements, The *Discrete Devices Specification* has a major and minor revision number. As a specification is changed, the major and minor revision numbers are incremented as follows:

- The addition of a new table and/or new enumerations to existing tables is a functional change to this specification. The major revision number must be incremented and the minor revision number reset to zero.
- Minor corrections or clarifications (where the meaning does not change and/or corrections to misspelled words) shall be considered a non-functional change to this specification and the minor revision number must be incremented.
- Any other change to this specification must follow the revision numbering and change control procedures found in the HART Field Communications Protocol Specification.

5.2. Additions to Tables in Section 10

Any HCF member company or the HCF staff may submit request proposals for additions to the existing tables. All Request proposals must state the tables affected, changes envisioned and the benefits the changes would provide.

All requests are maintained in a database. This database shall track who and how often specific enumerations are requested.

Note: The frequency of an enumeration request is one acceptance criteria used by the Common Tables Committee.

Proposals are forwarded to members of the Common Tables Committee for review. The Common Tables Committee may reject a proposal, modify the proposal or accept the proposal. Complete and accepted proposals are collected for inclusion in the next revision of the *Discrete Devices Specification*. Once the Request Proposal for a particular enumeration is approved, the actual numeric value for the enumeration is chosen by members of the Common Tables Committee.

5.3. Releasing a Discrete Devices Specification Revision

Accepted change proposals are collected and periodically a new revision of the *Discrete Devices Specification* is generated. The actual release must complete the following steps:

- A proposed revision of the Discrete Devices Specification is generated and all approved change proposals are incorporated.
- The proposed revision is forwarded to all members of the Common Tables Committee for review and approval.
- Once delivery of the proposed revision is confirmed, committee members have 15 working days to approve or disapprove the modifications. All rejections must include conditions for acceptance. No response by that committee member will be considered an approval.
- Unanimous approval by Common Tables Committee membership is required to release the *Discrete Devices Specification* revision.

6. OVERVIEW

Traditionally the HART has supported only process automation devices. These devices provide continuous measurements or control actions and use floating-point values (along with engineering units and status) to communicate with host applications. However, even in process intensive applications, many discrete I/O (e.g, for solenoid valves or motor-starters) points are required. The Discrete Devices specified in this document are intended to complement existing HART Process Automation Devices.

For HART, intelligent Discrete Devices are a new class[2] of field devices that support discrete (on/off) inputs and outputs. These discrete I/O may be simple Boolean inputs (e.g., limit switches) and outputs (e.g. contact closures). Alternatively, the discrete may be 'smart' (e.g, flame detection, vibrating level switches) that include diagnostic capabilities.

A Discrete Field Device (see Figure 2) consists of Discrete Variables and (optionally) a Discrete Logic Execution Unit (DLEU). Discrete Variables are 16-bits (i.e., 1 word) wide and are similar to registers in a Programmable Logic Controller (PLC). Discrete Variable may include input (e.g., Discrete Variables 0 and 1) or output (e.g., Discrete Variable 3) connections to the plant or may have no physical connection (e.g., Discrete Variable 2) and be available for general use within the Discrete Field Device. Discrete Variables include both a value and status/quality information.

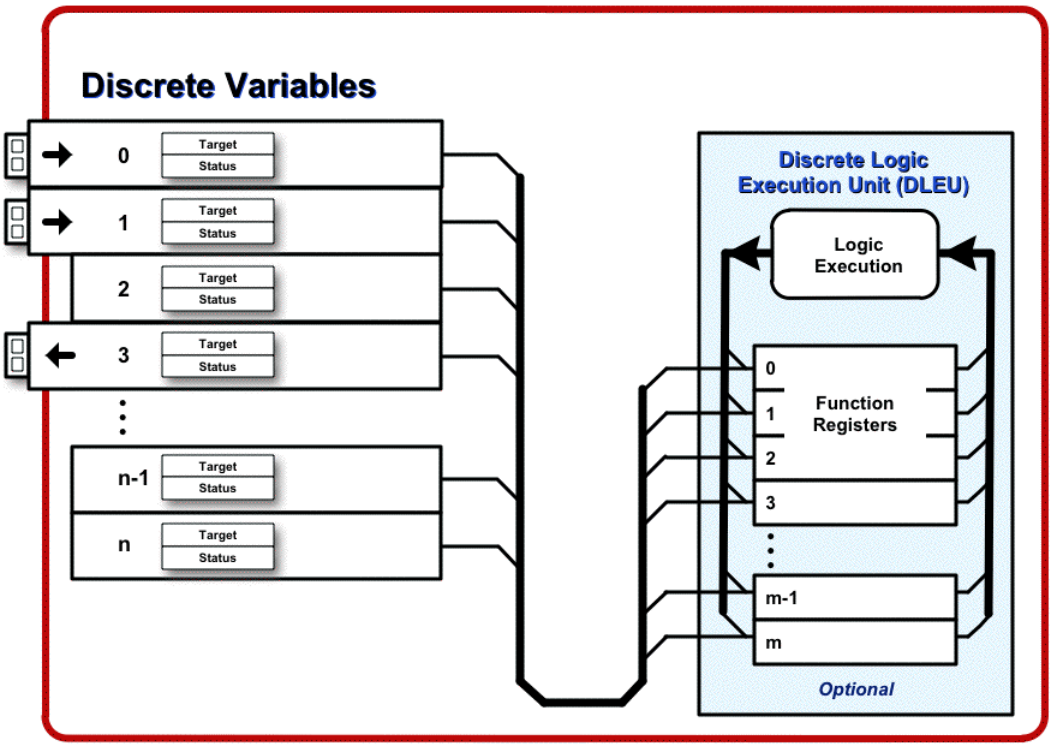


Figure 2. Discrete Field Device

The optional DLEU allows simple control actions to be performed local to the Discrete Field Device. In other words, the Discrete Field Device can be used to break the process into simpler modules and eliminate latency affects by providing fast reaction to events. The DLEU contains Function Registers used to control the Logic Execution. The Function Registers may reference a Discrete Variable or another Function Register and specifies a binary operation (e.g., AND, OR) to be performed.

For more powerful logic operations the DLEU may include counter/timers and drum registers. Counter/timers are 16/32 bits wide and can support a variety of timing functions (e.g., pulse timing, or on/off timer delay

[2] The Device Profile Code returned in identity commands (e.g., in the Command 0 response) indicates the type/class of the device (see Common Table 57).

functions). Operating as a counter allows events to be counted/totalized. When supported the drum registers allow complex sequences to be generated/controlled.

Requirements for all Discrete Devices are specified in the following Subsections:

- **Subsection 6.1, Discrete Devices**. A Discrete Field Device is one of the three types of Discrete Devices. This subsection defines all Discrete Device types and specifies requirements common to all three types.

- **Subsection 6.2, Discrete Variables**. All Discrete Devices must contain Discrete Variables and this subsection specifies the corresponding requirements. Like Process Variables, Discrete Variables include real-time values and associated properties (like status, classification, etc.)

- **Subsection 6.3, Discrete Logic Execution Unit** . In this Subsection, general operational requirements for the DLEU are specified along with full specifications for the standardized logic functions. The contents and properties of the Function Registers are also specified.

In the following Sections, detailed requirements are specified for Discrete and Hybrid Field Devices (Section 7) and Discrete Adapters (Section 8). Commands for Discrete Devices (Section 9) and common tables for Discrete Devices (Section 10) are specified as well.

6.1. Discrete Devices

Historically, HART has only directly supported Process Automation Devices. This document specifies three new types of devices:

- **Discrete Field Device**. A Discrete Device performs discrete (on/off) measurement, sensing or control actions.

- **Hybrid Field Device**. A Hybrid Device contains both Process Automation and Discrete Field Device capabilities.

- **Discrete Adapter**. A Discrete Adapter connects a PLC to a HART Network of I/O System. The Discrete Adapter provides access to the registers inside the PLC and allows the PLC program to be read/written via the HART Network using Block Data Transfer.

In Figure 3, a high-level model of all three types of Discrete Devices is shown. In addition, a block for Process Automation Device is shown for reference.

Process and Discrete Variables are the basis for communicating real-time data between HART Field Devices and Host Applications. Process Automation Devices must contain Process Variables (i.e., Dynamic and Device Variables) and Discrete Field Devices must contain Discrete Variables (see Subsection 6.2). Hybrid Field Devices must contain both Process and Discrete Variables. Discrete Devices may support loop signaling, and when they do, the Discrete Device may map a Discrete Variable to the loop current (input or output).

While Process Variables represent a continuous range of values, Discrete Variables must take-on distinct, Discrete Values (i.e., packed-Boolean or State values). Discrete Values must include the value along with status. In both Process and Discrete Variables, the "variables" are really a collection of real-time data and the properties associated with that real-time data. For example, all Discrete Variables may include a Fault Behavior in addition to the Discrete Values.

Discrete and Hybrid Devices may contain and DLEU to perform high-speed local logic processing. When included, the DLEU enables actions to be performed, in the field, with very low latency and without being delayed by communications to a centralized programmable automation controller.

When included, the DLEU must contain Function Registers and support the mandatory Logic Functions. In addition, the DLEU may support Counter/Timers and Drums. Counter/Timers are 32-bits wide and must include an Initial and Current Value. The DLEU may support multiple Drums and each Drum must contain an array of Drum Registers allowing complex sequences to be stepped through.

All Hybrid and Discrete Field Devices are recommended to support a DLEU, all Logic Functions, Counter/Timers and Drum logic.

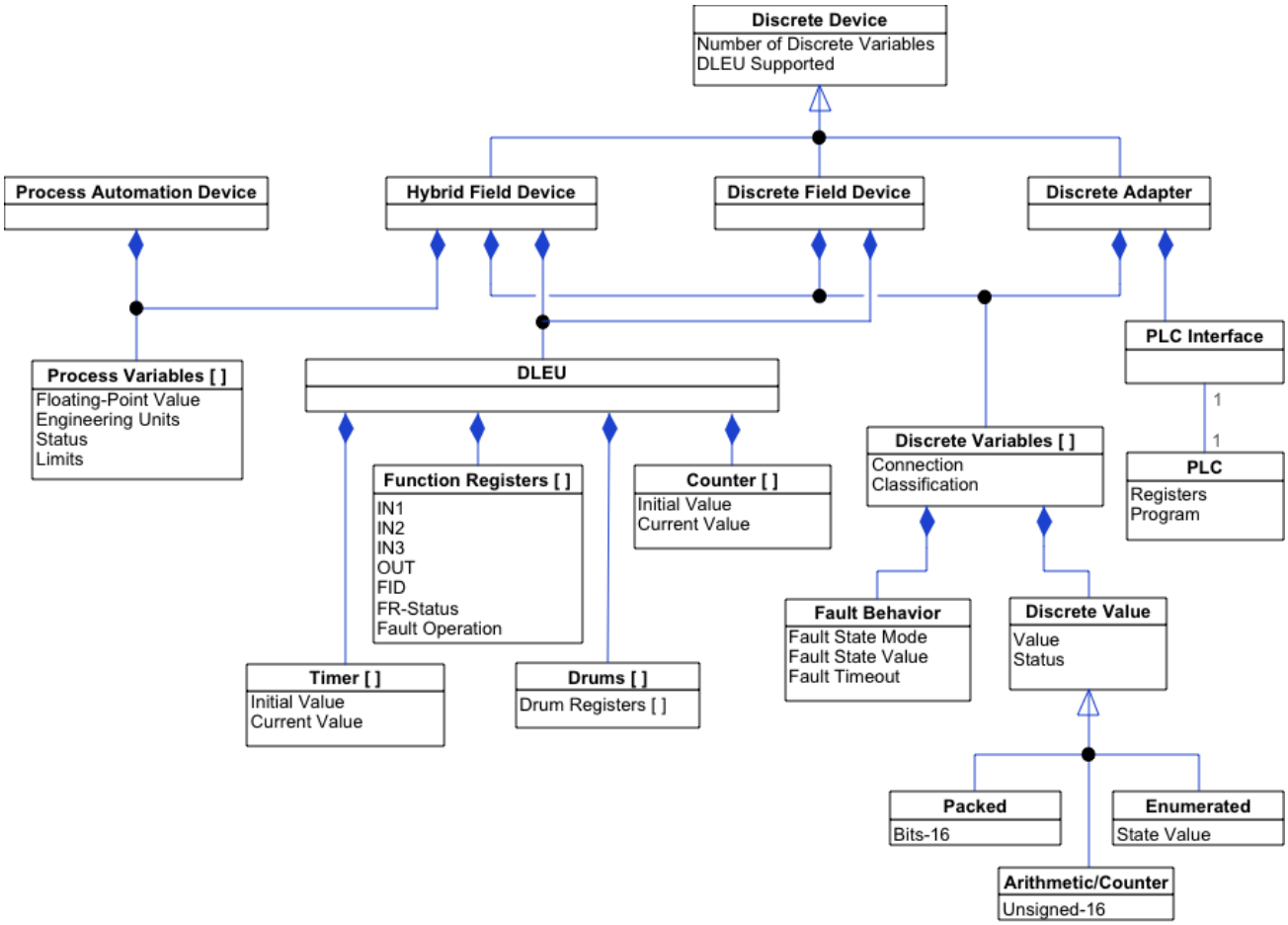


Figure 3. Discrete Device Model

Discrete Adapters must provide: access to registers in the attached PLC and Discrete Variables mapped to PLC Registers. Discrete Adapters may support reading/writing of the PLC program using Block Data Transfer. Discrete Adapters do not support a DLEU or their own physical discrete inputs and outputs.

6.1.1 Loop Current Support

Loop Current signaling should be supported by Wired Discrete Devices and may be supported by Wireless Discrete Devices[3]. When Current Loop signaling is supported

- Command 35, Write Primary Variable Range Values;
- Command 40, Enter/Exit Fixed Current Mode;
- Command 45, Trim Loop Current Zero; and
- Command 46, Trim Loop Current Gain

Must be supported.

Command 35 is used to set the 4 and 20mA points and, for Discrete Devices, URV is usually set to 1.0 and LRV to 0.0. PLCs often support math functions and may even support Analog I/O. Consequently, URV and LRV must be specified by the user (using Command 35) to meet plant requirements[4].

The Transfer Function normally returns "Linear".

6.1.2 Universal Command Support

All Discrete Devices must support all Universal Commands and Hybrid Field Devices must implement the Universal Commands as specified. In other words, Hybrid Field Devices must support (at least) Dynamic Variables and Command 1-3, 9, 14, and 15 must be implemented as specified.

Discrete Field Devices and Adapters may support Dynamic or Device Variables. When Dynamic or Device Variables are not supported in a Discrete Field Device or Discrete Adapter:

- PV Value must be set to "0x7F, 0xA0, 0x00, 0x00"; the Status must be set to 0x30, (i.e., Status = "Bad" and Limit = "Constant"); the Units Code must be set to "250" Not Used; and the Device Variable Classification set to "0", Not Yet Classified.

  These values must be returned in Command 1, 3, and 9[5].

- Command 3 must be truncated to return only PV.
- In Command 15, URV, LRV must be set to "0x7F, 0xA0, 0x00, 0x00"; the Units Code must be set to "250" Not Used; Damping must be set 0.0; and the Transfer Function must be set to 0.
- In Device Status byte, when loop current signaling is supported the Loop Current Fixed and Loop Current Saturated bits must be reset (0).

Aside from these modifications, for Discrete Field Devices and Discrete Adapters all Universal Commands must be implemented as specified.

[3] Wireless Discrete Devices must meet all WirelessHART requirements (e.g., support for a Maintenance Port when Current Loop signaling is not supported).

[4] In HART, the Loop Current is a communication channel that transmits a digital value using an analog signal.

[5] Classification and Status are not returned in Commands 1 and 3.

6.1.3 Discrete Device Address Map

A Discrete Device must contain Discrete Variables and can optionally contain a DLEU with Function Registers, Counter/Timer Registers and Drum Registers. To simplify programming of the DLEU Function Register inputs, the Discrete Device can be thought of as having up to 65,536 16-bit words. The addressing of these words are partitioned as follows:

Table 1 Discrete Device Address Map

Index/Address	Organization	Description	
0x0000-0x3FFF	16-bit Words	Discrete Variables (up to 16K words)	
0x4000-0xDFFF		Reserved	
0xE000-0xEFFF	16-bit Words	Function Registers (up to 4K words)	
0xF000-0xF1FF	32-bit Double Word Pairs	Timers-Target/Setpoint (up to512 words)	(non-volatile)
0xF200-0xF3FF	32-bit Double Word Pairs	Timers-Actual	
0xF400-0xF5FF	16-bit Words	Counters-Target/Setpoint (up to512 words)	(non-volatile)
0xF600-0xF7FF	16-bit Words	Counters-Actual	
0xF800-0xFFFE	16-bit Words	Drum Registers (up to 2K words)	(non-volatile)
0xFFFF	N/A	"Not Used"	

This address space is sparsely populated. For example, if a Discrete Device only contains 4 Discrete Variables then attempts to access Discrete Variable 6 may return "Invalid Selection" or "Set To Nearest Value" in the command response. To simplify Host Access, commands are provided to directly access the Discrete Variables, Function Registers, Counter/Timer Registers and Drum Registers.

While most of the registers are 16-bits wide, Timers are organized into two blocks of 32-bit double words. The first double word block is the Setpoint (or target) and the second double word block is the actual value associated with the counter timer. So, for example, Timer 0 (Preset) would map to address 0xF000-0xF001 with 0xF000 containing the most significant 16-bits of the Setpoint. The Actual Value of the Timer would map to 0xF200-0xF201.

Similarly, Counters are organized as two blocks of 16-bit words. Counter 0 (Preset) would map to address 0xF400 and the Actual value of the Counter would map to 0xF600.

Drum registers contain constants organized into 16-bit words. The registers can be used as inputs to the Drum functions or as constants that can be used as an input to any function.

Function Registers may be used to read (or write/force) the result of the DLEU execution. While the execution results are volatile the logic program (i.e., the configuration of each Function Register) is non-volatile.

Normally, the address map is only used to identify addresses when specifying inputs to functions in the DLEU.

6.2. Discrete Variables

All Discrete devices must contain Discrete Variables for communicating Discrete Values between a Host Application and the plant or plant equipment.

> **Discrete Variables** are uniquely defined data items within a Discrete Device for accessing Discrete Values. Each Discrete Variable must contain a Discrete Value and may include Fault Behavior. Each Discrete Variable also has properties (e.g., classification, type, etc.), which describe its behavior and capabilities.

Discrete Variables must take-on distinct, Discrete Values (e.g., packed-Boolean or State values). Discrete Values must include the value along with status (See Subsection 6.2.1). In addition, Discrete Variables may support a Fault Behavior (see Subsection 6.2.2). Fault Behavior allows the outputs from a Discrete Device to automatically assert a specified value upon a detecting a fault condition, Device Malfunction or when the Fault Timeout occurs.

A Discrete Variable has a connection type (e.g., input, output or general purpose). Input and output Discrete Variables have a physical connection the plant or plant equipment and general purpose Discrete Variables may contain (for example) intermediate results. Inputs must communicate the values determined by the field device (based on plant or equipment conditions) to a Host Application.

Outputs (i.e., "Output to plant" connections) are Discrete Values written by Host Applications to the field device (and therefore the output connection from the field device to the plant or plant equipment). When a target value is written (by the Host Application) to an output it may not take effect immediately[6]. Consequently, an output Discrete Variable may have a corresponding input Discrete Variable associated with it that returns the Actual Value. When this is the case, the index to Discrete Variable with the Actual Value must be indicated in Command 64,385, *Read Discrete Variable Properties*. The Actual Value must be the same type (e.g., packed-Boolean or State) as the output Discrete Variable.

Discrete Devices may support loop signaling, and when they do, the Discrete Device should map a Discrete Variable to the loop current (input or output). The mapping is fixed and, when present, indicated in Command 64,385.

The number of Discrete Variables contained in a Discrete Device is returned in Command 64,384. Discrete Variables must be numbered consecutively starting from zero (0).

Once an index is assigned to a Discrete Variable the index and the meaning of that Discrete Variable must not change for a given Expanded Device Type Code. The addition of a new Discrete Variable must result in an increment of the Device Revision and the major revision level of the manufacturer's device-specific document (i.e. a major Device Revision). Adding support for more classifications to a Discrete Variable (i.e., adding more State Tables) must result in an increment of the Device Revision.

6.2.1 Discrete Value and Status

The Discrete Value is the payload of the Discrete Variable. The Discrete Value communicates plant status and information to Host Applications and is used by Host Applications to write targets, setpoints and indicators to the Discrete Device. A **Discrete Value** must consist of two data-items: the Value and its associated Status.

> **Value**. The 16-bit Value must indicate the current value of the Discrete Variable as determined by the field device. This value may correspond to (for example) the value of a plant I/O connection, a discrete state, the value of a working register within the field device, or the result of DLEU execution.
>
> **Status**. Indicates the status of a Discrete Variable as determined by the field device.

The Discrete Variable Status is aligned with the Device Variable Status (See *Command Summary Specification*) and is defined in Figure 4. The most-significant two bits are defined to be the same as the Process Data Status. The next three bits (Limit Status and More Discrete Variable Status) must always be set to zero (0). In other words, Limit Status will always be "Not Limited" for Discrete Variables.

Field Definition								Description
Bit 7	6	5	4	3	2	1	0	
11 Good 01 Uncertain 10 Manual 00 Bad								Process Data Status (See *Command Summary Specification* for more information and specifications)
		0	0					Reserved (Limit Status). Must be set to 00
				0				More Discrete Variable Status. Must Be Set to 0
					X	X	X	Discrete Variable Specific Status (See Subsection 10.4 Table 4. Discrete Variable Specific Status)

Figure 4. Discrete Variable Status

Each Discrete Variable has a type (see Subsection 10.1 Table 1. Discrete Variable Types) and a classification (see Subsection 10.6 Table 6. Discrete Variable Classification Codes) that specify the format and values that

[6] For example, some actuators can take several seconds to actually close once commanded to do so.

must be assumed by the Discrete Value. When the Discrete Variable Type indicates "State" the Discrete Variable Classification associates a Discrete Variable with a specific Discrete Function Block (e.g., a motor starter or a solenoid valve). For all other Discrete Variable types the Discrete Variable Classification must return "Not Classified".

The initial value, and fault value of the Discrete Value is device-specific. The initial value and fault value of each Discrete Variable must be specified in the Field Device Specification (see the *Field Device Specific Specification Template*). The default initial and fault values for a Discrete Variable may be permanently fixed or user configurable.

Packed Discrete Variables
Packed Values consist of 16-bits each representing 16 independent Boolean values packed into 2-bytes. Each bit has an independent meaning and must only take on 2 values (0 = off, and 1 = on).

When the Discrete Values are of type Packed the Status applies to the underlying I/O connection (i.e., the status of each individual bit is not supplied). For a Packed Discrete Value, a status of "Bad" indicates a bad I/O Card or Module. Host Applications must consider all 16 bits to be bad.

When a Packed Discrete Variable has a Connection Code of "Output to plant" there may be an associated actual value. The actual value can be read to confirm the state of the connected plant equipment. For example, the output could be connected to a solenoid valve and the actual indicates whether it actually opened. Command 64,385 will contain the index to the Discrete Variable containing the actual value when it exists.

When the Discrete Variable Type indicates the Discrete Values are "Packed" the Discrete Variable Classification must indicate "Not Classified".

Arithmetic/Counter Discrete Variables
When the Discrete Variable Type indicates the Discrete Values are " Arithmetic/Counter" the Discrete Variable Classification must indicate "Not Classified". Arithmetic/Counter Values are unsigned 16-bit integers[7]. Arithmetic/Counter Discrete Variables contain inputs or outputs from, for example, DLEU operations, Counters, or Timers.

Arithmetic/Counter-type Discrete Variables should be Connection Type "No plant connection" (see Subsection 10.2 Table 2. Discrete Variable Connection Codes).

State Discrete Variables
For State-types the Discrete Variable Classification[8] specifies which discrete states that are valid for the Discrete Value[9] and specifies the Discrete Function Block for that Discrete Variable.

The Discrete Function Blocks that may be supported by a State type Discrete Variable are specified in Subsection 10.6 Table 6. Discrete Variable Classification Codes. The Discrete Function Block (and control sequence) currently specified is returned in Command 64,385 (Discrete Variable Classification Code). When the Discrete Variable Classification is in the range 240-249, the Discrete Function Blocks and state values that can be assumed are device-specific. Device-specific Discrete Function Blocks must be specified in the Field Device Specification (see the *Field Device Specific Specification Template*) along with the legal state values.

[7] Timers occupy two sequential Discrete Variables. The two Discrete Variables are concatenated when used in DLEU Timer Functions.
[8] When the Discrete Variable Type indicates the Discrete Values are "State" (i.e. enumerated) the Discrete Variable Classification must not indicate "Not Classified".
[9] i.e. the classification specifies the subset of values from Subsection 10.5 Table 5. Discrete Variable States that are valid.

The Discrete Function Block (and the State Discrete Variable associated with it) provides control for discrete devices such as switches, blocking valves, and simple motor starters. A Discrete Function Block may have several Discrete Variables associated with it. Figure 5 depicts a generic Discrete Function Block utilizing 4 Discrete Variables (Target State, Actual State, Inputs and Outputs).

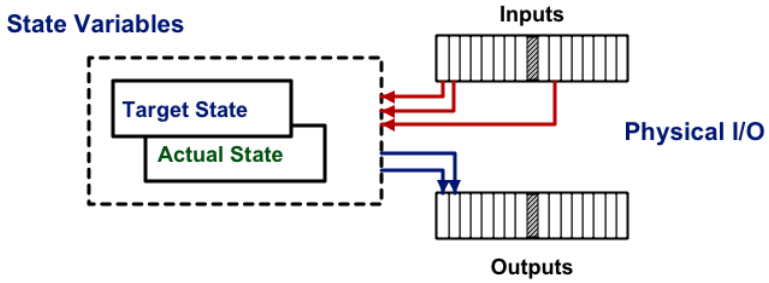


Figure 5. Discrete Variables in Discrete Function Block

Writing to "**Target State**" Discrete Variable changes or updates the setpoint for the Discrete Function Block. The Discrete Function Block reads in one or more input Discrete Variables (**Inputs**), performs its associated logic, and drives one or more output Discrete Variables (**Outputs**). The Inputs are read from, for example, actual switches or contacts connected to the plant. In turn, the Outputs are manipulated to perform the require control actions and adhere to the sequence specified for the Discrete Function Block[10].

The **Actual State** Discrete Variable[11] allows the Discrete Function Block sequencing to be monitored by the Host Application. The **Actual State** may include intermediate, transitional states that are sequenced through during Discrete Function Block operation (see Subsection 10.6 Table 6. Discrete Variable Classification Codes). In other words, when a target is written to output Discrete Variable a control sequence is triggered. The Discrete Function Block then sequence through the control sequence as event and status feedback is received from the connected equipment. Ultimately the sequence will complete and Discrete Function Block becomes quiescent.

In general, the Field Device compares the **Target State** to the **Actual State** and, after allowing time (Transition Time) for the device to change state, may generate an error if the **Actual State** does not match the Target State.

State Discrete Variables, based on the Discrete Variable Classification, supports a list of enumerations that are a subset of Table 5. Discrete Variable States[12] (Subsection 10.5). The classification indicates the Discrete Function Block specified for the State Discrete Variable. Each Discrete Function Block is specified in subtables to Subsection 10.6 (e.g., Table 6.1, Table 6.2, etc.). Each Discrete Function Block specification includes mandatory and optional requirements. Consequently, a State Discrete Variable may be simply providing translations between the enumerations and the low level I/O performed by the device.

[10] i.e., In Discrete Functions Blocks the State Variables are not (generally) directly connected to the plant. The actual plant connections are via the raw, physical I/O (Inputs and Outputs).

[11] Command 64,385, *Read Discrete Variable Properties* contains the index to the "**Actual State**" Discrete Variable.

[12] Actual State normally supports a longer list of enumerations then Target State does.

Depending on the Discrete Variable Classification (and device design) the Discrete Function Block may not support (or expose) all the Discrete Variables shown in Figure 5. For example, Figure 6 shows a simple State Discrete Variable that uses a switch as an input with no communication access to physical signals. For "Input from plant" connections, State Discrete enumerations shall be the list of actual values associated with the Discrete Function Block[13].

Figure 6. Simple Input State Discrete Variable

So, when configured as an "On/Off Switch" the enumerations are "On" and "Off" (see Subsection 10.6 Table 6. Discrete Variable Classification Codes). In this example, the State Discrete Variable may be configured as "Limit Switch" with enumerations of "Opened" and "Closed".

6.2.2 Fault Behavior

Discrete Variables that are outputs to the plant (see Subsection 10.2 Table 2. Discrete Variable Connection Codes) must support user-configurable Fault Behavior. All other Discrete Variables may support Fault Behavior. Process Data Status of "Manual/Fixed" must be set in the Discrete Variable Status Byte to indicate a fault has occurred. When supported, the Fault Behavior attributes specify the response of the Discrete Variable to the fault occurring in its input or control path. The Fault Behavior attributes are:

Fault State Mode. Specifies the behavior (e.g., hold last value; or assert Fault State Value) of the Discrete Variable under fault-state conditions.

Fault State Value. Specifies the value to be asserted when the Discrete Variable is in the fault condition[14].

Fault Timeout. The shed time that must lapse before a fault condition causes the fault state action to be executed.

There are a number of conditions or events that can cause a fault to occur Including: device malfunction; outputs not being updated from the controller; fault in the Discrete Variable itself (e.g., an output sensing a field-side failure); or Bad status being set by the DLEU.

The Fault Timeout is used to minimize nuisance faults and, in the case of an output Discrete Variable, used as the communication shed timeout. If the timeout reaches 0 the Process Data Status must be set to "Manual/Fixed" and the Fault State Mode must be asserted[15]. When Fault Timeout is set to 0, the Fault State Value will be immediately asserted upon fault detection.

The Field Device Specification (see the *Field Device Specific Specification Template*) for the Discrete Device must indicate which Discrete Variables support Fault Behavior[16] and must state the conditions that activate the fault state for each Discrete Variable.

6.2.3 Mapping Discrete Variables to Dynamic or Device Variables

If Current Loop Signaling is supported then the Discrete Device must support at least one Dynamic Variable (i.e., Device Variable Code 246, "Primary Variable or PV"). If the Discrete Device is a Hybrid Field Device then it may support other Device Variables as well. Device Variable Index (see Command 64,385) indicates the mapping of the Discrete Variable.

[13] Intermediate, transitional enumerations may not be reported. For example, the transition could happen fast enough to not be detected.

[14] The Fault State Mode may affect the application of the Fault State Value.

[15] Any Discrete Variable with fault state asserted must set "Discrete Variable in fault mode" in the Discrete Variable Specific Status (See Subsection 10.4 Table 4. Discrete Variable Specific Status)

[16] If a Discrete Variable does not support Fault behavior then the Fault State Mode must return "251", None.

- When the Discrete Variable is mapped to a Device Variable, Device Variable Index must be in the range 0-240 and must not change for the life of that Device Type[17].

- When the Discrete Variable is mapped to a Dynamic Variable then the Device Variable Index must be in the range 246-249. The mapping may be changed (if the Discrete Device supports re-mapping) using Command 64,393.

- When not mapped to a Device Variable or Dynamic Variable the Device Variable Index must return 251 (None).

Hybrid Field Devices must only support mapping of Discrete Variables to a Device Variable if that device exposes Device Variables (see *Command Summary Specification*). If the Hybrid Device only supports Dynamic Variables then the device must only support mapping to a Dynamic Variable.

Discrete Field Device or Discrete Adapter must only support mapping to a Dynamic Variable. If Current Loop Signaling is supported in a Discrete Field Device or Discrete Adapter a Discrete Variable must be mapped to each Dynamic Variable supported.

6.3. Discrete Logic Execution Unit

The DLEU enables the execution of logic sequences within a Discrete or Hybrid Field Device. While the DLEU is not intended to replace PLCs or PACs, it does allow local control action to be performed without encountering the latency communication (across the HART network to the controller) entails. In other words, the Discrete or Hybrid Field Device can be used to break the automation system into simpler modules and eliminate latency affects by providing fast local reaction to events. Operation can be monitored using standard HART practices (e.g., using Burst Mode).

The DLEU is simplified logic solver consisting of

- An array of Function Registers (essentially the logic program); and

- The Logic Execution Unit that sequences through the Function Registers executing the specified logic functions.

While the DLEU is optional, the minimum requirements for a DLEU are

- At least 8 Function Registers;

- All Logic Functions must be supported;

- If Counter Functions are supported then at least 4 Counter Registers must be supported;

- If Timer Functions are supported then at least 4 Timer Registers must be supported; and

- If Drum/MUX Functions are supported then at least 4 Drum/MUXs with at least 16 values for each must be supported

If the DLEU is supported the actual level of support must be indicated in Command 64,448.

6.3.1 DLEU Operation

The DLEU contains Function Registers used to control the Logic Execution. The Function Registers may reference a Discrete Variable or another Function Register and specifies the, for example, logic operation (AND, OR, etc) to be performed.

For more powerful logic operations the DLEU may include counter, timers and drum registers. Counters allow events to be counted/totalized. Drum/MUX functions allow relatively complex sequences to be generated/controlled.

[17] The meaning of a Device Variable must never change.

Timers are unique in that they are 32 bits wide and support a variety of timing functions (e.g., pulse timing, or on/off timer delay functions). Timer setpoints use the standard HART Time data type (see *Command Summary Specification*).

The Logic Execution Unit sequentially scans the Function Registers (FR) executing the function specified in each. When the last FR is has been scanned the current scan cycle is complete and the Logic Execution Unit begins the next scan cycle starting with the first FR. The scan cycles repeat as long as the DLEU is Enabled.

Should the DLEU be Disabled the Logic Execution Unit is halted. When the DLEU becomes Enabled the scanning is restarted from the first FR.

6.3.2 Function Registers

Function Registers (FR) are atomic programmable elements within the DLEU. The Function Register (see Figure 7) specifies the function to be performed; the inputs and outputs to that function; the Fault Behavior to perform when an error is detected; and the status of the FR. FRs consist of

- Three inputs (IN1, IN2 and IN3)
- An output (OUT);
- A Function Code (FCODE);
- Failure Behavior (FB); and
- Status (FSTATUS).

FRs are accessed via their Index. FRs must be numbered consecutively starting from zero (0).

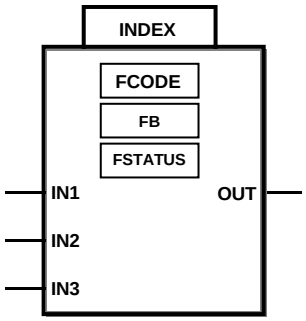


Figure 7. Function Register

Each FR contains a Function Code that defines the operation to be performed. Once configured and enabled, the DLEU executes, in order, the list of configured FRs. The FRs are evaluated and the Function Code performed once per scan cycle starting from FR[0]. The scan cycle is complete when the last FR supported by the field device is encountered. Collectively, the sequencing of the FRs and execution of their Function Codes support the defined/required control strategy.

The inputs to and output from each FR are specified as Discrete Variable indices[18] (see Table 1 Discrete Device Address Map). Both the value and status of the Discrete Variable are used when the FR is evaluated.

Inputs

IN1, IN2, IN3 specify the input data for the FR. They may specify, for example, another FR or a Discrete Variable (see Table 1 Discrete Device Address Map). If an input specifies another FR then the value and status from the FR is used as the input. This allows FRs to be cascaded together thus allowing complex logic and sequences to be specified.

If an input is not used it must be set to 0xFFFF.

[18] There is one exception: when the FR is programmed with the CONST function IN1 is a value not a reference (i.e. index) to a Discrete Variable

Output
OUT specifies the destination[19] for the result (value, status) of the function execution. The value assigned to OUT depends on the inputs and the function performed.

The status of the OUT depends on the status of the inputs[20] and must be set to the worst status of any input. Severity of status is ranked from worst to best as BAD, UNCERTAIN, and GOOD (in that order).

When not used, OUT must be set to 0xFFFF.

Function Code
DLEU functions are specified in Subsection 6.3.3 and categorized as Logic, Counter, Timer, and Drum/Mux functions. The DLEU must support all Logic Functions. Support for Counter, Timer and Drums/MUX Functions is strongly recommended. The FCODE specifies how the inputs are combined/used to update/generate the output from the FR. Upon reset the Function Codes in all FRs are reset to the END function (code 0xFFFF).

Fault Behavior
This parameter is set to describe how the DLEU should treat bad status indications on an input (see Subsection 10.12 Table 12. Function Register Fault). Normally, when a function is evaluated, and any of the inputs is BAD, OUT (and the specified Discrete Variable) will be set to BAD status. Irrespective of the status, the OUT value calculated normally. Setting the Fault Behavior can ignore one or more of the inputs' status.

Note - The status of the input to the ISBAD() function is always ignored regardless of the Fault Behavior setting.

Function Register Status
This enumerated value provides diagnostic information for the Function Register. Refer to Subsection 10.11 Table 11. Function Register Status.

6.3.3 DLEU Function Reference
This subsection provides the detailed specification of the functions available to the DLEU. Details on the Logic, Counter, Timer and Drums/MUX Functions are included in the following subsections. All Discrete and Hybrid Field Devices that support a DLEU must support all Logic Functions.

Table 9. DLEU Function Codes (see Subsection 10.9) provides a list of functions sorted by Function Code and Annex D provides an index to the functions sorted by name.

Logic Functions
In cases where the input is a packed 16-bit structure, the logic will be performed on the whole value.

All DLEU must implement all Logic Functions. Generally, logic functions are performed on each bit within the value mapped to the FR inputs. For example, if an AND FR is executed with IN1 value of 0x59, and IN2 value of 0x67, the output value would be 0x41 (see example below). The 2BOOL() FR is provided for the purpose of evaluating an entire byte as a single Boolean FALSE (0x0000) or TRUE (0xFFFF), instead of performing the standard bitwise logic.

0x59 0**1**01 100**1**

0x67 0**1**10 011**1**

0x41 0**1**00 000**1**

Figure 8. Example AND operation

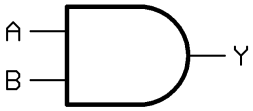
[19] The destination must not be another FR.
[20] There is one exception: when the FR is programmed with the ISBAD function OUT must always have a status of GOOD.

While the figures and truth tables for AND, OR, NAND, NOR, and XOR only show two inputs all three inputs are supported and ternary operation is performed as usual for these logic functions.

AND
The AND logic function performs a bit-wise evaluation of IN1 and IN2. If a specific bit is TRUE (logic 1) in both IN1 and IN2 then the corresponding OUT bit will be TRUE (logic 1). If IN1 or IN2 (or both) are FALSE (logic 0) then the OUT bit will be FALSE (logic 0).

For example, if IN1 is 01010011 and IN2 is 01111001, the value in the OUT when the function is finished will be 01010001.

Logic Symbol **Truth Table**



IN1	IN2	OUT
0	0	0
0	1	0
1	0	0
1	1	1

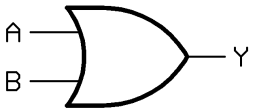
Figure 9. AND Function

Any unused input value must be tied to TRUE or the input set to Discrete Variable index 0xFFFF.

OR
OR logic function. The OR logic function a bit-wise evaluation of IN1 and IN2. If a bit in IN1 OR IN2 is TRUE then the corresponding bit in OUT will be TRUE (logic 1). Otherwise the OUT will be FALSE (logic 0).

For example, if IN1is 01010011 and IN2 is 01111001, the value in the OUT when the function is finished will be 01111011.

Logic Symbol **Truth Table**



IN1	IN2	OUT
0	0	0
0	1	1
1	0	1
1	1	1

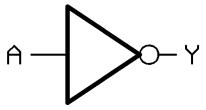
Figure 10. OR Function

Any unused input must be tied to FALSE or the input set to Discrete Variable index 0xFFFF.

NOT
NOT is a unary logic function that it operates on the state of IN1. If a bit in IN1 is FALSE the corresponding OUT bit will be TRUE.

For example, if IN1is 01010011, the value in the OUT when the function is finished will be 10101100.

Logic Symbol



Truth Table

IN1	IN2	IN3	OUT
0	Don't Care	Don't Care	1
1			0

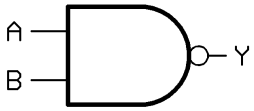
Figure 11. NOT Function

IN2 and IN3 are ignored.

NAND
This is the bit-wise 'not AND' evaluation of IN1, IN2 and IN3. If an input is not used it must be set to 0xFFFF. If a specific bit in IN1 IN2 and IN3 are all TRUE the corresponding OUT will be FALSE. All other combinations evaluate to TRUE.

For example, if IN1is 01010011 and IN2 is 01111001 (IN3 is not used), the value in the OUT when the function is finished will be 10101110.

Logic Symbol



Truth Table

IN1	IN2	OUT
0	0	1
0	1	1
1	0	1
1	1	0

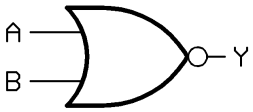
Figure 12. NAND Function

Any unused input must be tied to TRUE or the input set to Discrete Variable index 0xFFFF.

NOR
This is the 'not OR' logical function. The NOR performs a bit-wise evaluation of IN1 and IN2. If a specific bit in IN1 and IN2 are both FALSE the corresponding OUT will be TRUE. All other combinations evaluate to FALSE.

For example, if IN1is 01010011 and IN2 is 01111001, the value in the OUT when the function is finished will be 10000100.

Logic Symbol



Truth Table

IN1	IN2	OUT
0	0	1
0	1	0
1	0	0
1	1	0

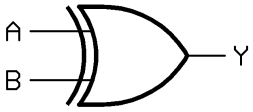
Figure 13. NOR Function

Any unused input must be tied to FALSE or the input set to Discrete Variable index 0xFFFF.

XOR
The XOR logic function evaluates the state of two integers to see if IN1 and 2 are equal. If either IN1 OR 2 is TRUE - the corresponding OUT bit will be TRUE (logic 1). If both IN1 and 2 are TRUE (logic 1) - the OUT will be FALSE (logic 0). If both IN1 and 2 are FALSE (logic 0) - the OUT will be FALSE (logic 0).

For example, if IN1is 01010011 and IN2 is 01111001, the value in the OUT when the function is finished will be 00101010.

Logic Symbol



Truth Table

IN1	IN2	OUT
0	0	0
0	1	1
1	0	1
1	1	0

Figure 14. Exclusive-OR Function

Any unused input must be tied to FALSE or the input set to Discrete Variable index 0xFFFF.

2BOOL
Convert Integer to Bool. 2BOOL is a unary logic function – it operates on the state of IN1. The 2BOOL function translates the input parameter from an 16-bit value to a simple TRUE or FALSE value. If the IN1 is any value except zero, the OUT is set to TRUE (0xFFFF). This is principally used when the user wants to refer to the input as a 16-bit packed value in one FR and as a simple Boolean in another.

IN2 and IN3 are ignored.

NOP
Not Configured. The function register is not configured, inputs will be ignored, and OUT is FALSE. Function Register Status will indicate 0 (not executed) for all NOP Function Registers.

All inputs are ignored.

BITS
Pick one bit out of a packed byte. This function takes IN1, and compares it to a mask defined in IN2. If the AND of IN1 and the mask is non zero, the OUT is set to TRUE. Otherwise the OUT is FALSE. IN3 is ignored.

FTRIG
Falling Edge Trigger. The FTRIG is a function that may be used to trigger another function on the falling edge of a transition on the input. When IN1 detects a TRUE to FALSE transition, the OUT is set to TRUE for one scan period.

IN2 and IN3 are ignored.

RTRIG
Rising Edge Trigger. The RTRIG is a function that may be used to trigger another function on the rising edge of a transition. When IN1 detects a FALSE to TRUE transition, the OUT is set to TRUE for one scan cycle.

IN2 and IN3 are ignored.

LATCH
Latched Coil. When IN1 moves to a TRUE state, the OUT latches to a TRUE. The OUT maintains its latched state until IN2 resets the latch.

If both IN1 and IN2 are TRUE the OUT will remain latched. IN3 is ignored.

UNLATCH
Unlatch Coil. When IN1 moves to a TRUE state, the OUT unlatches to a FALSE (value of 0). The OUT maintains its unlatched state until IN2 resets the latch.

If both IN1 and IN2 are TRUE the OUT will remain unlatched. IN3 is ignored.

ISBAD
Check status of input. ISBAD is a unary logic function – it operates on the state of IN1. The ISBAD function generates a TRUE value on its OUT when the status of IN1 is BAD. Status of OUT is always good regardless of the input status.

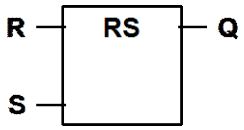
CONST
Constant value. A constant value as defined by IN1.

IN2 and IN3 are ignored.

RS

Reset Dominate Bistable. The RS function (sometimes referred to as an RS Flip-Flop) acts as a reset dominant bistable. If the Set Input (IN1) is TRUE, the OUT is TRUE. A TRUE on the Reset (IN2) line sets the OUT to FALSE (regardless of the Set (IN1) input state). IN3 is ignored.

Logic Symbol



Truth Table

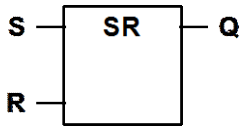
IN1 'S	IN2 'R'	IN3	OUT 'Q' (Present)	OUT 'Q' (Final)
0	0	Don't Care	0	0
0	0		1	1
0	1		0	0
0	1		1	0
1	0		0	1
1	0		1	1
1	1		0	0
1	1		1	0

Figure 15. RS Flip-Flop

SR

Set Dominate Bistable. The SR function (sometimes referred to as an SR Flip-Flop) acts as a set dominant bistable with an initial state of FALSE. If the Set Input (IN1) is TRUE, the OUT is TRUE. A TRUE on the reset (IN2) line sets the OUT to FALSE only if the set (IN1) input is also FALSE. IN3 is ignored.

Logic Symbol



Truth Table

IN1 'S'	IN2 'R'	IN3	OUT 'Q' (Present)	OUT 'Q' (Final)
0	0	Don't Care	0	0
0	0		1	1
0	1		0	0
0	1		1	0
1	0		0	1
1	0		1	1
1	1		0	1
1	1		1	1

Figure 16. SR Flip-Flop

Counter Functions
Counters are 1-word (16 bits) wide and, when supported, at least 4 counter registers must be supported. All functions in this section use IN3 to specify the counter register number (0, 1, 2, etc.). Table 2 shows the mapping of counter registers to the Discrete Variable address map.

Table 2. Counter registers, Input and Output Addresses

Register	Input	Actual
0	0xF400	0xF600
1	0xF401	0xF601
2	0xF402	0xF602
. . .		
n	0xF400 + *n*	0xF600 + *n*

Counter functions are optional. However, if they are supported then all functions in this subsection must be supported.

CTD
Down Counter. CTD counts input pulses in the down direction. While R (IN2) is TRUE the counter is set to the Target (IN3) value and Q (OUT) is set to FALSE. Once R is FALSE the counter is enabled to begin counting. When C (IN1) detects a rising edge transition the counter is decremented by one. Actual indicates the current count.

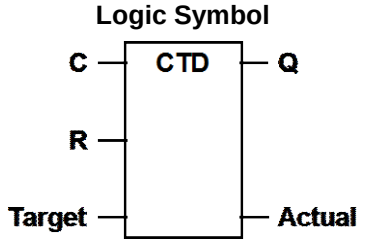


Figure 17. CTD Down Counter

When the counter reaches zero, Q (OUT) is set to TRUE and counting stops.

CTU
Up Counter. CTU counts input pulses in the up direction. While R (IN2) is TRUE the counter Actual value is held to zero and Q (OUT) is set to FALSE. Once R is FALSE the counter is enabled to begin counting. When C (IN1) detects a rising edge transition the counter is increment by one. Actual indicates the current count.

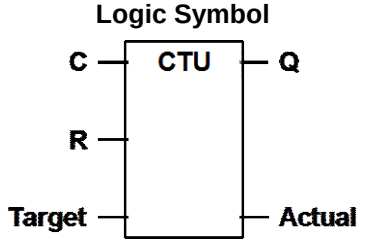


Figure 18. CTU Up Counter

When the counter reaches the Target (IN3) value, Q (OUT) is set to TRUE and counting stops.

Timer Functions
Timers are 32 bits wide and support a variety of timing functions (e.g., pulse timing, or on/off timer delay functions). When supported at least 4 timer registers must be supported. All functions in this section use IN3 to specify the timer register number (0, 1, 2, etc.). Table 3 shows the mapping of timer registers to the Discrete Variable address map.

Table 3. Timer registers, Input and Output Addresses

Register	Input	Actual
0	0xF000	0xF200
1	0xF002	0xF202
2	0xF004	0xF204
. . .		
n	0xF000 + 2**n*	0xF200 + 2**n*

Timer setpoints use the standard HART Time data type (see Command Summary Specification).

Timer functions are optional. However, if they are supported then all functions in this subsection must be supported.

TOFF
Off Delay Timer. While EN (IN1) is TRUE Q (OUT) is set to TRUE and the timer is reset. When EN becomes FALSE the timer starts. When the elapsed time is equal to the Preset (IN3) value Q is set to FALSE and the timer stops.

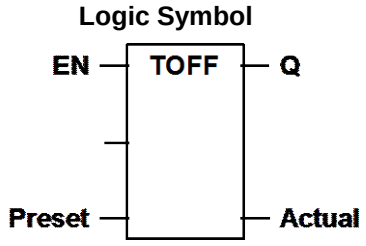


Figure 19. Off Delay Timer

Whenever EN is TRUE Q is TRUE. IN2 is ignored.

TON
On Delay Timer. When EN (IN1) detects a rising edge transition the timer is reset and begins timing. When the elapsed time is equal to the Preset (IN3) value Q (OUT) is set to TRUE and the timer is stopped. IN2 is ignored.

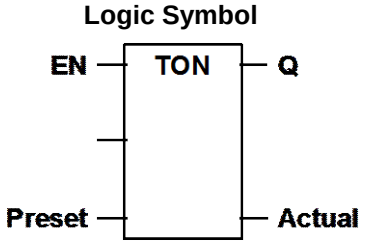


Figure 20. On Delay Timer

Whenever EN is FALSE Q is FALSE and the timer is reset. IN2 is ignored.

TIMER

Timer. When EN (IN1) detects a rising edge transition, the timer resets and begins timing from zero. When EN detects a falling edge transition, the current timer value is latched. IN3 specifies the Timer Register to be used (e.g., to read the Actual timer value).

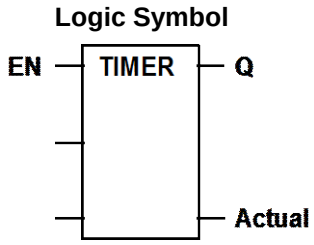


Figure 21. Timer

Q (OUT) is TRUE while the timer is running. IN2 is ignored.

PULSE

Pulse Generator. When Trigger (IN1) detects a rising edge transition the pulse generation timer is triggered and Q (OUT) is set to TRUE. Once triggered the pulse generation timer runs to completion (i.e., Trigger is ignored while the generator is running). Once complete Q is set to FALSE.

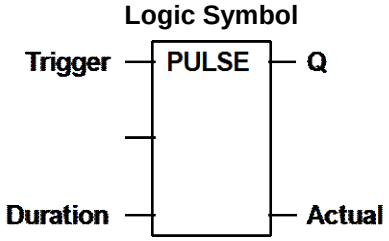


Figure 22. Pulse Generator

Duration (IN3) specifies the length of the pulse. IN2 is ignored.

CLOCK

Clock Generator. When EN (IN1) is TRUE Q (OUT) provides a square wave output toggling between TRUE and FALSE with a 50% duty cycle. Q will stay TRUE for the specified Duration (IN3) then toggles to FALSE for another interval of Duration.

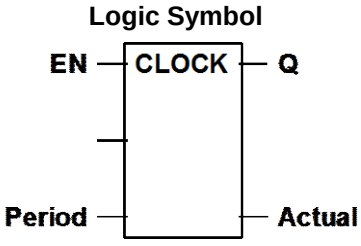


Figure 23. Clock

When EN is FALSE Q shall be FALSE. IN2 is ignored.

Drum/MUX Functions
Drum/MUX functions allow moderately complex sequences to step through. The number of Drum/MUXs (*N*) supported and the length (L) of each Drum/MUX register-array is indicated in the Read Discrete Device Capabilities Command. IN3 in Drum/MUX functions indicates the Drum/MUX being used. Table 4 shows the mapping of Drum/MUX to the Discrete Variable address map.

Table 4. Drum/MUX registers, Input and Output Addresses

Drum/MUX	First Address
0	0xF800
1	0xF800 +*L*
2	0xF800+ (2 * *L*)
. . .	
n	0xF800+ (*N* * *L*)

Drum/MUX functions are optional. However, if they are supported then all functions in this subsection must be supported.

DRUM_SEQ
Drum Sequencer. The DRUM_SEQ function steps through a pre-programmed set of values stored in the drum registers. While IN2 is TRUE (or Drum Control is set to "Drum Disabled") the drum sequencer is held to first step and OUT contains the first value in the drum register array (i.e., register-array [0]). When IN2 is set to FALSE drum stepping is enabled. When IN1 is pulsed, the drum sequencer counter is incremented, and the value of the OUT is set to the value at that offset in the Drum/MUX register-array. The counter is reset to 0 when it exceeds length of the register-array.

IN3 indicates the Drum/MUX being used.

MUX
Selection of input. The MUX function is used to select from pre-stored values. The OUT reflects the selected MUX value. IN1 is used as the index into the corresponding drum register-array and determines which MUX value is transferred to OUT. IN1 is constrained to index into the corresponding register-array In other words, the modulo function is used constrain IN1 to a valid offset into the corresponding Drum/MUX register-array.

IN3 indicates the Drum/MUX being used. IN2 is ignored

7. DISCRETE AND HYBRID FIELD DEVICES

Discrete Field Devices have one or more input, output or internal Discrete Variables. Hybrid Field Devices support Discrete Variables and Process Variables (i.e., Device Variables).

Discrete and Hybrid Field Devices may or may not support traditional current loop signaling[21]. When they do not have a traditional loop current connection, Discrete and Hybrid Field Devices must have a maintenance port.

Both Discrete and Hybrid Field Devices must support all Universal Commands and may be line, loop, or battery powered or they may be powered in some other fashion. Discrete and Hybrid Field Devices are connected to Process or Plant Equipment.

7.1. General Requirements

All devices must support all Universal Commands. Hybrid Field Devices support the Universal Commands in the normal fashion.

Discrete-only Field Devices must map at least one Discrete Variable to the Dynamic Variables (i.e., PV). The Discrete Variables can be identified using Command 64,385. For a simple device the map may be fixed (along with the range values, etc.) For wired Discrete Devices the loop current may simply switch between 4 and 20 milliamps (alarm levels can be signaled with the Loop Current as usual.

Wireless Discrete and Hybrid Field Devices may not support a Loop Current connection. When that is the case Loop Current must return 0x7F, 0xA0, 0x00, 0x00; the Status must be set to 0x30, (i.e., Status = "Bad" and Limit = "Constant") and the Device Status-"Loop Current Fixed" bit must remain reset.

7.2. Requirements for Discrete Field Devices

In addition to Universal Commands, Discrete Field Devices must support the Common Practice Commands specified in Table 5.

Table 5. Required Common Practice Commands for Discrete Field Devices

Common Practice

Cmd	Description
41	Perform Self Test
42	Perform Device Reset

Cmd	Description
59	Write Number Of Response Preambles
90	Read Real-Time Clock

Wired Discrete Field Devices must provide an analog interface to measure or control the loop current and must support Common Practice Commands 35, 40, 45, 46 and 79. Wired Discrete and Hybrid Field Devices should implement Command 34. Furthermore, all Discrete Field Devices should implement the Common Practice Commands listed in Table 6.

[21] Wired devices must support loop current signaling. Loop current signaling is optional for Wireless devices.

Table 6. Recommended Common Practice Commands for Discrete Field Devices

Common Practice

Cmd	Description
71	Lock Device
72	Squawk
73	Find Device
76	Read Lock Device State
78	Read Aggregated Commands
95	Read Device Communications Statistics
103	Write Burst Period
104	Write Burst Trigger
105	Read Burst Mode Configuration

Cmd	Description
106	Flush Delayed Response Buffers
108	Write Burst Mode Command Number[22]
109	Burst Mode Control
115	Read Event Notification Summary
116	Write Event Notification Bit Mask
117	Write Event Notification Timing
118	Event Notification Control
119	Acknowledge Event Notification

In addition to Universal and Common Practice Commands, the Field Device must support the standardized commands and procedures defined for Discrete Devices.

7.3. Requirements for Hybrid Field Devices

Hybrid Field Devices support both (digital) discrete transducers and (analog) process transducers. Consequently Hybrid Field Devices must support both Process and Discrete Variables. For example, a Hybrid Field Device could have Discrete Variables and Process Variables defined as follows:

Discrete	Description
0	Pressure Low Limit
1	Pressure High Limit

Process	Description
PV	Process Pressure
SV	Process Temperature
TV	Pressure Low Limit
QV	Pressure High Limit

This mapping would allow both the Discrete and Process Variables to be read in one command transaction. In this case, the mapping of the Discrete Variables to Process Variables could be fixed.

As with traditional HART Field Devices, the Dynamic Variables mapping can be configurable. This would allow Discrete Variable 0 to be mapped to PV, in turn driving the analog output signaling based on the state of Discrete Variable 0. The floating point value of such Process Variables would be set to the value of the corresponding Discrete Variable. Transducer limits and PV range values would likely be set to the minimum and maximum values obtainable by the Discrete Variable being mapped (e.g., LRV = 0.0 and URV = 1.0).

In addition, all Hybrid Field Devices must support the Common Practice Commands found in Table 7 and should support those found in Table 8.

[22] If Burst Mode is supported then the Field Device must support Command 64,395 and publishing of Command 64,386

Table 7. Mandatory Common Practice Commands for Hybrid Field Devices

Common Practice

Cmd	Description
41	Perform Self Test
42	Perform Device Reset
54	Read Device Variable Information

Cmd	Description
59	Write Number Of Response Preambles
90	Read Real-Time Clock

Wired Hybrid Field Devices must provide an analog interface to measure or control the loop current and must support Common Practice Commands 35, 40, 45, 46 and 79. Wired Hybrid Field Devices should implement Command 34.

Table 8. Recommended Common Practice Commands for Hybrid Field Devices

Common Practice

Cmd	Description
52	Set Device Variable Zero
53	Write Device Variable Units
55	Write Device Variable Damping Value
71	Lock Device
72	Squawk
73	Find Device
76	Read Lock Device State
79	Write Device Variable
80	Read Device Variable Trim Points
81	Read Device Variable Trim Guidelines
82	Write Device Variable Trim Point
83	Reset Device Variable Trim
91	Read Trend Configuration
92	Write Trend Configuration
93	Read Trend

Cmd	Description
95	Read Device Communications Statistics
96	Read Synchronous Action
97	Configure Synchronous Sampling
103	Write Burst Period
104	Write Burst Trigger
105	Read Burst Mode Configuration
106	Flush Delayed Response Buffers
107	Write Burst Device Variables
108	Write Burst Mode Command Number[23]
109	Burst Mode Control
115	Read Event Notification Summary
116	Write Event Notification Bit Mask
117	Write Event Notification Timing
118	Event Notification Control
119	Acknowledge Event Notification

[23] If Burst Mode is supported then the Field Device must support Command 64,395 and publishing of Command 64,386

7.4. Requirements for Wireless Discrete and Hybrid Field Devices

All Field Devices must support all Universal Commands. In addition, all Wireless Discrete and Hybrid Field Devices must support the Common Practice Commands found in Table 9 and should support those found in Table 10.

Table 9. Mandatory Common Practice Commands for Wireless Discrete and Hybrid Field Devices

Common Practice

Cmd	Description
54	Read Device Variable Information[24]
78	Read Aggregated Commands
79	Write Device Variable
103	Write Burst Period
104	Write Burst Trigger
105	Read Burst Mode Configuration
106	Flush Delayed Response Buffers
107	Write Burst Device Variables

Cmd	Description
108	Write Burst Mode Command Number[25]
109	Burst Mode Control
115	Read Event Notification Summary
116	Write Event Notification Bit Mask
117	Write Event Notification Timing
118	Event Notification Control
119	Acknowledge Event Notification

While not required, a Wireless Field Device may provide an analog interface to measure or control the loop current. When this is the case Commands 35, 40, 45, 46 must be supported to allow access to and control of the loop current (see Command Summary Specification).

Table 10. Recommended Common Practice Commands for Wireless Discrete and Hybrid Field Devices

Common Practice

Cmd	Description
98	Read Command Action
99	Configure Command Action

Cmd	Description

In addition to Universal and Common Practice Commands, the Field Device must support the standardized commands and procedures defined for WirelessHART devices.

[24] Command 54 and 79 are required for Wireless Hybrid Field Devices. They are not required for Wireless Discrete Field Devices.

[25] The Field Device must support Command 64,395 and publishing of Command 64,386

8. DISCRETE ADAPTER

Discrete Adapters enable integration of small PLCs into a HART network (especially WirelessHART networks). All Discrete Adapters (via HART Communications) must: cache and publish (as Discrete Variables) specified PLC registers; provide access to PLC register memory; and support Burst-Mode. Discrete Adapters may support reading and writing of PLC programs using Block Data Transfer. Aside from these specializations, a Discrete Adapter is a Discrete Device and must adhere to all Discrete Device Requirements.

Discrete Adapters must support HART communications and must have a communication interface to the PLC[26] (e.g., Modbus, DeviceNet, Ethernet-IP, PPI, DF1, PROFIBUS-DP, etc). Using its PLC interface, the Discrete Adapter must read/write specified holding registers or blocks of registers, and must provide host application access to their values via HART Discrete Variables.

Discrete Adapters may also provide a block transfer mechanism used to pass PLC programming information to the PLC via HART.

Because the PLCs offer a wide variety of open and/or proprietary communication protocols this specification focuses on enabling host application access via HART. The Field Device Specification for the Discrete Adapter must describe the physical connection to the PLC, the PLC(s) supported and communication protocols supported to communicate with the PLC.

8.1. General Requirements

Typically, small PLCs contain program memory, I/O and registers (i.e., user memory). Registers are organized into 16-bit words and PLC programs normally set aside a block of registers to exchange information with external systems and User Interfaces. Small PLCs generally have 256-1024 16-bit registers.

PLC I/O can consist of discrete inputs and outputs and Analog I/O. Discrete Adapters must be able to transfer each of these data types; results from PLC program execution; and application settings between the PLC and the host application. This is accomplished by mapping PLC registers to Discrete Variables In other words, the Discrete Adapter must support Command 64,397, *Map PLC Registers to Discrete Variables*. PLC registers are mapped as a block into Discrete Variables[27]. Discrete adapters must be able to map at least three (3) blocks of PLC registers to Discrete Variables.

In addition, wired Discrete Adapters must be able to map a PLC register value to the loop current using Command 64,393, *Map Discrete Variable to Dynamic Variable*.

[26] The PLC could also be embedded in the Discrete Adapter (e.g., through the use of single chip PLC technology).

[27] The Byte ordering may be different between the Discrete Adapter and the PLC. Irrespective of the PLC byte ordering the MSByte of the PLC register must be transferred to/from the MSByte of the Discrete Variable.

8.2. Discrete Adapter Support for Common Practice Commands

In addition to Universal Commands, Discrete Adapters must support the Common Practice Commands specified in Table 11.

Table 11. Required Common Practice Commands for All Discrete Adapters

Common Practice

Cmd	Description
41	Perform Self Test
42	Perform Device Reset
59	Write Number Of Response Preambles

Cmd	Description
90	Read Real-Time Clock
103	Write Burst Period

Wired Discrete Adapters must provide an analog interface to measure or control the loop current and support Common Practice Commands 35, 40, 45, and 46. Wired Discrete Adapter should implement Command 34. Furthermore, all Discrete Adapters should implement the Common Practice Commands listed in Table 12.

Wireless Discrete Adapters must be able to burst Discrete Variables (i.e., burst mode is mandatory for wireless Discrete Adapters). Burst may be configured to be burst periodically or whenever a corresponding register value changes.

Table 12. Recommended Common Practice Commands for Discrete Adapters

Common Practice

Cmd	Description
71	Lock Device
72	Squawk
73	Find Device
76	Read Lock Device State
78	Read Aggregated Commands
95	Read Device Communications Statistics
96	Read Synchronous Action
97	Configure Synchronous Action
98	Read Command Action

Cmd	Description
99	Configure Command Action
111	Transfer Service Control
112	Transfer Service
103	Write Burst Period
104	Write Burst Trigger
105	Read Burst Mode Configuration
106	Flush Delayed Response Buffers
108	Write Burst Mode Command Number[28]
109	Burst Mode Control

In addition to Universal and Common Practice Commands, the Field Device must support the standardized commands and procedures defined for Discrete Devices.

[28] If Burst Mode is supported then the Field Device must support Command 64,395 and publishing of Command 64,386

9. COMMANDS

9.1. General Discrete Field Device Commands

9.1.1 Command 64,384 Read Discrete Device Capabilities

This command reads the capabilities of the discrete device. Host systems can read this information in order to know what other information is available from this device.

Request Data Bytes

Byte	Format	Description
None		

Response Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Number of Discrete Variables contained in this device
2	Enum-8	DLEU support in this device (see Subsection 10.7 Table 7. DLEU Support)
3	Unsigned-8	(Adapter) Number of PLC register blocks that can be mapped (must be set to 0 if Not Discrete Adapter)

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1-127		Undefined

9.1.2 Command 64,385 Read Discrete Variable Properties

Provides information about a Discrete Variable, including its type, process connection, fault behavior as well as the classification of the Discrete Variable.

The Discrete Variable Type declares whether the Discrete Variable is, for example, a packed set of bits or a State Discrete Variable (i.e, a Discrete Function Block).

The Discrete Variable Connection indicates the Discrete Variable is general purpose or has a direct or indirect connection to the plant

The Discrete Variable Classification contains information about what the numeric values of the Discrete Variable mean (e.g. Start, Stop, Run, On, Off, etc).

The Fault State defines what the device will do for Discrete Variables, when the field device otherwise doesn't know what the value should be. One example might be when a discrete output loses communication with the controlling host. Refer to the appropriate field device specification for conditions in which the field device drives values to fault states.

Request Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Discrete Variable Index

Response Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Discrete Variable Index
2.7-2.6	Enum-2	Discrete Variable Type (see Subsection 10.1 Table 1. Discrete Variable Types)
2.5-2.2	Bits-4	Reserved. Must be set to zero.
2.1-2.0	Enum-2	Discrete Variable Connection. Indicates type of connection to the Plant (see Subsection 10.2 Table 2. Discrete Variable Connection Codes)
3	Enum	Discrete Variable Classification (see Subsection 10.6 Table 6. Discrete Variable Classification Codes)
4	Unsigned-8	Device Variable Index[29] (See Common Table 34).
5	Enum	Fault State Mode[30] (see Subsection 10.3 Table 3. Fault State Modes)
6-7	Unsigned-16	Fault State Value[31]
8-11	Time	Fault Timeout. Shed time before asserting the Fault State Value[32].
12-13	Unsigned-16	Index to Discrete Variable returning actual value (must return 0x0FFFF if Actual value cannot be read back).

[29] If Discrete Variable is not mapped to a Device Variable then the device must return "251", None, for the Device Variable Index.

[30] The Fault State Mode must return "251", None, when Fault Behavior is not supported for this Discrete Variable.

[31] The Fault State Value must return 0xFFFF when Fault Behavior is not supported for this Discrete Variable.

[32] The Fault Timeout must return 0x00000000 when Fault Behavior is not supported for this Discrete Variable.

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7-15		Undefined
16	Error	Access Restricted
17-127		Undefined

9.1.3 Command 64,386 Read Discrete Variables

This command reads a block of Discrete Variables (values and status). If necessary, the "Number of of Discrete Variables to read" must be coerced to a valid quantity and the "Set to nearest value" Response Code returned.

The device response data must return the values and status[33] actually/currently used in the device whether the Discrete Variable is online (live), in Simulation (see Command 64,388), in Local Override (see Command 64,389) or in Fault Mode (see Subsection 6.2.2).

If the device supports burst mode then this command must be capable of being burst by the device. If this command is being burst then Command 64,395 is used to define which Discrete Variables must be burst.

Request Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Index of First Discrete Variable to read
2	Unsigned-8	Number of Discrete Variables to read (n)

Response Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Index of first Discrete Variable Returned
2	Unsigned-8	Number of Discrete Variables returned (n)
3	Bits-8	Extended Device Status
4-7	Time	Time stamp for most recent change to Actual Value of first Discrete Variable Value.
8-9	Unsigned-16	First Discrete Variable Value
10	Bits-8	First Discrete Variable Status (See *Command Summary Specification* and Subsection 10.4 Table 4. Discrete Variable Specific Status)
11-12	Unsigned-16	Second Discrete Variable Value
13	Bits-8	Second Discrete Variable Status
	• • •	• • •
3n+8 - 3n+9	Unsigned-16	Last Discrete Variable Value
3n+10	Bits-8	Last Discrete Variable Status

[33] The Discrete Variable Specific Status bits must be set/reset in the status of each individual Discrete Variable according to the Discrete Variable's current mode. e.g., "Discrete Variable in Simulation or Local Override" and "Discrete Variable in Fault Mode" must be set/reset accordingly.

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7		Undefined
7		Undefined
8	Warning	Set to nearest value
9-15		Undefined
16	Error	Access Restricted
17-127		Undefined

9.1.4 Command 64,387 Write Discrete Variables

This command writes a block of Discrete Variables. If necessary, the "Number of of Discrete Variables to write" must be coerced to a valid quantity and the "Set to nearest value" Response Code returned.

Writes to inputs must be ignored and must not affect device operation (i.e., the input Discrete Variable value and status must not be changed by this command).

This command is used by Host Applications to implement the control strategy designed for the corresponding process/manufacturing unit. Status is included in this write command to allow its propagation through the control chain. Valid command requests must be accepted without error even when write protect is asserted. Configuration Changed counter and bit are unaffected by reception of this command.

The response data must return the values and status actually stored in the device[34]. When available, the actual output Discrete Variable values can be read from the Discrete Variable indicated in Command 64,385 Read Discrete Variable Properties.

Request Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Index of first Discrete Variable to Write
2	Unsigned-8	Number of Discrete Variables to write (n)
3-4	Unsigned-16	First Discrete Variable Value
5	Bits-8	First Discrete Variable Status (See *Command Summary Specification* and Subsection 10.4 Table 4. Discrete Variable Specific Status)
6-7	Unsigned-16	Second Discrete Variable Value
8	Bits-8	Second Discrete Variable Status
	• • •	• • •
3n+3 - 3n+4	Unsigned-16	Last Discrete Variable Value
3n+5	Bits-8	Last Discrete Variable Status

Response Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Index of first Discrete Variable
2	Unsigned-8	Number of Discrete Variables Written (n)
3-4	Unsigned-16	First Discrete Variable Actual Value
5	Bits-8	First Discrete Variable Status
6-7	Unsigned-16	Second Discrete Variable Actual Value
8	Bits-8	Second Discrete Variable Status
	• • •	• • •
3n+3 - 3n+4	Unsigned-16	Last Discrete Variable Actual Value
3n+5	Bits-8	Last Discrete Variable Status

[34] Response Code 14 must be returned if one of the Discrete Variables are being simulated or overridden

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7		Undefined
8	Warning	Set to nearest value
9-13		Undefined
14	Warning	Discrete Variable being simulated or overridden
15		Undefined
16	Error	Access Restricted
17-127		Undefined

9.1.5 Command 64,388 Simulate Discrete Variables

Simulation is used to modify the values accessed by the Host Application. Physical Output connections are unaffected by this command. This command is used to test the configuration and behavior of connected Host Applications.

The Discrete Variable value and status must be modified in Device as specified in this command. Discrete Variable values written by this command must be used within the Field Device. For example, Discrete Variables used

- Within a DLEU will be used within the DLEU calculations during the next scheduled computation cycle of the DLEU;
- As Actual State Values, must trigger appropriate transitions within associated Discrete Function Block; and
- As Physical Inputs to a State Discrete Variable must result in the corresponding changes in the Actual State Discrete Variable.

For Discrete Variables automatically calculated by the field device, normal operation is resumed when power is removed from the device, or upon performing a device reset.

Request Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Discrete Variable Index
2	Enum-8	Discrete Variable Simulation Mode (See Common Table 19)
3-4	Unsigned-16	Discrete Variable Simulated Value
5	Bits-8	Discrete Variable Simulated Status (See *Command Summary Specification* and Subsection 10.4 Table 4. Discrete Variable Specific Status)

Response Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Index of first Discrete Variable
2	Enum-8	Discrete Variable Simulation Mode
3-4	Unsigned-16	Discrete Variable Simulated Value
5	Bits-8	Discrete Variable Simulated Status

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7	Error	In Write Protect Mode
8	Warning	Discrete Variable Status bits not set.
9	Error	Undefined
10		Invalid Simulation Mode
11	Error	Invalid Simulated Value
12-15		Undefined
16	Error	Access Restricted
17	Error	Invalid Discrete Variable Index. The Discrete Variable does not exist in this Field Device.
18		Undefined
19	Error	Device Variable index not allowed for this command.
20-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.1.6 Command 64,389 Local Override

Local Override affects both the values accessed by the Host Application and the physical outputs driven by this command. In effect, this command allows outputs to be controlled manually without impacting the automation configured into the Discrete Device.

This command allows the state of Discrete Variables to be set to a specified value, and held at that value instead of taking on new (automatic) control values. The Discrete Variable status shall be set to Manual/Fixed when Local Override is in effect.

The Discrete Variable value and status must be modified in the Device as specified in this command. Consumers of this Discrete Variable must use the values written by this command within the Field Device. For example, Discrete Variables used

- Within a DLEU will be used within the DLEU calculations during the next scheduled computation cycle of the DLEU;
- As Actual State Values, must trigger appropriate transitions within associated Discrete Function Block; and
- As Physical Inputs to a State Discrete Variable must result in the corresponding changes in the Actual State Discrete Variable.

For Discrete Variables automatically calculated by the field device, normal operation is resumed when power is removed from the device, or upon performing a device reset.

Request Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Discrete Variable Index
2	Unsigned-8	Discrete Variable Override Mode (See Common Table 19)
3-4	Unsigned-16	Discrete Variable Override value

Response Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Discrete Variable Index
2	Unsigned-8	Override Mode (0= Not overridden, 1 = Override).
3-4	Unsigned-16	Override value

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7	Error	In Write Protect Mode
8-9		Undefined
10	Error	Invalid Override Mode
11	Error	Invalid Override Value
12-15		Undefined
16	Error	Access Restricted
17	Error	Invalid Discrete Variable Index. The Discrete Variable does not exist in this Field Device.
18		Undefined
19	Error	Device Variable index not allowed for this command.
20-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.1.7 Command 64,390 Write Discrete Variable Classification

This command writes the classification of the specified Discrete Variable.

Request Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Discrete Variable Index
2	Enum	Discrete Variable Classification (see Subsection 10.6 Table 6. Discrete Variable Classification Codes)

Response Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Discrete Variable Index
2	Enum	Discrete Variable Classification

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7	Error	In Write Protect Mode
8-9		Undefined
10	Error	Invalid Discrete Variable Classification Code
11	Error	Invalid Discrete Variable Index
12-15		Undefined
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.1.8 Command 64,391 Write Discrete Variable Type and Connection Code

This command writes the Type and Connection Code for the specified Discrete Variable.

Request Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Discrete Variable Index
2.7-2.6	Enum-2	Discrete Variable Type (see Subsection 10.1 Table 1. Discrete Variable Types)
2.5-2.2	Bits-4	Reserved. Must be set to zero.
2.1-2.0	Enum-2	Discrete Variable Connection. Indicates type of connection to the Plant (see Subsection 10.2 Table 2. Discrete Variable Connection Codes)

Response Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Discrete Variable Index
2.7-2.6	Enum-2	Discrete Variable Type
2.5-2.2	Bits-4	Reserved. Must be set to zero.
2.1-2.0	Enum-2	Discrete Variable Connection.

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection (e.g., Discrete Variable Type)
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7	Error	In Write Protect Mode
8-9		Undefined
10	Error	Invalid Discrete Variable Connection Code
11	Error	Invalid Discrete Variable Code
12-15		Undefined
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.1.9 Command 64,392 Write Discrete Variable Fault Operation

This command configures fault behavior for the specified Discrete Variable.

Request Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Discrete Variable Index
2	Enum	Fault State Mode (see Subsection 10.3 Table 3. Fault State Modes)
3-4	Unsigned-16	Fault State Value
5-8	Time	Fault Timeout – Shed time before asserting the Fault State Value. When set to 0, the Fault State Value will be immediately asserted upon fault detection.

Response Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Discrete Variable Index
2	Enum	Fault State Mode
3-4	Unsigned-16	Fault State Value
5-8	Time	Fault Timeout

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1-2		Undefined
3	Error	Passed Parameter Too Large (Fault Timeout)
4	Error	Passed Parameter Too Small
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7	Error	In Write Protect Mode
8	Warning	Set to nearest value
9-10		Undefined
11	Error	Invalid Discrete Variable Index
12	Error	Invalid Fault State Mode
13	Error	Invalid Fault State Value
14-15		Undefined
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.1.10 Command 64,393 Map Discrete Variable to Dynamic Variable

This command maps a Discrete Variable to a Dynamic Variable.

Request Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Discrete Variable Index
2	Unsigned-8	Analog Channel Index (See Common Table 15)[35]

Response Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Discrete Variable Index
2	Unsigned-8	Analog Channel Index

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7	Error	In Write Protect Mode
8-10		Undefined
11	Error	Invalid Discrete Variable Code
12-15		Undefined
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

[35] PV is Analog Channel [0], SV is Analog Channel [1], etc. See *Command Summary Specification* and *Universal Command Specification*.

9.1.11 Command 64,394 Read Burst Discrete Variables

This command reads which Discrete Variables are returned with Command 64,386 when it is being burst by the device.

Request Data Bytes

Byte	Format	Description
0	Unsigned-8	Burst Message

Response Data Bytes

Byte	Format	Description
0	Unsigned-8	Burst Message
1-2	Unsigned-16	Index of first Discrete Variable to burst
3	Unsigned-8	Number of Discrete Variables to burst

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7-15		Undefined
16	Error	Access Restricted
17-127		Undefined

9.1.12 Command 64,395 Write Burst Discrete Variables

This is a burst mode command

Selects the Discrete Variables that will be used by a bursting device to be returned by burst Command 64,386.

Request Data Bytes

Byte	Format	Description
0	Unsigned-8	Burst Message
1-2	Unsigned-16	Index of first Discrete Variable to burst
3	Unsigned-8	Number of Discrete Variables to burst

Response Data Bytes

Byte	Format	Description
0	Unsigned-8	Burst Message
1-2	Unsigned-16	Index of first Discrete Variable to burst
3	Unsigned-8	Number of Discrete Variables to burst

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection (Burst Message)
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7	Error	In Write Protect Mode
8	Warning	Set to Nearest (Number of Discrete Variables to burst)
9	Error	Invalid Discrete Variable Index
10-15		Undefined
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.1.13 Command 64,396 Read PLC Register Mapping

This command is used by Discrete Adapters to disclose the block(s) of registers in the connected PLC that are mapped to Discrete Variables.

Request Data Bytes

Byte	Format	Description
0	Unsigned-8	PLC Block ID

Response Data Bytes

Byte	Format	Description
0	Unsigned-8	PLC Block ID
1-2	Unsigned-16	Index of first PLC Register to map
3-4	Unsigned-16	Number of PLC Registers to Map
5-6	Unsigned-16	Index of First Destination Discrete Variable

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7-8		Undefined
9	Error	Invalid PLC Block ID
12-15		Undefined
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.1.14 Command 64,397 Map PLC Registers to Discrete Variables

This command is used by Discrete Adapters to map block(s) of registers in the connected PLC to Discrete Variables.

Request Data Bytes

Byte	Format	Description
0	Unsigned-8	PLC Block ID
1-2	Unsigned-16	Index of first PLC Register to map
3-4	Unsigned-16	Number of PLC Registers to Map
5-6	Unsigned-16	Index of First Destination Discrete Variable

Response Data Bytes

Byte	Format	Description
0	Unsigned-8	PLC Block ID
1-2	Unsigned-16	Index of first PLC Register to map
3-4	Unsigned-16	Number of PLC Registers to Map
5-6	Unsigned-16	Index of First Destination Discrete Variable

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7	Error	In Write Protect Mode
8	Warning	Number of PLC Registers set to nearest possible value
9	Error	Invalid PLC Block ID
10	Error	Invalid PLC Register number
11	Error	Invalid Discrete Variable ID.
12-15		Undefined
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.2. DLEU Commands

The Discrete Logic Execution Unit is configured using standard HART commands. These commands are described below. All devices supporting a DLEU must support all commands in this section.

9.2.1 Command 64,448 Read DLEU Information

This command reads the capabilities of the DLEU. Host systems can read this information in order to know what other information is available from this device.

Request Data Bytes

Byte	Format	Description
None		

Response Data Bytes

Byte	Format	Description
0	Unsigned-8	Mode (see Subsection 10.8 Table 8. DLEU Mode)
	Time	Scan Rate of DLEU. Worst-case time to evaluate a function register.
	Unsigned-16	Number of Function Registers contained in this device
	Unsigned-16	Number of Counters supported by this device.
	Unsigned-16	Number of Timers supported by this device.
	Time	Timer resolution. Must be set to 0xFFFF FFFF if Timers not supported
	Unsigned-16	Number of Drum/MUXs supported by this device.
	Unsigned-16	Number of Drum Registers per Drum/MUX supported by this device (must be set to 0 if Drum/MUX functions are not supported).

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1-127		Undefined

9.2.2 Command 64,449 Write DLEU Mode

This command enables or disables the DLEU execution. While execution of the DLEU is disabled, all output values for configured FRs remain at last value, and the status is set to Fixed – Constant.

Request Data Bytes

Byte	Format	Description
0	Unsigned-8	Mode (see Subsection 10.8 Table 8. DLEU Mode)

Response Data Bytes

Byte	Format	Description
0	Unsigned-8	Mode

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7	Error	In Write protect
8-15		Undefined
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.2.3 Command 64,450 Reset DLEU

This command performs a 'hard' reset on the DLEU. The DLEU mode is set to "Disabled" and all Function Registers are set to the END function (i.e., 0xFFFF). All programmed DLEU functions are discarded.

Request Data Bytes

Byte	Format	Description
None		

Response Data Bytes

Byte	Format	Description
None		

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1-4		Undefined
6	Error	Device-Specific Command Error
7	Error	In Write protect
8-15		Undefined
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.2.4 Command 64,451 Read FR Configuration and Status

This command reads the configuration of the FR specified by Index. FRs must be numbered consecutively starting from zero (0).

Request Data Bytes

Byte	Format	Description
0-1	Unsigned-16	Index of FR to read

Response Data Bytes

Byte	Format	Description
	Unsigned-16	Index of FR read
	Unsigned-8	Function (See Subsection 10.9 Table 9. DLEU Function Codes)
	Unsigned-8	Fault Behavior (See Subsection 10.12 Table 12. Function Register Fault)
	Unsigned-8	FR Status (See Subsection 10.11 Table 11. Function Register Status)
	Unsigned-16	OUT
	Unsigned-16	IN1
	Unsigned-16	IN2
	Unsigned-16	IN3

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7-15		Undefined
16	Error	Access Restricted
17-127		Undefined

9.2.5 Command 64,452 Write FR Configuration

This command configures the FR inputs, output, function type, and fault behavior.

Request Data Bytes

Byte	Format	Description
	Unsigned-16	Index of FR to write
	Unsigned-8	Function Code (See Subsection 10.9 Table 9. DLEU Function Codes)
	Unsigned-8	Fault Behavior (See Subsection 10.12 Table 12. Function Register Fault)
	Unsigned-16	OUT. Index of Discrete Variable to write output value to. Must be set to 0xFFFF if value not to be written to a Discrete Variable.
	Unsigned-16	IN1.
	Unsigned-16	IN2.
	Unsigned-16	IN3.

Response Data Bytes

Byte	Format	Description
	Unsigned-16	Index of FR written
	Unsigned-8	Function Code
	Unsigned-8	Fault Behavior
	Unsigned-16	OUT.
	Unsigned-16	IN1.
	Unsigned-16	IN2.
	Unsigned-16	IN3.

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection (FR Index)
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7	Error	In Write Protect
8		Undefined
9	Error	Invalid IN1 Value
10	Error	Invalid IN2 Value
11	Error	Invalid Function
12	Error	Invalid Output
13	Error	Invalid Fault Behavior
14		Undefined
15	Error	Invalid IN3 Value
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.2.6 Command 64,453 Drum Control

This command allows a Drum/MUX register to be disabled, and reset. If the FR tied to this register is configured as a MUX, disabling and resetting the Drum/MUX register has no effect on the FR computation. Because this command is expected to occur as part of normal operating conditions, it shall not be prevented by write protect, nor shall it set the configuration change bit or counter.

Request Data Bytes

Byte	Format	Description
0	Unsigned-16	Index of DRUM Register to control
1	Unsigned-8	Mode (see Subsection 10.10 Table 10. Drum Control Mode)

Response Data Bytes

Byte	Format	Description
0	Unsigned-16	Index of DRUM Register to control
1	Unsigned-8	Mode

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection (Drum Register index)
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7		Undefined
8	Warning	Response truncated (Number of entries to write is too high)
9-15		Undefined
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.2.7 Command 64,454 Read Drum Registers

This command reads the values to be used by a FR configured for the Drum/MUX function type. Devices supporting this command must support a minimum of 16 Drum/MUX values for each Drum/MUX register supported. If the number of drum/MUX values to be read causes the last index to exceed the number supported by the device, the request will be truncated to the last index supported, and "Response Truncated" Response Code will be returned.

Drum registers and their array of values must be numbered consecutively starting from zero (0).

Request Data Bytes

Byte	Format	Description
	Unsigned-16	Index (d) of Drum/MUX Register to read array from
	Unsigned-16	Index (a) of first array value to read
	Unsigned-8	Number of array values to read

Response Data Bytes

Byte	Format	Description
	Unsigned-16	Index of Drum/MUX Register to read array from
	Unsigned-16	Index of first array value to read
	Unsigned-8	Number of array values to read (*n*)
	Unsigned-16	First Array Value (i.e., Drum Register [d] [a])
	Unsigned-16	Second Array Value (i.e., Drum Register [d] [a+1])
	• • •	• • •
	Unsigned-16	Last Array Value

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection (Drum Register index)
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7		Undefined
8	Warning	Response truncated (Number of entries to read is too high)
9	Error	Invalid array index
10-15		Undefined
16	Error	Access Restricted
17-127		Undefined

9.2.8 Command 64,455 Write Drum Registers

This command configures the values to be used by a FR configured for the Drum/MUX function type. Devices supporting this command must support a minimum of 16 Drum/MUX values for each Drum/MUX register supported. If the number of drum/MUX values to be written causes the last index to exceed the number supported by the device, the request will be truncated to the last index supported, and "Response Truncated" Response Code will be returned.

Request Data Bytes

Byte	Format	Description
	Unsigned-16	Index of DRUM Register to write array for
	Unsigned-16	Index of first array value to write
	Unsigned-8	Number of array values to write
	Unsigned-16	First Array Value
	Unsigned-16	Second Array Value
	• • •	• • •
	Unsigned-16	Last Array Value

Response Data Bytes

Byte	Format	Description
	Unsigned-16	Index of DRUM Register to write array for
	Unsigned-16	Index of first array value to write
	Unsigned-8	Number of array values written (*n*)
	Unsigned-16	First Array Value
	Unsigned-16	Second Array Value
	• • •	• • •
	Unsigned-16	Last Array Value

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection (Drum Register index)
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7	Error	In Write Protect
8	Warning	Response truncated (Number of entries to write is too high)
9	Error	Invalid array index
10-15		Undefined
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.2.9 Command 64,456 Read Counter Values

This command returns the information for the counter register specified by Index. Counter registers must be numbered consecutively starting from zero (0).

Request Data Bytes

Byte	Format	Description
	Unsigned-16	Index of Counter to read

Response Data Bytes

Byte	Format	Description
	Unsigned-16	Index of Counter to read
	Unsgined-16	Initial/Preset Value
	Unsigned-16	Current Value

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7-15		Undefined
16	Error	Access Restricted
17-127		Undefined

9.2.10 Command 64,457 Write Counter Value

This command configures the specified counter register.

Request Data Bytes

Byte	Format	Description
0	Unsigned-16	Index of Counter to read
1-4	Unsigned-16	Initial/Preset Value.

Response Data Bytes

Byte	Format	Description
0	Unsigned-16	Index of Counter to read
1-4	Unsigned-16	Initial/Preset Value.

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7	Error	In Write protect
8 - 15		Undefined
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

9.2.11 Command 64,458 Read Timer Values

This command returns the information for the timer register specified by Index. Timer registers must be numbered consecutively starting from zero (0).

Request Data Bytes

Byte	Format	Description
	Unsigned-16	Index of Timer to read

Response Data Bytes

Byte	Format	Description
	Unsigned-16	Index of Timer to read
	Time	Initial /Preset Value.
	Time	Current Value.

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7-15		Undefined
16	Error	Access Restricted
17-127		Undefined

9.2.12 Command 64,459 Write Timer Value

This command configures the specified timer/counter register. If the exact time value is not achievable in the device, the response will contain the time value actually set in the device, and the "Set to Nearest" Response Code must be returned.

Request Data Bytes

Byte	Format	Description
0	Unsigned-16	Index of Timer to read
1-4	Time	Initial/Preset Value.

Response Data Bytes

Byte	Format	Description
0	Unsigned-16	Index of Timer to read
1-4	Time	Initial /Preset Value.

Command-Specific Response Codes

Code	Class	Description
0	Success	No Command-Specific Errors
1		Undefined
2	Error	Invalid Selection
3-4		Undefined
5	Error	Too Few data Bytes Received
6	Error	Device-Specific Command Error
7	Error	In Write protect
8	Warning	Set to Nearest
9 - 15		Undefined
16	Error	Access Restricted
17-31		Undefined
32	Error	Busy (A DR Could Not Be Started)
33	Error	DR Initiated
34	Error	DR Running
35	Error	DR Dead
36	Error	DR Conflict
37-127		Undefined

10. TABLES

Several tables are used to describe Discrete Devices. These tables are specified below.

10.1. Table 1. Discrete Variable Types

The following Discrete Variable types are defined in this specification:

Code	Description
0	State
1	Arithmetic/Counter (e.g., output from a DLEU counter function
2	Packed (i.e., 16 Boolean Values)
3	Reserved

10.2. Table 2. Discrete Variable Connection Codes

Discrete variables can be either input or output but not both in the same variable. Output variables have the "Target Value" written by a host system, and reports current state via the "Actual Value". Input variables are not writable by the host, and report current state via the "Actual Value".

Code	Description
0	No plant connection (general purpose Discrete Variable).
1	Input from plant
2	Output to plant
3	Reserved

10.3. Table 3. Fault State Modes

Code	Description
0	No Fault State – The Discrete Variable is not altered as a result of a detected fault condition
1	Fail to value – The Discrete Variable will be set to the Fault State Value configured for that Discrete Variable whenever an applicable fault condition is detected by the field device.
2	Hold Last Value – The Discrete Variable will be maintained at the state it was prior to any applicable fault conditions being detected.
251	None. Faults are not detected or acted upon.

10.4. Table 4. Discrete Variable Specific Status

Bit	Description
0x01	Discrete Variable in Simulation (see Command 64,388) or Local Override (see Command 64,389). This bit must be set/reset in the status of each individual Discrete Variable according to the Discrete Variable's current mode.
0x02	Discrete Variable in Fault Mode (see Subsection 6.2.2). This bit must be set/reset in the status of each individual Discrete Variable according to the Discrete Variable's current mode.
0x02-0x04	Reserved. Must be set to zero (0)

10.5. Table 5. Discrete Variable States

This table lists the states allowed by the Protocol. The Discrete Function Block specifies the subset of these states it supports. The Discrete Function Blocks allowed by the Protocol are specified in Subsection 10.6 Table 6. Discrete Variable Classification Codes.

Code	Description	Code	Description
0	Off	32	Dry
1	On	33	Wet
2	Stop	34	Turning Off
3	Run	35	Turning On
4	Close	36	Turn Off
5	Open	37	Turn On
6	Closed	38	Up
7	Fast	39	Down
8	Slow	40	Fill
9	Forward	41	Drain
10	Reverse	42	Heat
11	Top	43	Cool
12	Bottom	44	Present
13	Full	45	Absent
14	Empty	46	Opened
15	Hot	47	Start
16	Cold	48	Fail
17	Opening	49	Drying
18	Closing	50	Wetting
19	Raising	51	Overload
20	Lowering	52	Partially Open
21	Filling	53	Intermediate
22	Draining	54	High
23	Heating	55	Low
24	Cooling	55-239	Reserved.
25	Stopped	240-249	Enumeration May Be Used For Manufacturer Specific Definitions
26	Running		
27	Stopping	250-65,534	Reserved. Must not be used by any device.
28	Starting		
29	Too Fast	65,535	*null-value* (used to indicate no fault state value)
30	Too Slow		
31	Reversing		

10.6. Table 6. Discrete Variable Classification Codes

Each Discrete Variable has a classification that associates it with a specific Discrete Function Block (e.g., a block for interfacing to a motor starter or a solenoid valve). This table lists the Classifications allowed by the Protocol and Discrete Function Block associated with each.

For each Discrete Function Block a Sub-table is specified that specifies the control sequence (i.e., the finite state machine) used and the states (see Subsection 10.5 Table 5. Discrete Variable States) allowed.

Code[36]	Function Block	Passive State[37]	Description
0	Not Classified		Packed-type Discrete Variables must return "Not Classified". State-type Discrete Variables must NOT return this code.
1	On/Off Switch	Determined by device Manufacturer	General Purpose On/Off Switch. When power is lost the switch could be designed to hold last value or fail off.
2	Limit Switch	N/A	Limit or Proximity Switch. Limit and Proximity switches are unpowered.
3	Motor	Stop	Motor
4	Valve, NO	Open	Normally Open Valve (assume valve resets to default position when it loses power)
5	Valve, NC	Closed	Normally Closed Valve (assume valve resets to default position when it loses power)
6	Mtr Op Valve	Stop	Motor Operated Valve
7	2 Spd Motor	Stop	2 Speed Motor
8	2 Dir Motor	Stop	2 Direction Motor
9	Elevator	Stop	Elevator
10	Tank	Stop	Tank Filling/Emptying Operations
11	Heat/Cool	Off	Heat / Cool
12	Wet/Dry	N/A	This is a switch indicating the presence or absence of a liquid.
13	Presence / Absence	N/A	This is a switch indicating the presence or absence of a medium.
240-249			Enumeration May Be Used For Manufacturer Specific Definitions (see Discrete Function Block definitions in appropriate device-specific document).

This Discrete Function Block is used in-context in each of the subsections below. The Simulation parameters are not shown in each of these descriptions.

The following are the state tables show the control sequence for each Discrete Function Block. Any Discrete Variable that is not classified will specify transition logic in the field device specification for that product. Steady states (indicated by bold font) are required to be supported for each respective discrete classification. Intermediate states (italics) may or may not actually occur depending on the nature of the device. This means that it is possible to skip an intermediate state and go directly to a final steady state.

[36] Any codes not shown are reserved and must not be used by any device.

[37] Passive State is the default state when the device is unpowered, and the initial value on startup.

10.6.1 Table 6.1. Output On/Off Switch State Table

Valid Discrete Variable States (Subset of Table 5)

Code	Description	Code	Description
0	Off	35	Turning On
1	On	48	Fail
34	Turning Off		

State Discrete Variable Specification

There are two types of On/Off Blocks. In the first block control logic is used to drive the On/Off state. In the second case a manual On/Off switch is used – in this second case the Function Block is only used to track the state of the switch.

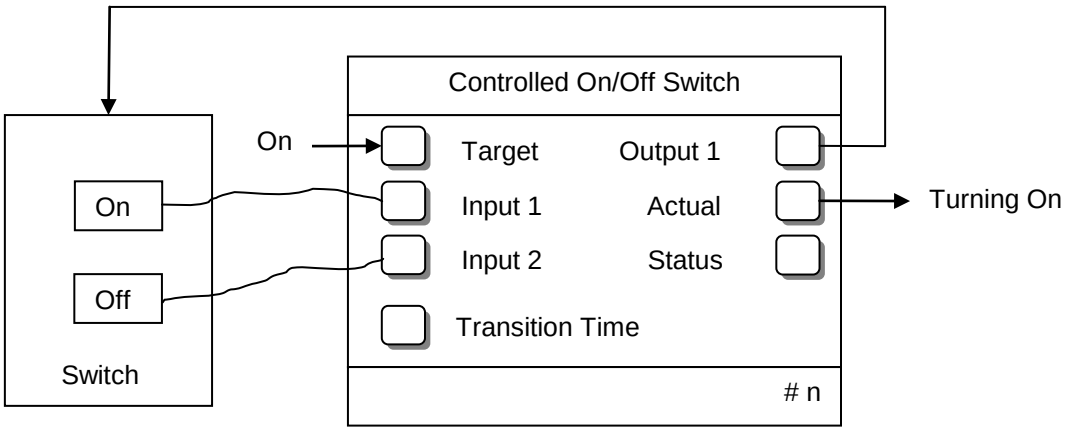


Figure 24. On/Off Switch Controller (Output)

The transition logic along with inputs and outputs are summarized in the table below

Table 13. On/Off Switch Controller State Machine(Output)

Initial State	Target Value		Actual Value	Inputs		Outputs
	On	Off		Input 1 (On)	Input 2 (Off)	Output 1 (On)
On	✓		**On**	1	0	1
		✓	*Turning Off*	0	0	0
		✓	***Fail***	0	0	0
		✓	**Off**	0	1	0
Off		✓	**Off**	0	1	0
	✓		*Turning On*	0	0	1
	✓		***Fail***	0	0	1
	✓		**On**	1	0	1

NOTE: The Intermediate (optional) Fail State indicates that the Transition Time was exceeded. If the Actual Value achieves its Target Value sometime after Transition Time, two things will happen: 1) the Final State will be indicated and 2) the Status will indicate that the Transition Time was exceeded.

In this case a manual On/Off switch is used – the operation of the On/Off Block is illustrated below.

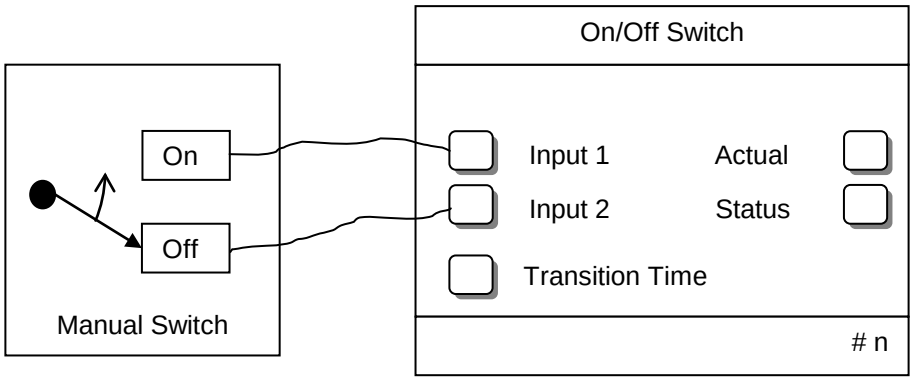


Figure 25. Manual On/Off Switch (Input)

The transition logic along with inputs and outputs are summarized in the table below:

Table 14. Manual On/Off Switch State Table (Input)

Initial State	Actual Value	Inputs	
		Input 1 (On)	Input 2 (Off)
On	**On**	1	0
	Turning Off	0	0
	Fail	0	0
	Off	0	1
Off	**Off**	0	1
	Turning On	0	0
	Fail	0	0
	On	1	0

10.6.2 Table 6.2. Limit Switch State Table

Valid Discrete Variable States (Subset of Table 5)

Code	Description	Code	Description
5	Open	18	Closing
6	Closed	46	Opened
17	Opening		

State Discrete Variable Specification

Limit switches detect the physical position of an object by direct contact with that object. The usage information for a single position limit switch is shown below:

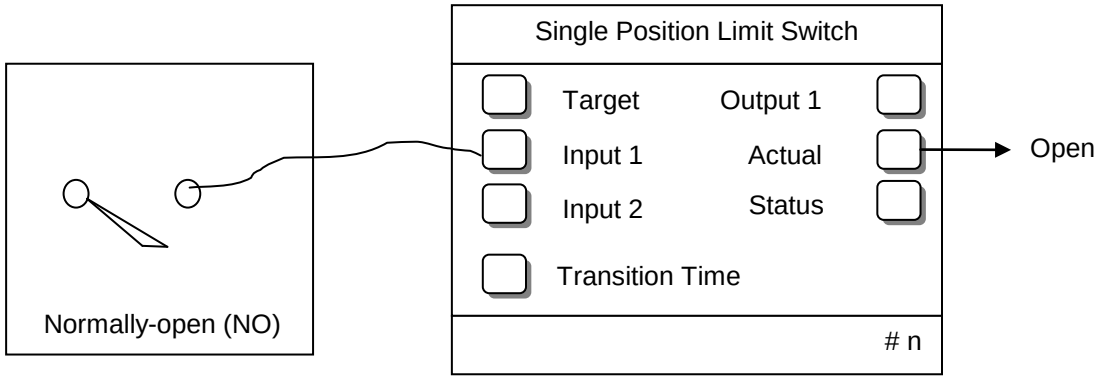


Figure 26. SPST Limit Switch (Input)

The transition logic along with inputs and outputs are summarized in the table below

Table 15. SPST Limit Switch State Table (Input)

Initial State	Actual Value	Input 1 (Closed)
Opened	**Opened**	0
	Closed	1
Closed	**Closed**	1
	Opened	0

In some cases the limit switch may detect both upper and lower limits. The usage information for a single-position double-throw limit switch is shown below.

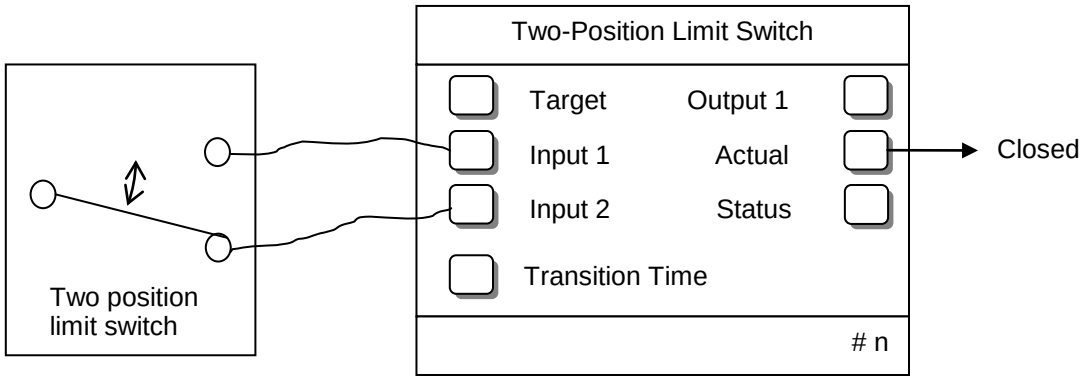


Figure 27. SPDT Limit Switch (Input)

The transition logic along with inputs and outputs are summarized in the table below:

Table 16. SPDT Limit Switch State Table (Input)

Initial State	Actual Value	Inputs	
		Input 1 (Open)	Input 2 (Closed)
Opened	**Opened**	1	0
	Closing	0	0
	Closed	0	1
Closed	**Closed**	0	1
	Opening	0	0
	Opened	1	0

10.6.3 Table 6.3. Motor State Table

Valid Discrete Variable States (Subset of Table 5)

Code	Description	Code	Description
2	Stop	28	Starting
3	Run	47	Start
25	Stopped	48	Fail
26	Running	51	Overload
27	Stopping		

State Discrete Variable Specification

Definition of inputs and outputs:

Start	(Optional) Input. The 'Start' field start push button
Stop	(Optional) Input. The 'Stop' field start push button
Run	Input. Indication that the motor or contactor is actually engaged
Overload	(Optional) Input. Indicates the protection on the motor has engaged (e.g., thermal overload)
Coil	Output. The coil of the contactor or the motor itself
On	(Optional) Output. Indication that the motor is running
Error	(Optional) Output. Indication that the motor has faulted (tripped)

The control circuit for this block is shown below:

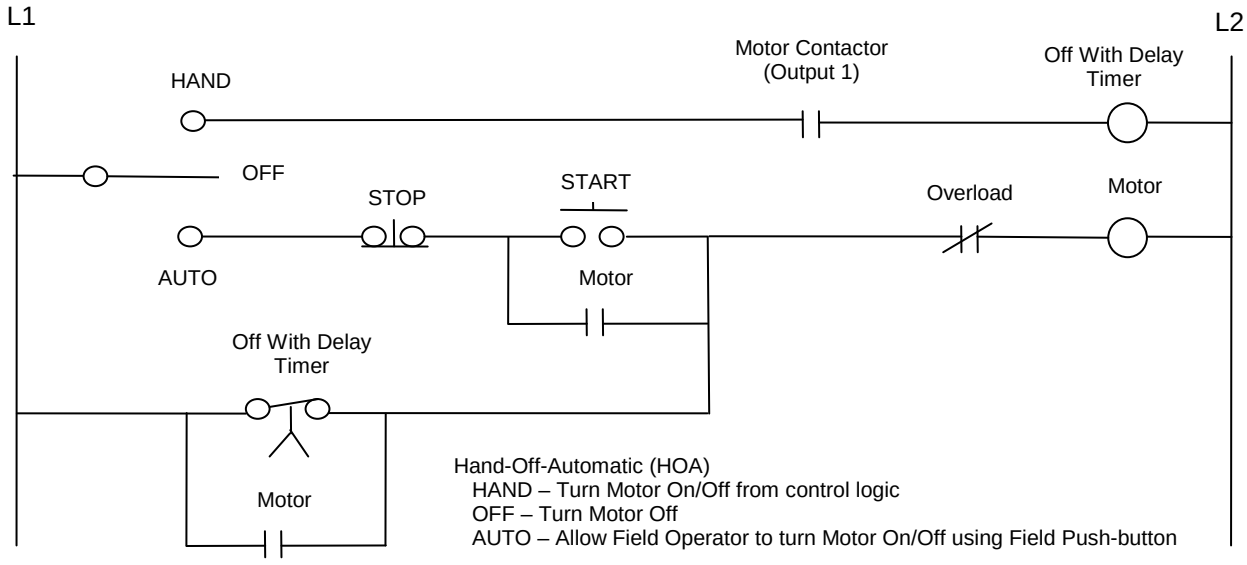


Figure 28. Motor Control Circuit

HAND Logic

1- When the HOA setting is set to HAND and the Motor Contactor is set to 1, the 'Off With Delay Timer' Coil is activated.

2- The 'Off With Delay Timer' deactivates after a timed interval. The 'Motor Run' coil seals the Motor Run logic.

Auto Logic
1- When the START button is pressed the 'Motor Run' coil seals the Motor Run logic.

The block motor control with Hand/Off/Auto (HOA) is illustrated in the diagram below. It is important to have

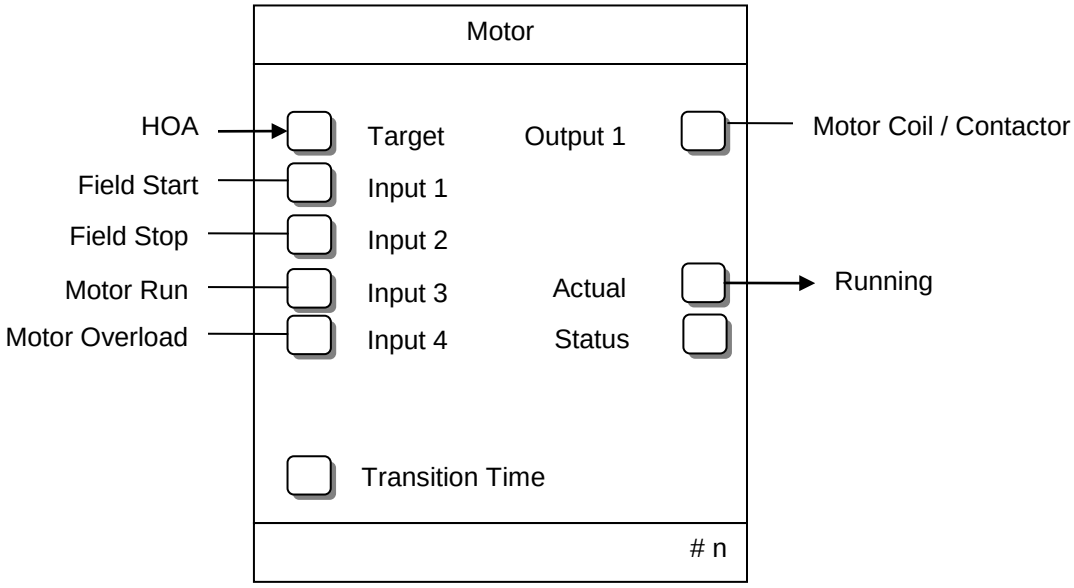


Figure 29. Motor (Output)

The state table is shown below.

Table 17. Motor State Table (Input)

Initial State	Target Value			Actual Value	Inputs				Outputs
	Hand	Off	Auto	Final State	Input 1 (Start)	Input 2 (Stop)	Input 3 (Run)	Input 4 (Overload)	Output 1 (Coil)
Hand	✓			**Stopped**	-	-	0	0	0
	✓			**Running**	-	-	1	0	1
		✓		*Stopping*	-	-	1	0	0
		✓		**Stopped**	-	-	0	0	0
			✓	**Stopped**	0	1	0	0	0
			✓	*Starting*	1	0	1	0	0
			✓	**Running**	1	0	1	0	0
	-	-	-	**Stopped**	-	-	0	1	0
Off		✓		**Stopped**	-	-	0	0	0
	✓			*Starting*	-	-	0	0	1
	✓			**Running**	-	-	1	0	1
			✓	**Stopped**	-	-	0	0	0
			✓	*Starting*	1	0	0	0	0
			✓	**Running**	1	0	1	0	0
			✓	**Stopped**	0	1	0	0	0
	-	-	-	*Fail*	-	-	0	1	0
Auto			✓	**Stopped**	0	1	0	0	0
			✓	**Running**	1	0	1	0	0
	✓			**Stopped**	-	-	0	0	0
	✓			*Starting*	-	-	0	0	1
	✓			**Running**	-	-	1	0	1
		✓		**Stopped**	-	-	0	0	0
	-	-	-	**Stopped**	-	-	0	1	0

10.6.4 Table 6.4. Valve (Normally Open) State Table

Valid Discrete Variable States (Subset of Table 5)

Code	Description	Code	Description
4	Close	18	Closing
5	Open	46	Opened
6	Closed	48	Fail
17	Opening		

State Discrete Variable Specification

Blocking valves are used to block movement in one or more directions. A normally-open valve will have no effect on movement in the Off position and totally block movement in the On. The block compares the requested state (Target) to the actual reported state (Actual) and, after allowing time for the device to change state, generates an error on the status if the Actual value does not match the Target Value. A normally open blocking valve is illustrated in the diagram below.

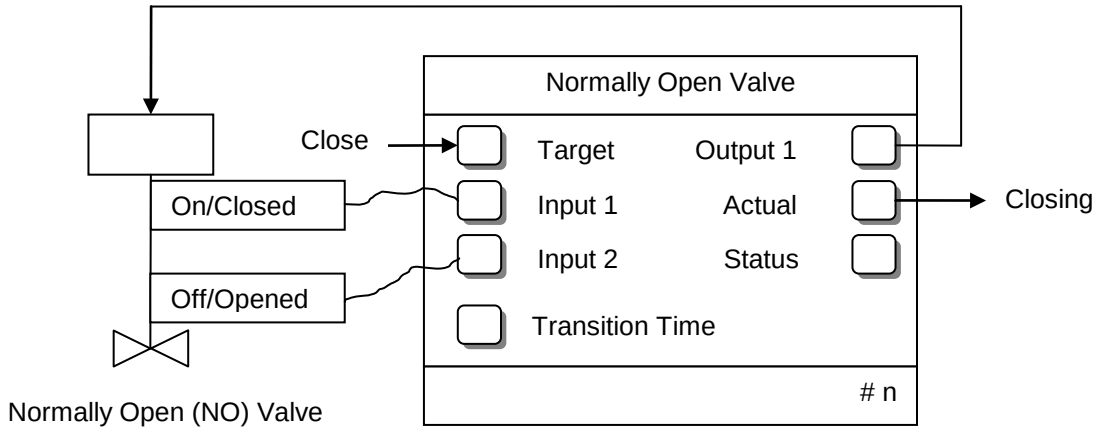


Figure 30. NO Valve (Output)

The transition logic along with inputs and outputs are summarized in the table below:

Table 18. NO Valve State Table (Output)

Initial State	Target Value		Actual Value	Inputs		Outputs
	Open *or* Opened	Close *or* Closed	Final State	Input 1 (Closed)	Input 2 (Open)	Output 1 (On/Close)
Opened	✓		**Opened**	0	1	0
		✓	*Closing*	0	0	1
		✓	*Fail*	0	0	1
		✓	**Closed**	1	0	1
Closed		✓	**Closed**	1	0	1
	✓		*Opening*	0	0	0
	✓		*Fail*	0	0	0
	✓		**Opened**	0	1	0

NOTE: The Intermediate Fail State indicates that the Transition Time was exceeded. If the Actual Value achieves its Target Value sometime after Transition Time, two things will happen: 1) the Final State will be indicated and 2) the Actual Status will indicate that the Transition Time was exceeded.

10.6.5 Table 6.5. Valve (Normally Closed) State Table

Valid Discrete Variable States (Subset of Table 5)

Code	Description	Code	Description
4	Close	18	Closing
5	Open	46	Opened
6	Closed	48	Fail
17	Opening		

State Discrete Variable Specification

Blocking valves are used to block movement in one or more directions. A normally-closed valve will have no effect on movement in the On position and totally block movement in the Off position. The block compares the requested state (Target) to the actual reported state (Actual) and, after allowing time (Transition Time) for the device to change state, generates an error on the status if the Actual value does not match the Target Value. A normally closed blocking valve is illustrated in the diagram below.

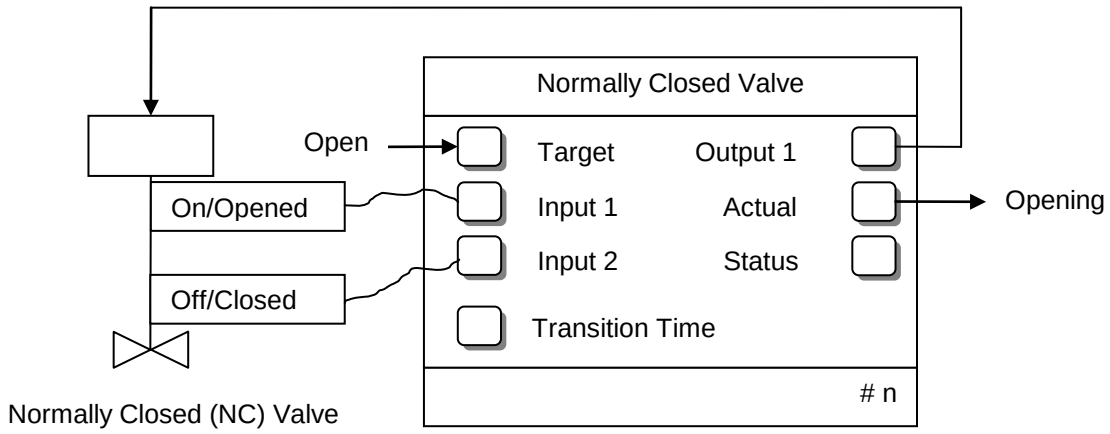


Figure 31. NC Valve (Output)

The transition logic along with inputs and outputs are summarized in the table below:

Table 19. NC Valve State Table (Output)

Initial State	Target Value		Actual Value	Inputs		Outputs
	Open *or* Opened	**Close *or* Closed**		**Input 1 (Open)**	**Input 2 (Closed)**	**Output 1 (On/Open)**
Opened	✓		**Opened**	1	0	1
		✓	*Closing*	0	0	0
		✓	*Fail*	0	0	0
		✓	**Closed**	0	1	0
Closed		✓	**Closed**	0	1	0
	✓		*Opening*	0	0	1
	✓		*Fail*	0	0	1
	✓		**Opened**	1	0	1

NOTE: The Intermediate Fail State indicates that the Transition Time was exceeded. If the Actual Value

achieves its Target Value sometime after Transition Time, two things will happen: 1) the Final State will be indicated and 2) the Actual Status will indicate that the Transition Time was exceeded.

10.6.6 Table 6.6. Motor Operated Valve

Valid Discrete Variable States (Subset of Table 5)

Code	Description	Code	Description
2	Stop	17	Opening
4	Close	18	Closing
5	Open	46	Opened
6	Closed	48	Fail

State Discrete Variable Specification

A motor operated valve opens and closes in response to user command to Open or Close the valve. Once the value reaches it final destination either the Opened or Closed limit switches will be engaged. A user can command the valve to stop by resetting the Open or Close output – in which case the value may be left in an intermediate state somewhere between Opened and Closed. The block is illustrated in the diagram below.

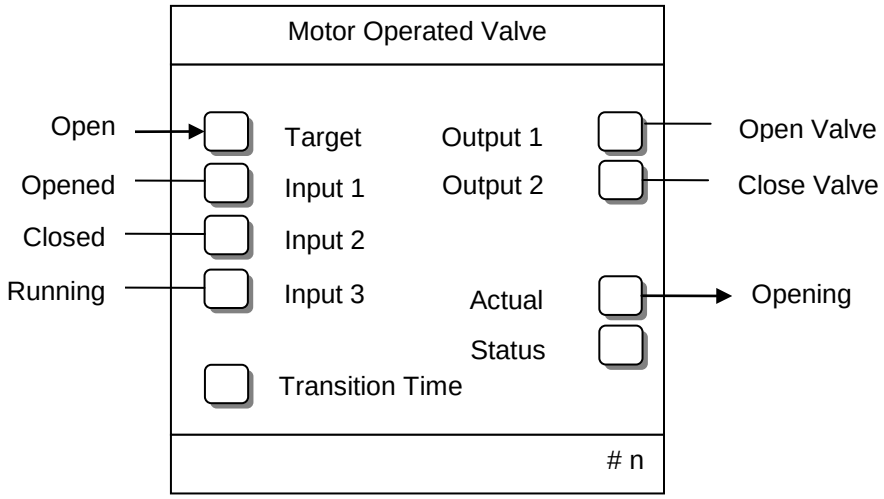


Figure 32. Motor Operated Valve (Output)

The transition logic along with inputs and outputs are summarized in the table below.

Table 20. Motor-Operated Valve State Table (Output)

Initial State	Target Value			Actual Value	Inputs			Outputs	
	Stop	Open	Close		Input 1 (Opened)	Input 2 (Closed)	Input 3 (Running)	Output 1 (Open)	Output 2 (Close)
Stopped	✓			**Stopped**	x	x	0	0	0
		✓		*Opening*	0	0	1	1	0
		✓		*Fail*	0	0	1	1	0
		✓		**Opened**	1	0	0	1	0
			✓	*Closing*	0	0	1	0	1
			✓	*Fail*	0	0	1	0	1
			✓	**Closed**	0	1	0	0	1
Opened	✓			**Opened**	1	0	0	0	0
		✓		**Opened**	1	0	0	1	0
			✓	*Closing*	0	0	1	0	1
			✓	*Fail*	0	0	1	0	1
			✓	**Closed**	0	1	0	0	1
Closed	✓			**Closed**	0	1	0	0	0
		✓		*Opening*	0	0	1	1	0
		✓		*Fail*	0	0	1	1	0
		✓		**Opened**	1	0	0	1	0
			✓	**Closed**	0	1	0	0	1

NOTE 1: x means that the state can be either ‘0’ or ‘1’.
NOTE 2: The Intermediate Fail State indicates that the Transition Time was exceeded. If the Actual Value achieves its Target Value sometime after Transition Time, two things will happen: 1) the Final State will be indicated and 2) the Actual Status will indicate that the Transition Time was exceeded.

10.6.7 Table 6.7. Two-Speed Motor State Table

Valid Discrete Variable States (Subset of Table 5)

Code	Description	Code	Description
2	Stop	27	Stopping
7	Fast	29	Too Fast
8	Slow	30	Too Slow
25	Stopped	48	Fail

State Discrete Variable Specification

The block for two-speed motor control is illustrated in the diagram below.

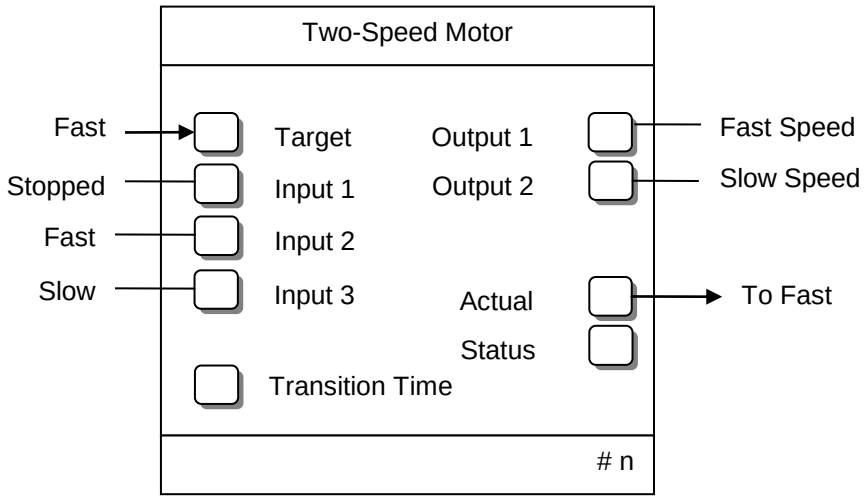


Figure 33. Two-Speed Motor (Output)

The state table is shown below.

Table 21. Two-Speed Motor State Table (Output)

Initial State	Target Value			Actual Value	Inputs			Outputs	
	Stop	Fast	Slow	Final State	Input 1 (Stopped)	Input 2 (Fast)	Input 3 (Slow)	Output 1 (Fast)	Output 2 (Slow)
Stopped	✓			**Stopped**	1	0	0	0	0
		✓		*Too Fast*	0	0	0	1	0
		✓		*Fail*	0	0	0	1	0
		✓		**Fast**	0	1	0	1	0
			✓	*Too Slow*	0	0	0	0	1
			✓	*Fail*	0	0	0	0	1
			✓	**Slow**	0	0	1	0	1
Fast	✓			*Stopping*	0	0	0	0	0
	✓			*Fail*	0	0	0	0	0
	✓			**Stopped**	1	0	0	0	0
		✓		**Fast**	0	1	0	1	0
			✓	*Too Slow*	0	0	0	0	1
			✓	*Fail*	0	0	0	0	1
			✓	**Slow**	0	0	1	0	1
Slow	✓			*Stopping*	0	0	0	0	0
	✓			*Fail*	0	0	0	0	0
	✓			**Stopped**	1	0	0	0	0
		✓		*Too Fast*	0	0	0	1	0
		✓		*Fail*	0	0	0	1	0
		✓		**Fast**	0	1	0	1	0
			✓	**Slow**	0	0	1	0	1

NOTE 1: x means that the state can be either ‘0’ or ‘1’.

NOTE 2: The Intermediate Fail State indicates that the Transition Time was exceeded. If the Actual Value achieves its Target Value sometime after Transition Time, two things will happen: 1) the Final State will be indicated and 2) the Actual Status will indicate that the Transition Time was exceeded.

10.6.8 Table 6.8. 2 Direction Motor State Table

Valid Discrete Variable States (Subset of Table 5)

Code	Description	Code	Description
2	Stop	27	Stopping
9	Forward	28	Starting
10	Reverse	31	Reversing
25	Stopped		

State Discrete Variable Specification

The block for 2-speed direction motor control is illustrated in the diagram below.

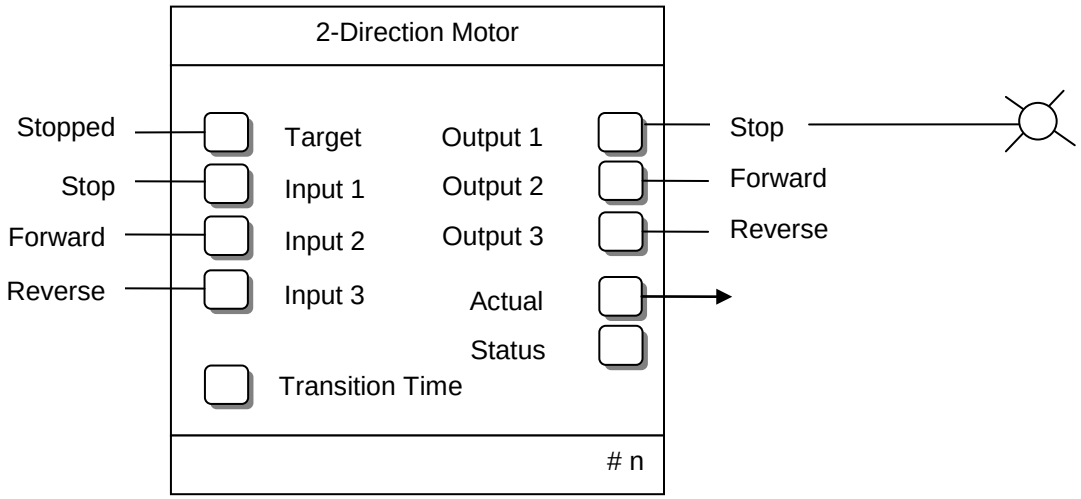


Figure 34. Two-Direction Motor (Output)

The state table is shown below.

Table 22. Two-Direction Motor State Table (Output)

Initial State	Inputs				Actual Value	Outputs		
	Input 1 (Stop)	Input 2 (Forward)	Input 3 (Reverse)	Input 4 (Stopped)	Final State	Output 1 (Stop)	Output 2 (Forward)	Output 3 (Reverse)
Stopped	1	x	x	x	**Stopped**	1	0	0
	0	1	x	x	**Forward**	0	1	0
	0	0	1	x	**Reverse**	0	0	1
Forward	0	x	x	x	**Forward**	0	1	0
	1	0	0	x	*Stopping*	1	0	0
	0	0	1	x	*Stopping-Reverse*	1	0	0
Reverse	0	x	x	x	**Reverse**	0	0	1
	1	x	x	x	*Stopping*	1	0	0
	0	1	0	x	*Stopping-Forward*	1	0	0
Stopping	x	x	x	0	*Stopping*	1	0	0
	x	x	x	1	**Stopped**	1	0	0
Stopping-Reverse	0	x	x	0	*Stopping-Reverse*	1	0	0
	1	x	x	0	*Stopping*	1	0	0
	0	x	x	1	**Reverse**	0	0	1
Stopping-Forward	0	x	x	0	*Stopping-Forward*	1	0	0
	1	x	x	0	*Stopping*	1	0	0
	0	x	x	1	**Forward**	0	1	0

Note – the 2-dir motor is more complex because it needs to stop, reverse its direction, start.

10.6.9 Table 6.9. Elevator State Table
Valid Discrete Variable States (Subset of Table 5)

Code	Description	Code	Description
11	Top	25	Stopped
12	Bottom	27	Stopping
19	Raising	38	Up
20	Lowering	39	Down

State Discrete Variable Specification
The block for elevator control is illustrated in the diagram below.

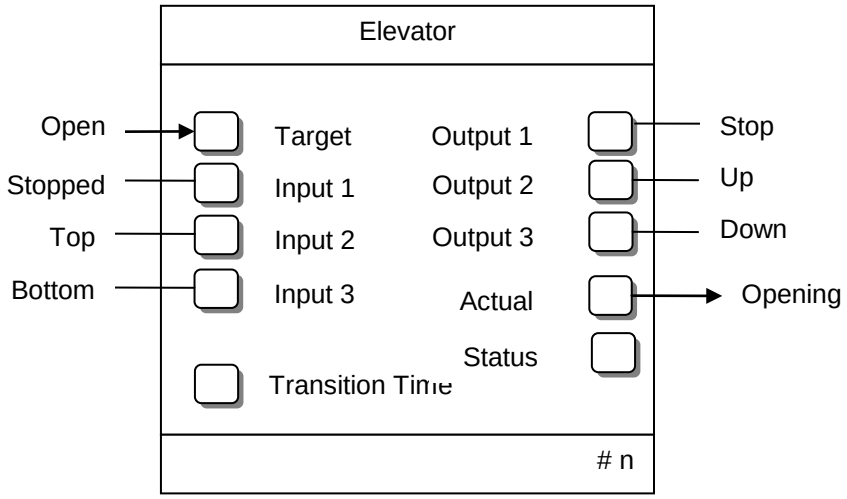


Figure 35. Elevator Control (Output)

The state table is shown below.

Table 23. Elevator Control State Table (Output)

Initial State	Target Value			Actual Value	Inputs			Outputs		
	Stop	Up	Down		Input 1 (Stopped)	Input 2 (Top)	Input 3 (Bottom)	Output 1 (Stop)	Output 2 (Up)	Output 3 (Down)
Stopped	✓			**Stopped**	1	0	0	1	0	0
		✓		*Raising*	0	0	0	0	1	0
		✓		**Top**	0	1	0	1	0	0
			✓	*Lowering*	0	0	0	0	0	1
			✓	**Bottom**	1	0	1	0	0	0
Top	✓			*Stopping*	0	0	0	1	0	0
	✓			**Stopped**	1	0	0	1	0	0
		✓		**Top**	0	1	0	0	0	0
			✓	*Lowering*	0	0	0	0	0	1
			✓	**Bottom**	0	0	1	0	0	0
Bottom	✓			*Stopping*	0	0	0	1	0	0
	✓			**Stopped**	1	0	0	1	0	0
		✓		*Raising*	0	0	0	0	1	0
		✓		**Top**	0	1	0	0	0	0
			✓	**Bottom**	0	0	1	0	0	0

10.6.10 Table 6.10. Tank Level Control State Table

Valid Discrete Variable States (Subset of Table 5)

Code	Description	Code	Description
13	Full	25	Stopped
14	Empty	40	Fill
21	Filling	41	Drain
22	Draining		

State Discrete Variable Specification

The block for tank level control is illustrated in the diagram below.

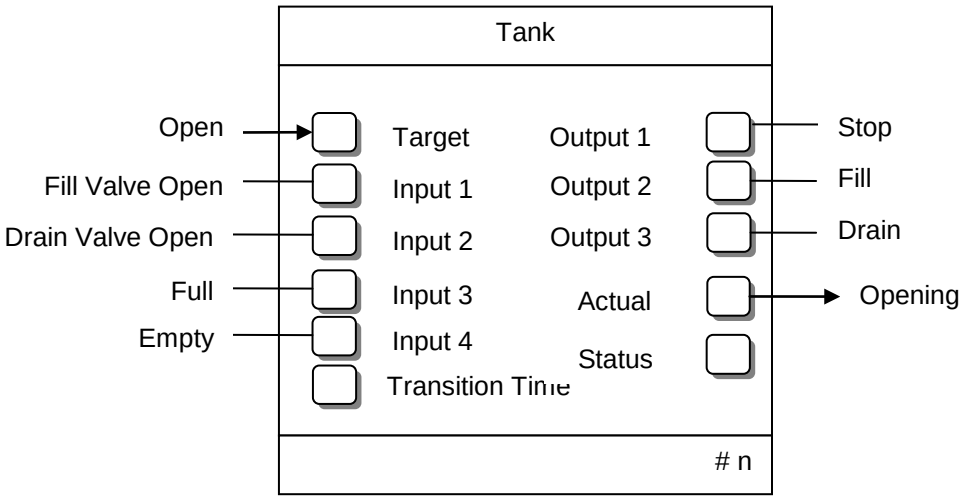


Figure 36. Tank Level Controller (Output)

The state table is shown below.

Table 24. Tank Level Controller State Table (Output)

Initial State	Target Value			Actual Value	Inputs				Outputs		
	Stop	Fill	Drain		Input 1 (Fill)	Input 2 (Drain)	Input 3 (Full)	Input 4 (Empty)	Output 1 (Stop)	Output 2 (Fill)	Output 3 (Drain)
Stopped	✓			**Stopped**	0	0	x	x	1	0	0
		✓		*Filling*	1	0	0	0	0	1	0
		✓		**Full**	0	0	1	0	0	0	0
			✓	*Draining*	0	1	0	0	0	0	1
			✓	**Empty**	0	0	0	1	0	0	0
Full	✓			*Stopping*	0	0	0	0	1	0	0
	✓			**Stopped**	0	0	0	0	1	0	0
		✓		**Full**	0	0	1	0	0	0	0
			✓	*Draining*	0	1	0	0	0	0	1
			✓	**Empty**	0	0	0	1	0	0	0
Empty	✓			*Stopping*	0	0	0	0	1	0	0
	✓			**Stopped**	0	0	0	0	1	0	0
		✓		*Filling*	1	0	0	0	0	1	0
		✓		**Full**	0	0	1	0	0	0	0
			✓	**Empty**	0	0	0	1	0	0	0

NOTE 1: x means that the state can be either ‘0’ or ‘1’.

10.6.11 Table 6.11. Heat/Cool State Table

Valid Discrete Variable States (Subset of Table 5)

Code	Description	Code	Description
2	Stop	25	Stopped
15	Hot	27	Stopping
16	Cold	42	Heat
23	Heating	43	Cool
24	Cooling		

State Discrete Variable Specification

The block for heat/cool control is illustrated in the diagram below.

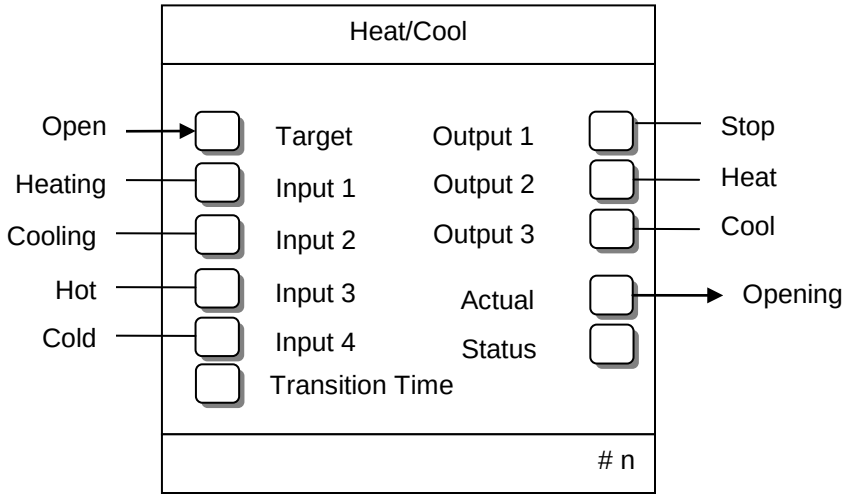


Figure 37. Heat/Cool Control (Output)

The state table is shown below.

Table 25. Heat/Cool Control State Table (Output)

Initial State	Target Value			Actual Value		Inputs				Outputs		
	Stop	Heat	Cool	Intermediate State	Final State	Input 1 (Heating)	Input 2 (Cooling)	Input 3 (Hot)	Input 4 (Cold)	Output 1 (Stop)	Output 2 (Heat)	Output 3 (Cool)
Stopped	✓				**Stopped**	0	0	0	0	1	0	0
		✓		*Heating*		1	0	0	0	0	1	0
		✓			**Hot**	0	0	1	0	0	0	0
			✓	*Cooling*		0	1	0	0	0	0	1
			✓		**Cold**	0	0	0	1	0	0	0
Hot	✓			*Stopping*		0	0	0	0	1	0	0
	✓				**Stopped**	0	0	0	0	1	0	0
		✓			**Hot**	0	0	1	0	0	1	0
			✓	*Cooling*		0	1	0	0	0	0	1
			✓		**Cold**	0	0	0	1	0	0	0
Cold	✓			*Stopping*		0	0	0	0	1	0	0
	✓				**Stopped**	0	0	0	0	1	0	0
		✓		*Heating*		1	0	0	0	0	1	0
		✓			**Hot**	0	0	1	0	0	0	0
			✓		**Cold**	0	0	0	1	0	0	1

10.6.12 Table 6.12. Wet/Dry State Table

Valid Discrete Variable States (Subset of Table 5)

Code	Description	Code	Description
2	Stop	33	Wet
25	Stopped	49	Drying
32	Dry	50	Wetting

State Discrete Variable Specification

The block wet/dry control is illustrated in the diagram below.

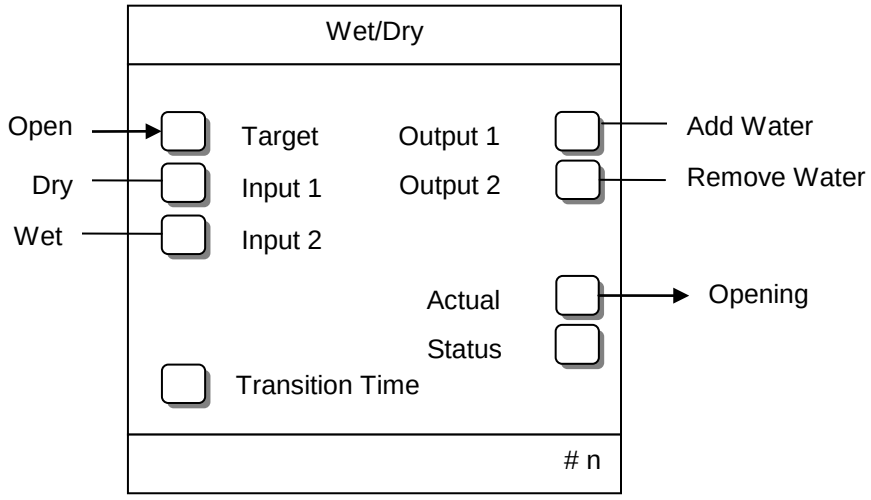


Figure 38. Wet/Dry Control (Output)

The state table is shown below.

Table 26. Wet/Dry Control State Table (Output)

Initial State	Target Value			Actual Value		Inputs		Outputs	
	Stop	Dry	Wet	Intermediate State	Final State	Input 1 (Dry)	Input 2 (Wet)	Output 1 (Add Water)	Output 2 (Remove Water)
Wet	✓				**Stopped**	0	1	0	0
		✓		*Drying*		0	0	0	1
		✓			**Dry**	1	0	0	0
			✓		**Wet**	0	1	0	0
Dry	✓				**Stopped**	1	0	0	0
		✓			**Dry**	1	0	0	0
			✓	*Wetting*		0	0	1	0
			✓		**Wet**	0	1	0	0
Stop	✓				**Stopped**	x	x	0	0
		✓		*Drying*		0	0	0	1
		✓			**Dry**	1	0	0	0
			✓	*Wetting*		0	0	1	0
			✓		**Wet**	0	1	0	0

NOTE 1: x means that the state can be either ‘0’ or ‘1’.

10.7. Table 7. DLEU Support

Code	Description
0	No DLEU supported in this device
1	DLEU is supported in this device

10.8. Table 8. DLEU Mode

Code	Description
0	DLEU Disabled
1	DLEU enabled

10.9. Table 9. DLEU Function Codes

ID Code	M/O	Type	Function Mnemonic	Description
0x0000	M	Logic	NOP	No Operation (i.e., FR Not Configured)
0x0001	M	Logic	CONST	Constant value
0x0002	M	Logic	ISBAD	Check status of input
0x0003	M	Logic	AND	AND logic function
0x0004	M	Logic	OR	OR logic function
0x0005	M	Logic	NAND	NAND logic function
0x0006	M	Logic	NOR	NOR logic function
0x0007	M	Logic	XOR	XOR logic function
0x0008	M	Logic	NOT	NOT logic function
0x0009	M	Logic	2BOOL	Convert packed value to bool
0x000A	M	Logic	BITS	Pick one bit out of a packed byte
0x000B	M	Logic	FTRIG	Falling Edge Trigger
0x000C	M	Logic	RTRIG	Rising Edge Detector
0x000D	M	Logic	LATCH	Latched Coil
0x000E	M	Logic	UNLATCH	Unlatch Coil
0x000F				Undefined
0x0010				Undefined
0x0011	M	Logic	RS	Reset Dominate Bistable
0x0012	M	Logic	SR	Set Dominate Bistable
0x0013	O	Drum/MUX	DRUM_SEQ	Drum Sequencer
0x0014				Undefined
0x0015	O	Drum/MUX	MUX	Select a constant from a Drum Register.
0x0016	O	Counter	CTD	Down Counter
0x0017	O	Counter	CTU	Up Counter
0x0018	O	Timer	TOFF	Off Delay Timer
0x0019	O	Timer	TON	On Delay Timer
0x001A	O	Timer	PULSE	Pulse Timer
0x001B	O	Timer	TIMER	Timer function
0x001C	O	Timer	CLOCK	Clock
0x001D-0xFFFE				Undefined
0xFFFF	M	Logic	END	Must be last DLEU FR to be Executed (FR 0x00000 executed next cycle)

10.10. Table 10. Drum Control Mode

Code	Description
0	Drum Enabled (default)
1	Reset Drum Step. Resets Drum to first step and enables the Drum. Subsequent reads of Drum Control Mode will return "Drum Enabled".
2	Drum Disabled. Drum is resets to first step and remains held at firs t step.

10.11. Table 11. Function Register Status

Value	Description
0	FR has not yet been run
1	FR operation SUCCESS (FR has been executed at least once)
2	Invalid IN1 parameter
3	Invalid IN2 parameter
4	Invalid IN3 parameter
5	Invalid Function ID
6	Invalid OUT parameter
7	DLEU in disabled state

10.12. Table 12. Function Register Fault

Bit	Description
0x01	Ignore IN1 status
0x02	Ignore IN2 status
0x03	Ignore IN3 status

ANNEX A. CONFIGURATION CHANGED AND DISCRETE COMMAND USE TABLE

The following table summarizes the Discrete Commands that must be supported by different types of devices and whether a given command affects the Configuration Changed bit and counter.

Device Type abbreviations are as follows:

DFD	Discrete/Hybrid Field Device	**WDD**	Wireless Discrete/Hybrid Field Device
DA	Discrete Adapter	**WDA**	Wireless Discrete Adapter

The abbreviations in the columns are:

Y	Yes. The Configuration Changed bit and counter are affected	**R(L)**	Recommended if DLEU Supported
N	No. The Configuration Changed bit and counter are un-affected	**M(C)**	Mandatory if Counter/Timer Supported
M	Mandatory. Device must implement the command	**M(D)**	Mandatory if Drum Supported
M(L)	Mandatory if DLEU Supported	**M(O)**	Mandartory if Discrete Outs Supported
M(B)	Mandatory if Burst Mode Supported	**R**	Recommended

Table 27. Discrete Device Command Summary

Command	CC	Device Type			
		DFD	DA	WDD	WDA
64,384 Read Discrete Device Capabilities	N	M	M	M	M
64,385 Read Discrete Variable Properties	N	M	M	M	M
64,386 Read Discrete Variables	N	M	M	M	M
64,387 Write Discrete Variables	N	M(O)	M	M(O)	M
64,388 Simulate Discrete Variables	N	M	M	M	M
64,389 Local Override	N	M(O)		M(O)	
64,390 Write Discrete Variable Classification	Y	R		R	
64,391 Write Discrete Variable Type and Connection Code	Y	R		R	
64,392 Write Discrete Variable Fault Operation	Y	M(O)		M(O)	
64,393 Map Discrete Variable to Dynamic Variable	Y	R			
64,394 Read Burst Discrete Variables	N	R	R	M	M
64,395 Write Burst Discrete Variables	Y	R	R	M	M
64,396 Read PLC Register Mapping	N		M		M
64,397 Map PLC Registers to Discrete Variables	Y		M		M
64,448 Read DLEU Information	N	M(L)		M(L)	
64,449 Write DLEU Mode	N	M(L)		M(L)	
64,450 Reset DLEU	N	M(L)		M(L)	
64,451 Read FR Configuration and Status	N	M(L)		M(L)	
64,452 Write FR Configuration	Y	M(L)		M(L)	
64,453 Drum Control	N	R(L)		R(L)	
64,454 Read Drum Registers	N	M(D)		M(D)	
64,455 Write Drum Registers	Y	M(D)		M(D)	
64,456 Read Counter Values	N	R(L)		R(L)	
64,457 Write Counter Value	Y	M(C)		M(C)	
64,458 Read Timer Values	N	M(C)		M(C)	
64,459 Write Timer Value	Y	M(C)		M(C)	

ANNEX B. DLEU USAGE EXAMPLE: RECIPROCATING MOTION PROCESS

This Annex is included for informational purposes only.

A workpiece must travel back and forth on a conveyor. The location of the workpiece is determined by two limit switches. When the location is detected control signals are sent to a reversing motor contactor. The machine is started and stopped from a local set of push button switches. The following diagram illustrates the process.

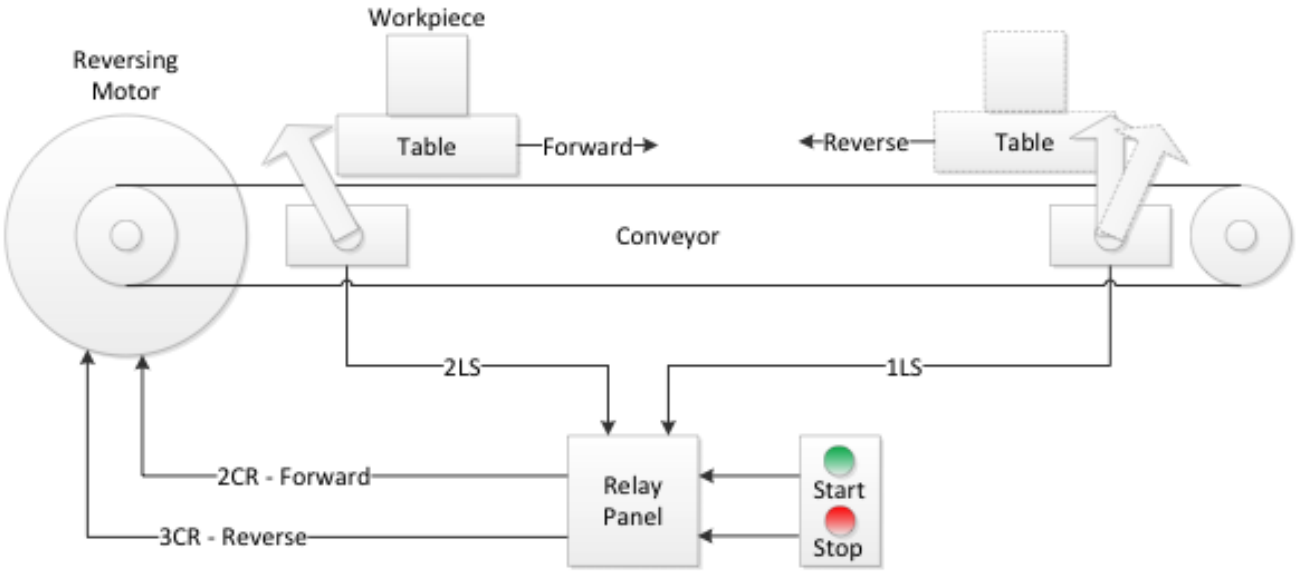


Figure 39. DLEU Example

The States of the system are:

S0: Off
S1: On-Forward
S2: On-Reverse
S3: Off-Forward
S4: Off-Reverse

The IO is defined below.

DLEU	Signal
I0	Start PB
I1	Stop PB
I2	Limit Switch 1 (1LS)
I3	Limit Switch 2 (2LS)
I4	Forward drive cut-off
I5	Reverse drive cut-off
O0	Start motor forward (2CR)
O1	Start motor reverse (3CR)

Figure 40. DLEU I/O Signal Descriptions

Here is the example Ladder Logic:

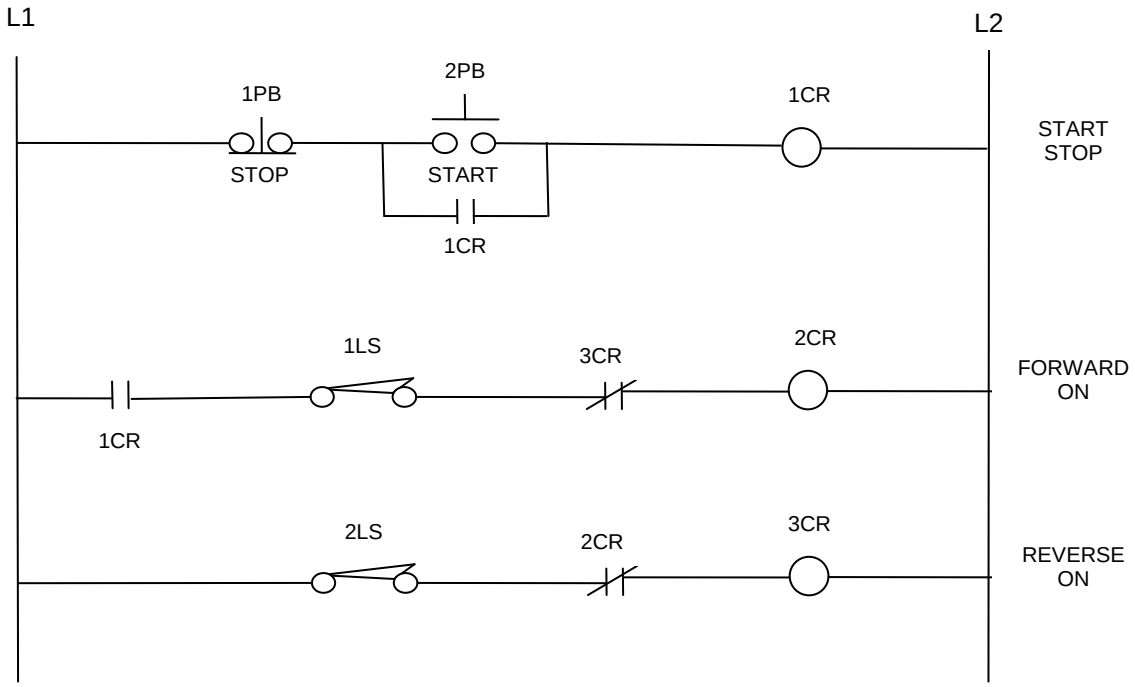


Figure 41. Example Ladder Diagram

The ladder logic can be read as follows:

Rung 1: Start PB. Machine control 1CR seals in around the start PB. Assume the machine begins moving in reverse.

Rung 2: At reverse Limit 2LS is open, 1LS closed, 2CR energized. 3CR de-energized by 2CR.

Rung 3: 2LS closes in forward direction until FWD limit reached. 2CR de-energized 3CR energized and 3CR rung 2 is open.

The Process repeats until Stop PB is pressed

The Discrete Variables and FR's for this example can also be defined. The Discrete Variables along with their information is summarized in the table below:

DLEU	Signal	Type	Connection Code	Fault State	Classification Code
I0	Start PB	Packed	Input from plant	None	Not Classified
I1	Stop PB	Packed	Input from plant	None	Not Classified
I2	Limit Switch 1 (1LS)	Packed	Input from plant	None	Not Classified
I3	Limit Switch 2 (2LS)	Packed	Input from plant	None	Not Classified
I4	Forward drive cut-off	Packed	Input from plant	None	Not Classified
I5	Reverse drive cut-off	Packed	Input from plant	None	Not Classified
O0	Start motor forward (2CR)	Packed	Output to Plant	None	Not Classified
O1	Start motor reverse (3CR)	Packed	Output to Plant	None	Not Classified
R0	Internal Register (1CR)	Packed	No connection	None	Not Classified
R1	Internal Register (2CR)	Packed	No connection	None	Not Classified
R2	Internal Register (3CR)	Packed	No connection	None	Not Classified
R3	Internal Register	Packed	No connection	None	Not Classified
R4	Internal Register	Packed	No connection	None	Not Classified
Target	Target from DCS	State	No connection	Fail to value	Motor
Actual	Actual Value returned to DCS	State	No connection		Motor

Figure 42. Discrete Variables

The configuration for this example is shown in the following figure.

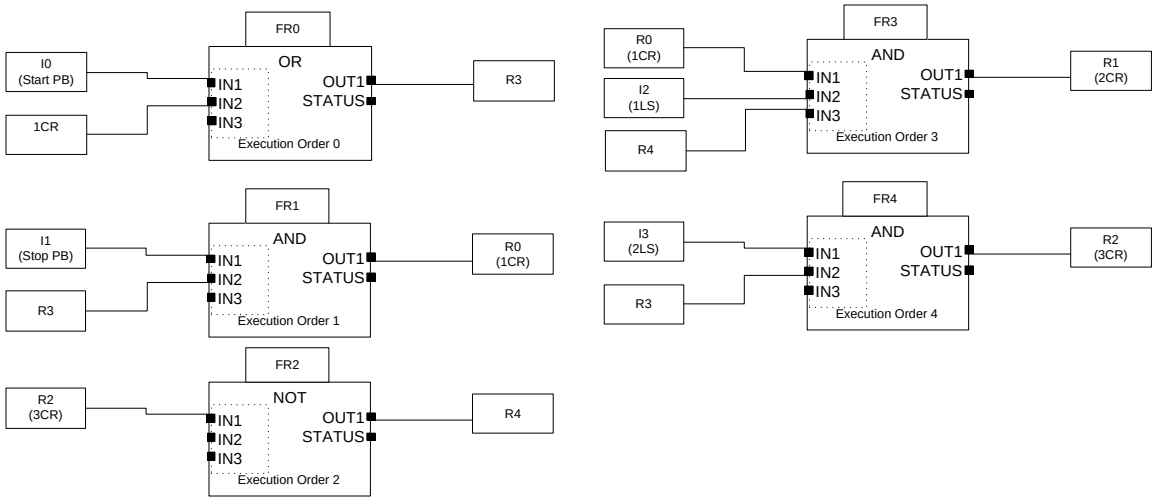


Figure 43. Block Configuration

ANNEX C. OUTPUT WRITE SEQUENCE

This Annex is included for informational purposes only.

The following sequence diagram shows the transactions involved when writing discrete outputs to a field device using continuous burst feedback.

1.) Host can request 1 or more discrete input or output variables. Response will be whatever was last bursted to wirelessHART Gateway.

2.) Host can write 1 or more discrete output variables. In this example, the output immediately transitions to final state, so response to write request contains actual variable state which may or may not match the written target state (e.g. 'START' in request may return 'RUNNING' in reply).

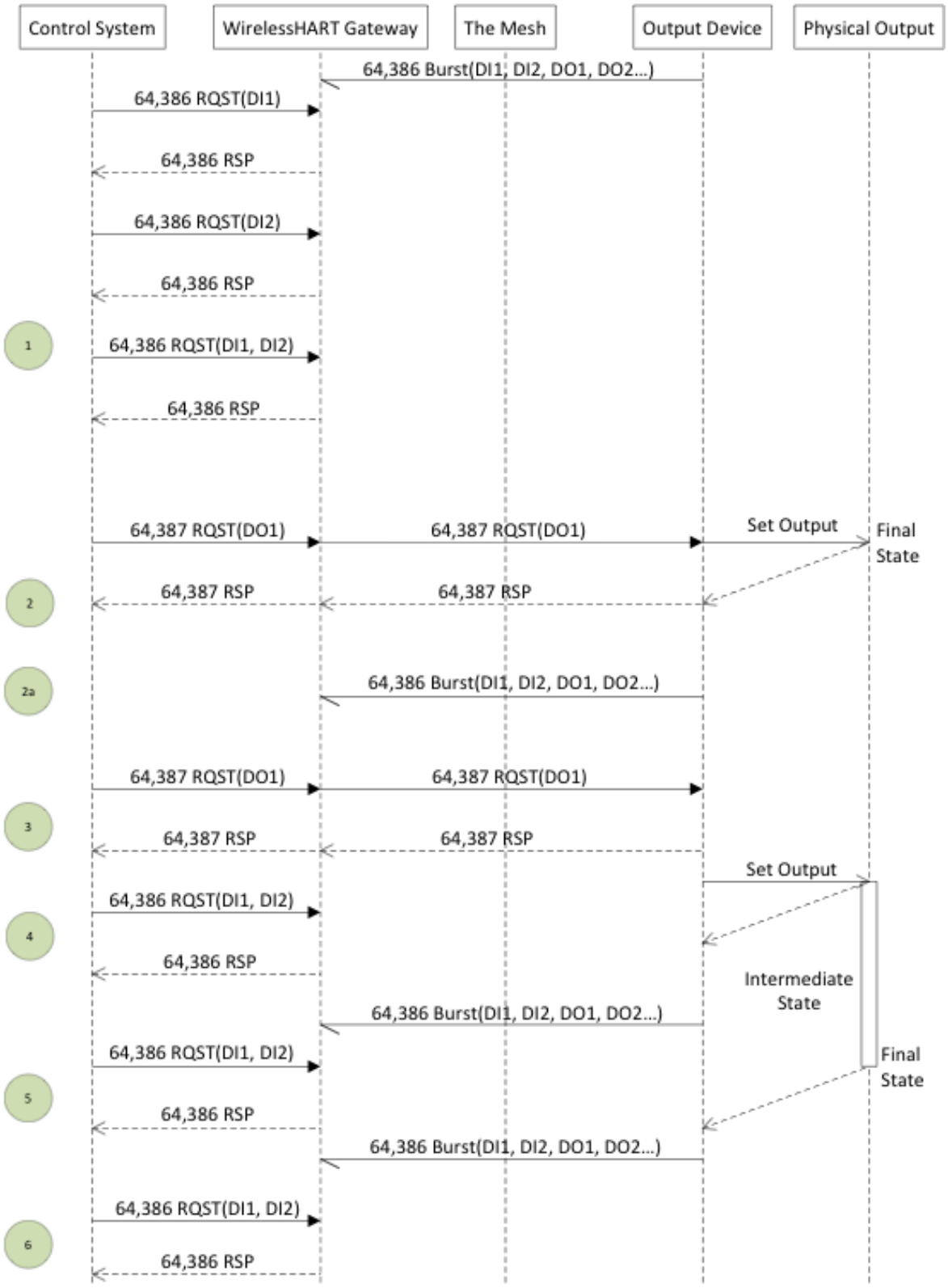
3.) Host writes 1 or more outputs. In this example, the device does not act on the write request immediately. The response is just an echo of the request.

4.) Even though the physical output has changed state to an intermediate state, a burst has not occurred – so the cached write value is still read.

5.) Host now reads an intermediate actual state (e.g. 'OPEN' was written as target in case 3, now 'OPENING' is read back as actual).

6.) Another burst has occurred, now the final actual state is read back.

2a.) In this example, the burst provides redundant information for the written output (because the response of the write already told the gateway the final value of the output). However – other Discrete Variables may have changed as a result of the change in output. These states are communicated in this burst message.


Control System
WirelessHART Gateway
The Mesh
Output Device
Physical Output
64,386 Burst(DI1, DI2, DO1, DO2…)
64,386 RQST(DI1)
64,386 RSP
64,386 RQST(DI2)
64,386 RSP
1
64,386 RQST(DI1, DI2)
64,386 RSP
64,387 RQST(DO1)
64,387 RQST(DO1)
Set Output
Final State
2
64,387 RSP
64,387 RSP
2a
64,386 Burst(DI1, DI2, DO1, DO2…)
64,387 RQST(DO1)
64,387 RQST(DO1)
3
64,387 RSP
64,387 RSP
Set Output
64,386 RQST(DI1, DI2)
4
64,386 RSP
Intermediate State
64,386 Burst(DI1, DI2, DO1, DO2…)
64,386 RQST(DI1, DI2)
Final State
5
64,386 RSP
64,386 Burst(DI1, DI2, DO1, DO2…)
64,386 RQST(DI1, DI2)
6
64,386 RSP

The following sequence diagram shows the transactions involved when writing discrete outputs to a field device using triggered burst feedback.

1.) Host can request 1 or more discrete input or output variables. Response will be whatever was last bursted to wirelessHART Gateway.

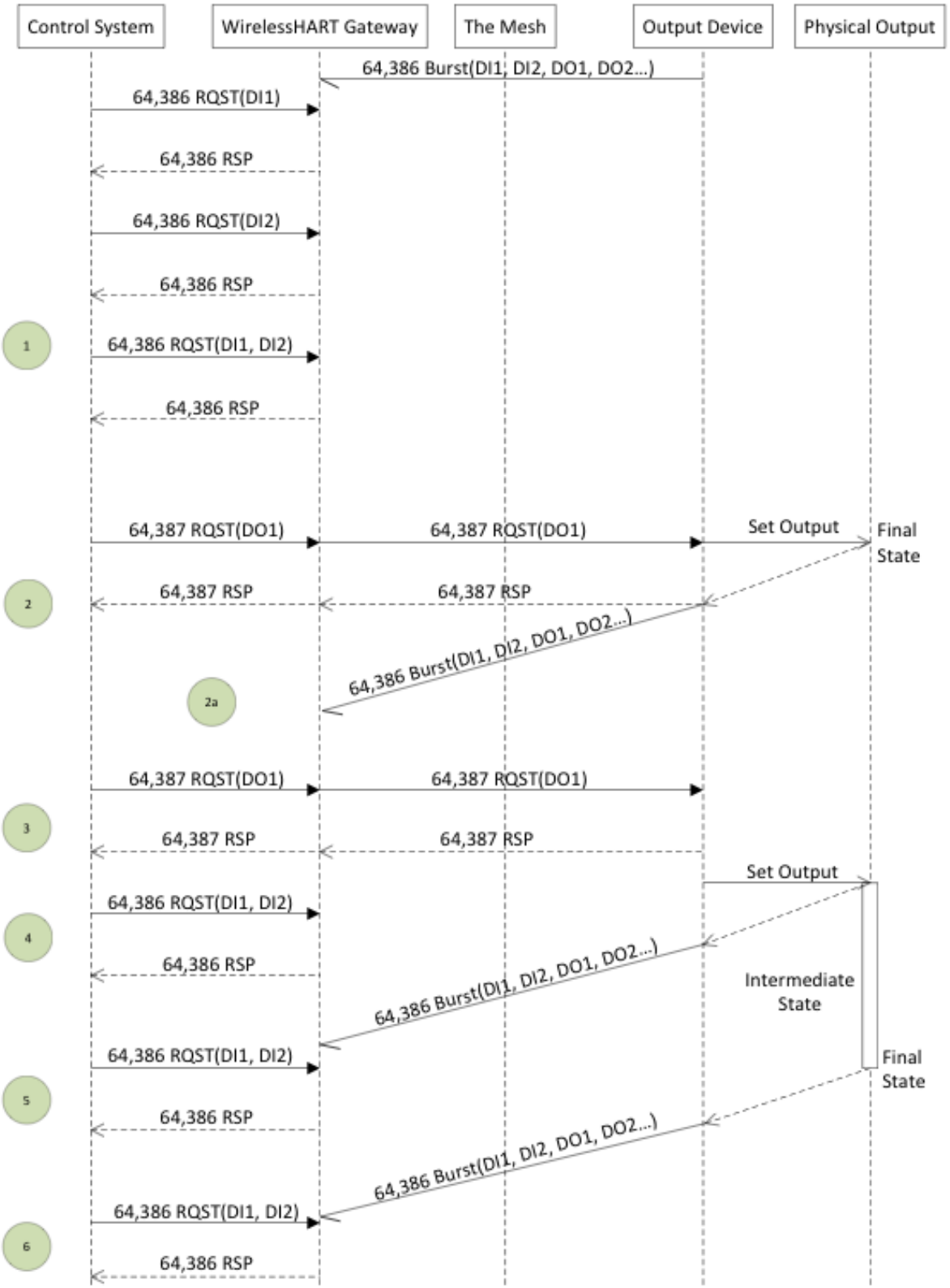
2.) Host can write 1 or more discrete output variables. In this example, the output immediately transitions to final state, so response to write request contains actual variable state which may or may not match the written target state (e.g. 'START' in request may return 'RUNNING' in reply).

3.) Host writes 1 or more outputs. In this example, the device does not act on the write request immediately. The response is just an echo of the request.

4.) Because a burst is triggered on change of output state, the host can sometimes discover the change in physical output actual state with lower latency.

5.) Because the change in output to final state triggers a burst message, the host can sometimes discover the final actual state of the physical output with lower latency.

2a.) In this example, the burst provides redundant information for the written output (because the response of the write already told the gateway the final value of the output). However – other Discrete Variables may have changed as a result of the change in output. These states are communicated in this burst message.


Control System
WirelessHART Gateway
The Mesh
Output Device
Physical Output
64,386 Burst(DI1, DI2, DO1, DO2...)
64,386 RQST(DI1)
64,386 RSP
64,386 RQST(DI2)
64,386 RSP
1
64,386 RQST(DI1, DI2)
64,386 RSP
64,387 RQST(DO1)
64,387 RQST(DO1)
Set Output
Final State
2
64,387 RSP
64,387 RSP
64,386 Burst(DI1, DI2, DO1, DO2...)
2a
64,387 RQST(DO1)
64,387 RQST(DO1)
3
64,387 RSP
64,387 RSP
Set Output
64,386 RQST(DI1, DI2)
4
64,386 RSP
64,386 Burst(DI1, DI2, DO1, DO2...)
Intermediate State
64,386 RQST(DI1, DI2)
Final State
5
64,386 RSP
64,386 Burst(DI1, DI2, DO1, DO2...)
64,386 RQST(DI1, DI2)
6
64,386 RSP

ANNEX D. INDEX TO DLEU FUNCTIONS

This Annex is included for informational purposes only.

ANNEX E. REVISION HISTORY

E.1. Revision 2.0 (11 June 2012)

First complete specification.

E.2. Revision 1.0 (5 September, 2007)

Initial Revision. This revision includes only enough information to act as a placeholder for future development of Discrete Application Layer commands and related items.