APPLICATION GUIDE

FieldComm Group™

Connecting the World of Process Automation

HART® COMMUNICATION PROTOCOL

HART DeviceInfo - Technical Overview

FCG AG21073
Revision 2.2

Release Date: 23 January 2023

Release Date: 23 January 2023

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

FIELDCOMM GROUP™
Connecting the World of Process Automation

FieldComm Group
9430 Research Boulevard
Suite 1-120
Austin, TX 78759, USA

Voice: +1 512-792-2300
FAX: 1-512-792-2310

http://www.fieldcommgroup.org

Table of Contents

Table of Figures

Index to Tables

Introduction

The target audience for this document is Operational (OT) and Information Technology (IT) consumers of HART runtime data and assumes limited knowledge of HART Technology. Any Host Application developer desiring metadata associated HART runtime or status data will also find this document beneficial.

The HART Protocol includes comprehensive Application Layer that supports a broad range of applications. Core to the technology are requirements that ensure meaningful applications can be developed without the need of any materials beyond the HART Specifications themselves. In fact, many applications have been developed with little more than the *HART Technical Overview*. The key to this flexibility and ease of use includes

- Well-defined data types (Unit Codes, IEEE floating-point, integers, strings, standardized statues, etc.);
- Universal Commands that all Field Devices must support exactly as specified; and
- Common Practice Commands, many of which are supported by a majority of Field Devices.

For example, using only the Protocol, the "Dynamic Variables" can be accessed and the Primary Secondary, Tertiary, and Quaternary Variables displayed. Of course these labels are very generic/abstract. Using the Common Tables Specification the corresponding Unit Codes can be translated into human-readable strings. For Command 48, the status bits can be displayed (as on/off) but their description and meaning cannot be easily shown (it at all).

While many valuable applications exist only using the Protocol, simple to use additional metadata about the Field Device is widely requested by PLC and DCS vendors to display Command 48 status and requested by HART-IP Clients that are not HART-knowledgeable. DeviceInfo files are designed to provide metadata about the runtime and status data to facilitate continued growth of simple, valuable HART-enabled applications.

1. OVERVIEW

The HART mantra is: simple, easy-to-use, reliable, and high-value. HART communication is relatively simple for Host Applications. The essential runtime and status data commands are well defined and universally supported in the devices. The field devices are installed and the data are there ready to be consumed by the Host Applications. However, HART communication is binary and, for some, comprehending and consuming binary data can be a barrier to its use. For example, unit codes are returned with all HART process values. However, that code is a number and the text associated with the code (ºC, kg, mA, etc.) would need to be known to the host (all unit codes and their strings) are specified in the *Common Tables Specification*).

Utilizing the HART data can be simplified by providing semantics for the binary data. Providing semantics and templates for HART field devices is the core objective of DeviceInfo technology. DeviceInfo technology makes it easier to utilize the Runtime Data and Status found in the 40-million+ HART field devices.

DeviceInfo extracts metadata from the field device's EDD encoded file based. The extracted metadata focuses on runtime and status data and is a tiny portion of the EDD's complete set of Field Device metadata. Device configuration and associated settings is beyond the scope of DeviceInfo. Consequently, the resulting DeviceInfo files require a small amount of space and host processing power. For example, the DeviceInfo files for all available registered HART EDDs require about the same space as a few pictures taken with a smart phone.

The core files are output in both JSON and XML formats. NodeSet files are XML. Host Application developers can use whichever of these formats best suit their experience; tooling and development strategy

1.1 How DeviceInfo can help

DeviceInfo technology provides templates for HART-enabled field devices. Each template corresponds to a specific Expanded Device Type and Device Revision[1]. The template is applicable to all installed field devices with that Expanded Device Type and Device Revision. These templates provide:

- Specifications for parsing HART binary data from Commands;
- Structure to logically organize the data;
- Semantics and metadata for that runtime data and status; and
- Simple information model to enable Host Application access to the transformed binary data

depicts a typical model for utilizing DeviceInfo technology. HART field devices are continuously monitoring and controlling the process. The runtime data they generation is the result. In its simplest realization, a Host Application can use DeviceInfo technology to consume and transform the binary data into a text-based stream. The Device Connection / Context provides basic data about a specific device instance installed in the plant. The Identity information for that device instance is standardized in Command 0[2]. Using the Identity, the DeviceInfo File associated with that Device Type is located and used to instantiate a transform function.

[1] The combination of Expanded Device Type and Device Revision identifies the fixed, invariant set of commands and data items supported by the Field Device. The Expanded Device Type and Device Revision are returned in the Identity Commands (e.g., Command 0) used to initiate communications with a HART field device.

[2] A summary of Command 0 *Read Unique Identifier* can be found in Annex **Error! Reference source not found.**

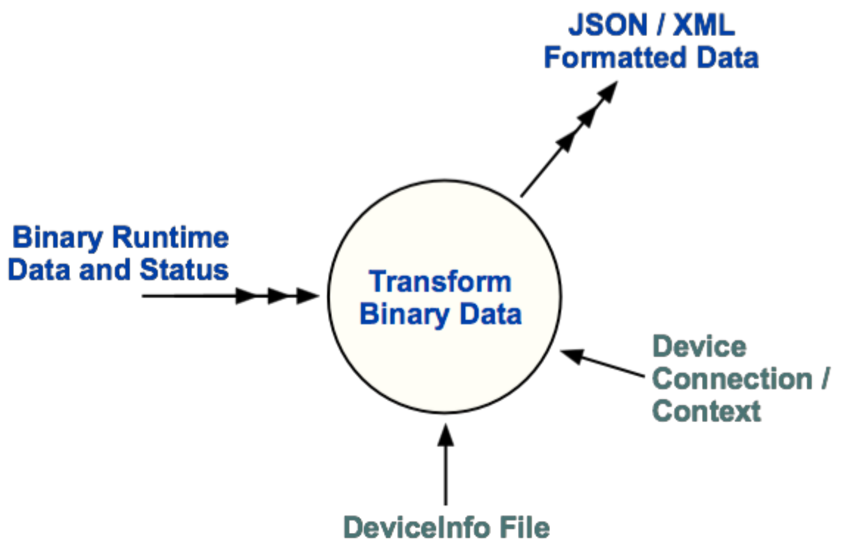

Figure 1. Transforming HART binary data using DeviceInfo

The Transform function is used continuously as data is received from the field device. As binary Runtime Data and Status is received the commands are parsed and the data transformed using the metadata from the DeviceInfo file. The resulting Formatted Data contains the current values and their associated metadata. For example:

- IEEE floating-point data is converted to ASCII using the printf formatting specified by the EDD.
- Engineering unit *codes* are converted to unit code strings.
- Status data bits are converted to human-readable strings; and
- Labels (and help strings) are added to the formatted values

Figure 2 shows one possible pseudo code corresponding to the diagram in Figure 1. The basic idea is to create a DeviceInfoTransform object for each field device instance. Then use that object to transform the incoming runtime and status data. The runtime and status data could be published by the device[3] or could be polled by the Host Application. The object is instantiated using the Device Identity information found in Command 0. The DeviceInfo file is identified using the Expanded Device Type and Device Revision[4]. This DeviceInfo File is used to instantiate a field device database within the object. Upon receiving the device's runtime or status data (i.e., a command), the binary data is used to update the object's database and serialize the data for the client application.

This example is merely one possible way to use DeviceInfo technology. DeviceInfo technology could be used on an I/O System to create a generic server application; used to process and forward annotated data to a SCADA system or Data Historian; or used to push HART runtime and status data to cloud based services like Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform. Section 2 discusses several possible applications in detail. Furthermore, a companion document, "*HART DeviceInfo - NodeSet Files*" provides specific guidance for using DeviceInfo technology with OPC UA servers and clients.

[3] All WirelessHART and HART-IP field device must support publishing. Most HART 4-20mA devices support publishing. HART-IP Remote I/O must support publishing.

[4] The device identity information includes the Device ID. The Device ID is like a serial number and unique for every device having that Expanded Device Type. Consequently, concatenating the Expanded Device Type and Device ID forms the HART "Unique ID" and identifies a unique field device instance. The field device can be tracked through its lifecycle using it Unique ID.

The DeviceInfoTransform object is an example of the process to convert binary HART command data into human readable XML or JSON text. In this example, one DeviceInfoTransform instance would be needed for each instance of a field device.

```
DeviceInfoTransform {
      context             // A copy of the Command 0 Response
      deviceInfoDB        // The DeviceInfo temple for this field device instance
      formattedData       // The resulting formatted data produced from the binary command
                          // response from the field device instance
```

The constructor - instantiates a DeviceInfoTransform using the Command 0 Response from the field device instance. Host applications usually start a conversation with a field device by identifying it using Command 0 "*Read Unique Identifier*"

```
      DeviceInfoTransform ( command0Data ){
           context = command0Data
           get ExpandedDeviceTypeCode , DeviceRevision from context
           lookup DeviceInfo file using ExpandedDeviceTypeCode , DeviceRevision
           instantiate deviceInfoDB from DeviceInfo file
      }
```

The Transform method does the work. It takes binary command data; looks up the command number in the deviceInfoDB and transforms the binary data into (XML, JSON) formatted text.

```
      char [] Transform ( char *commandBinary){
           get CommandNumber from commandBinary
           find the command spec in deviceInfoDB using the CommandNumber
           parse commandBinary using the command spec
           store the parsed data in the deviceInfoDB
           generate the formattedData for the command from the deviceInfoDB
           return the formattedData
      }
}
```

Figure 2. Example pseudo code for transforming HART binary data using DeviceInfo

1.2 A Simplified HART Application Layer Data Model

The DeviceInfo file design is derived from a simplified data model of the HART Application Layer. While HART field devices can be very sophisticated and contain many data items, the data mode shown in Figure 3 reduces the HART Application Layer to the runtime and status data. HART DeviceInfo focuses on clients that consume HART runtime and status data as opposed to the complexities that must be accommodate by Plant Asset Management (PAM). There are five major constituents of the DeviceInfo file:

- The Device Type and DeviceInfo file identification (Device-Type-Model);
- Communications specifications;
- Runtime Data accessors;
- Semantic Map specifications; and
- Data specifications (DataModel).

The Device-Type-Model identifies: (1) the revision of the DeviceInfo format for the file; (2) the device type and device revision this file (i.e., this DeviceInfo template) applies to. Metadata provided includes: the Manufacturer, Private Label Distributor[5], and Device-Type names (strings) for the device. In other words, once the Host Application identifies the device, the company and device-type strings can be displayed.

[5] Hosts display the Private Label Distributor (not the actual Manufacturer).

The Communications subsection includes a command specification for each HART command supported by DeviceInfo. The command definition includes specifications for Request-Data and Response-Data. These reference data-item definitions in the Data-Model. The order of the references in the requests and response list match those in the HART Command Specifications and the binary data sequence found in the HART communications. The references to the Data-Model lead to primitive data types that include VarSizeof (the data-item size in bytes) allowing the binary communications to be parsed.

The command specifications support the device-facing interface of a HART-enabled application (effectively the OT communications). The definitions found in the Communications subsection facilitate the parsing commands responses (polled or published) from the field device and the transformation of that binary data into IT compatible runtime and status date (when metadate like variable labels and unit code strings).

The Data-Model is central to the simplified Application Layer model used in DeviceInfo files. In addition to primitive data types (floats, enums and bit-enums) aggregated collections of data are also included to clarify relationships between primitive data. For example, two lists of collections are included: ProcessValueStatus and ProcessValue. ProcessValue models the data tuples returned in Command 3 and 33. This tuple is designed to make it clear that these data items belong with each other (i.e., show the unit code with the value). The sequence in the tuple enables the parsing of the tuple from the binary data stream. Likewise, the ProcessValueStatus models the tuple returned in Command 9.

While communication specifications are process facing, three interface are provided for higher level applications: StatusList, RuntimeDataList, and SemanticMapList.

- The StatusList provides a list of all the primitive data-items that contain status or health information in the field device (e.g., a list of Command 48 status data).
- The RuntimeDataList is a list of tuples representing the process values.
- Semantic Map provide a generic access to data items via standardized semantic identifiers.

Detailed specifications of the tags and structures in the DeviceInfo files are defined in Section 3.

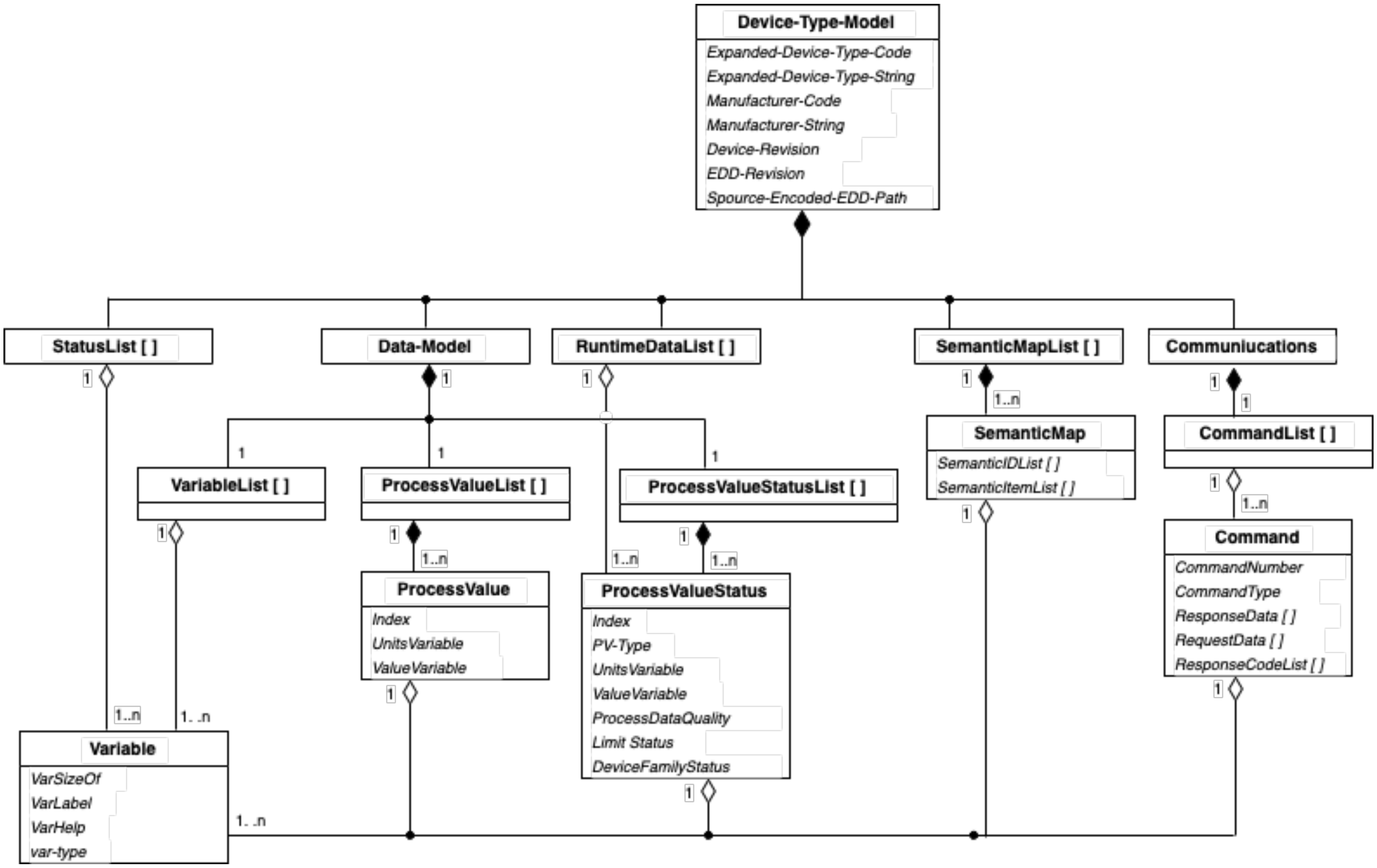

Figure 3. DeviceInfo Model (R.2.2)

1.3 What's in this document

This document provides an overview of DeviceInfo technology, example Applications, the data model that provides the basis for DeviceInfo and the defines the format of the generic, "core" DeviceInfo files

DeviceInfo is based on a simplified model (see Figure 3) of the HART Application Layer and Section 3 provides the details of the DeviceInfo file syntax and tags. Section 4 specifies the format of DeviceInfo filenames. Core DeviceInfo files come in two formats: XML and JSON. While the notation (syntax) is different between the JSON and XML, for DeviceInfo files the underlying object model, structure and keywords are the same.

This document uses some specialized terms and acronyms. For reference, the jargon used in this document is summarized in Annex D. Also, references to supplements HART Specifications and other relevant documents can be found in Annex C.

Forward Compatibility rules govern the numbering the DeviceInfo file format revision (see Annex E). They also govern what enhancements are allowed and their impact on the revision numbering. The fundamental objective (like with the HART Protocol) is to ensure that Host Applications can safely ignore data that are unknown to them. In other words, in minor revisions data, semantics, and tags can be added only as long as the meaning of the existing data, semantics, and tags are unaffected. HART calls this "Forward Compatibility".

The next section (Section 2) highlights several potential applications of DeviceInfo technology.

2. APPLICATIONS

In this section four applications are presented. The objective is to illuminate several possibilities DeviceInfo offers to facilitate Host Application development or improvement. The Applications include

- Embedding field device centric web pages in a Remote I/O;
- Enhancing field device faceplates on operator consoles;
- Simplifying the consumption / utilization of HART data in IT-centric Applications; and
- Enabling HART data transport to the cloud.

Using DeviceInfo to transform binary HART communication is essential to all of these applications. However, where the transformation takes place and the utilization of the results vary.

2.1 Browser on a Remote I/O

There is a growing number of intelligent HART-enabled Remote I/O for both HART 4-20mA and WirelessHART networks. Often these Remote I/O (like most inexpensive printers) include a web interface for simple configuration and troubleshooting. DeviceInfo enable the Remote I/O to include web pages for the connected field devices. The field-device web pages can be generated using the DeviceInfo file and include (in human readable form) pages/tabs like the following:

- **Device Identity**. Basic Device Identity (Device Tag, Private Label Distributor, Device Type, Device Revision, Device ID, HART Protocol Revision, etc.);
- **Device Status**. Detailed device status including the meaning of every status bit; and
- **Process Values**. The field device's current process values including the label, value, units code string, and data quality.

Remote I/O often have limited memory and DeviceInfo is well suited to this application. For example, the entire library of registered HART EDDs / FDI Device Packages or many gigabytes are reduced to by a factor of a 100 or more (down to about the size of about 2 cell phone pictures).

2.2 Faceplate on operator consoles

HART runtime and status data are often displayed on operator consoles. The runtime and status data commands are universal, well documented, and generic device faceplates are relatively simple to develop and deploy. However, demand for device-specific tailoring dates back to the 1990's - especially for decoding of Command 48. DeviceInfo contains the metadata to greatly simplify creation of device-specific faceplates.

Typically, the control system I/O captures the HART data and provides it to the controller and to the operator console. Process data is captured continuously. At the operator station, DeviceInfo can be used continuously or on demand (when the faceplate is displayed) to transform the binary communications by adding labels, unit code symbols, etc. to the faceplate being viewed. Status data is generally captured only when the HART "More Status Available" bit is set. DeviceInfo provides definitions (strings) for each bit in Command 48 *Read Additional Device Status*.

2.3 Bridge to IT

Since its first field use in 2009, HART-IP support in applications like historians (OSIsoft Pi) and databases (SAP, Oracle) continue to grow. HART-IP enables the publishing of runtime and status data from the field device through Remote I/O over the Internet to IT-centric applications. While HART-IP is relatively simple to support[6], the binary HART communication payload limits growth of it-centric applications. DeviceInfo reduces barriers to the development of IT-centric applications by providing:

- JSON/XML command definitions for parsing incoming HART communications; and
- Providing metadata and semantics about the runtime and status data.

HART-IP devices (Remote I/O and field devices) support multiple clients and HART-IP clients are relatively simple to develop[7]. For simple HART-IP clients, the required HART-specific knowledge is limited to:

- Establishing an IP connection to the HART-IP server
- Knowing how to construct binary HART Commands
- Identifying the server using Command 0 (Device Identity) and Command 20 (Device Tag)
- Subscribing to the runtime and status data (Command 533)

These steps are demonstrated in the example Windows HART-IP client and discussed in the *HART-IP Client Guide*. Once these are complete the HART-IP client uses the DeviceInfo files to transform the binary runtime and status data (see Subsection 1.1) JSON or XML data records.

[6] An example HART-IP client is available on GitHub - https://github.com/FieldCommGroup/WindowsHartIpClient.
[7] See the *HART-IP Client Guide*.

2.4 IIoT - HART data in the cloud

IIoT and big data success depend on the availability and consumption of field device data. With over 40 million installed HART-enabled devices the amount of stranded data is vast. DeviceInfo can facilitate the transport of HART field device data to big data repositories in the cloud (e.g., Amazon Web Services, Microsoft Azure, Google Cloud Platform, IBM Cloud, etc). Typically the data flow would be in order:

- HART field device ➔
- Remote I/O (HART-IP, proprietary I/O, HART multiplexer, etc.) ➔
- Edge Gateway ➔
- Cloud Service (Amazon Web Services, Microsoft Azure, Google Cloud Platform, IBM Cloud, etc)

The Edge Gateway can be standalone or a function embedded in existing IT/OT equipment. The Edge Gateway serves as a data concentrator and message broker using MQTT, AMQP or other technology to push the HART runtime and status data to the cloud. Within the Edge Gateway DeviceInfo facilitates the transformation of binary HART communication into JSON of XML plain text for publishing to the cloud.

3. DEVICEINFO SYNTAX

DeviceInfo file design is derived from a simplified model of the HART Application Layer. The core concept is to provide Host Applications with specifications for decoding HART Command request and responses and the metadata for the corresponding data-items. Figure 3 shows the data model reflecting the contents and structure of the DeviceInfo files. There are three major constituents of the DeviceInfo file:

- The Device Type and DeviceInfo file identification;
- Command specifications (Communications); and
- Data specifications (DataModel).

The Device-Type identification provides information on what revision of the DeviceInfo format is contained in the file and identifying information about the device the file applies to. The command specifications include one definition for each command contained in the file. The data specifications contain metadata associated with each data-item found in the commands.

3.1 Device-Type

Table 1 shows the tags providing information about the specific DeviceInfo file. This includes file information (SDIRevision; SourceEncodedDDPath; EDDRevision; EDDErrors; and EDDWarnings) and Field Device Identity information (ExpandedDeviceTypeCode; ExpandedDeviceTypeString; ManufacturerCode; ManufacturerString and DeviceRevision).

Applications should use SDIRevision to confirm it is compatible with the revision of the DeviceInfo file format.

The ExpandedDeviceTypeCode and DeviceRevision match that embedded in the DeviceInfo filename.

Table 1. Tags for Device-Type Metadata

SDIRevision	Indicates the revision of the file format specification used for the DeviceInfo file. This is a decimal number. The major revision is found to the left of the decimal and the minor to the right.
ExpandedDeviceTypeCode	Indicates the Expanded Device Type code for the device this file is modeling. The code is a decimal integer.
ExpandedDeviceType String	The device's product name string found in the EDD.
ManufacturerCode	Indicates the Manufacturer ID Code (in decimal)
ManufacturerString	The Manufacturer name string found in the EDD.
DeviceRevision	The Device Revision number
EDDRevision	(Diagnostic use only) The revision of the EDD used to create the DeviceInfo file
SourceEncodedDDPath	(Diagnostic use only) The relative path to the EDD used to create this DeviceInfo file.
EDDErrors	(Diagnostic use only) DeviceInfo tools require all EDDs to be HART Compliant. Any errors encountered are listed using the EDDErrors tag.
EDDWarnings	(Diagnostic use only) DeviceInfo tools require all EDDs to be HART Compliant. Any warnings encountered are listed using the EDDWarnings tag.

There are a number of tags that are widely used in the DeviceInfo file (see Table 2). For example, these tags are used in the references found in command specifications to providing access into the data model.

Table 2. Common Tags

SymbolName	All primitive items have a symbol name. this is analogous to a identifier in a programming language. SymbolName is used in Reference to link (for example) an entry in a Command to the primitive Variable.
SymbolNumber	Every SymbolName also has a decimal SymbolNumber
Reference	Reference is used to connect items together. For example, Reference connects Command ResponseData and ProcessValue to primitive Variable.
Index	When referencing a list, the index is provided. Indexes are found embedded in the list itself and in references to items in the list. Indexes may include an integer constant or a reference. The reference ultimately resolves to an integer variable.

3.2 Data Specifications

Data Specifications are referenced from Commands to allow the binary stream to be parse into its constituent elements. The DataModel is the top-level section and is divided into three main abstractions (see Table 3): ProcessValueList representing the slots found in Command 3 and 33; ProcessValueStatusList (slots from Command 9) and the VariableList containing the primitive, atomic data-items.

Table 3. Tags for Data Specifications

DataModel	The section containing the metadata for the data-items found in the commands. There are three lists: VariableList (the primative data-items); ProcessValueList (tuples modeling slots in Command 3); and ProcessValueStatusList (tuples modeling the slots in Command 9)
VariableList	The list of all primitive data-items.
ProcessValueList	This list of Dynamic Variables found in Command 3 - one entry (tuple) per slot.
ProcessValueStatusList	The list of all Device Variables. This list models the structure of the slots returned in Command 9

3.2.1 Primitive Variables

Variables are the primitive, atomic data element communicated by the HART Protocol. Table 4 specifies the unique tags used when defining a variable's metadata.

Table 4. Tags for Variables

Variable	Indicates start of a variable's description
VarSizeof	The size (in bytes) of the variable
VarLabel	The label associated with the variable
VarHelp	A help string used to clarify function of the variable
VarType	Indicates the type of the variable
Float	Single Precision IEEE floating point
Unsigned	Unsigned integer
BitEnum	Bit enumerated (bit mask)
Enum	Enumerated. Each numeric value has specific meaning
Date	3 binary bytes containing Day, Month, and (Year-1900) respectively.
Packed	HART Packed ASCII string
Latin-1	ISO Latin-1 string
VarBitEnum	Indicates beginning of the type bit-enum's definition
BitEnumSpec	Each bit definition starts with the BitEnumSpec tag
BitMask	The mask used to capture this bit
BitDescription	The description associated with this bit. i.e., when the bit is set this is what should be displayed.
BitHelp	(Optional) The help text for this bit.
VarEnum	Indicates beginning of the type enum's definition
VarEnumSpec	Each enumeration definition starts with the VarEnumSpec tag
EnumValue	The value of this enumeration
EnumDescription	The description associated with this EnumValue
EnumHelp	(Optional) The help associated with this EnumValue
VarFloat	Indicates beginning of the type float's definition. In HART all multi-byte data, including floats, network byte order is most significant byte first (i.e., big endian)
DisplayFormat	How the floating-point number should be formatted for display. The DisplayFormat contains a format string per the specification found in ANSI-C for printf
VarUnsigned	Indicates beginning of unsigned integer definition. Unsigned may include multiple bytes and network byte order is Most Significant Byte first (i.e., big endian). VarUnsigned generally will include a DisplayFormat.

3.2.2 Process Values

The ProcessValueList contains all process values defined in the Field Device. Items in this list are referenced from, for example, Command 3 and 33. Table 5 specifies the unique tags used within the ProcessValueList.

Table 5. Tags for Process Values

ProcessValue	ProcessValueList.
UnitsVariable	Indicates the unit code associated with this process value
ValueVariable	A reference to the variable containing the numeric value (float) associated with this process value

3.2.3 Process Values with Status

The ProcessValueStatusList contains all process values defined in the Field Device along with the status and classification of that process value. Items in this list are referenced from Command 9. Table 6 specifies the unique tags used within the ProcessValueStatusList.

This list will not exist if Command 9 is not contained in the Field Device (i.e., the device is HART 5).

Table 6. Tags for Process Values with Status

ProcessValueStatus	The start of an element in the ProcessValueStatusList
DeviceVariable Classification	Indicates the kind of Device Variable (e.g., pressure, temperature, mass flow, etc.)
LimitStatus	Indicates whether the Process Value is limited (e.g., low-limited by the Lower Transducer Limit)
ProcessDataQuality	Indicates the "goodness" of the Process Value
DeviceFamilyStatus	Provides specialized status information. For example, a sensor break on a Process Value complying with the Temperature Device Family.
Mask	LimitStatus, ProcessDataQuality, and DeviceFamilyStatus are packed into the same byte. Masks are provided to extract each field from the byte.

3.3 Host Application Access

Three tags are supplied to access for higher level applications: StatusList, RuntimeDataList and SemanticMapList (see Table 7). These provide organized, generic access to the HART Application Layer data. This data may already be cached or may result in polling (command dispatching) by the underlying DeviceInfo application.

Table 7. Tags Host Application Access

SemanticMapList	This is an entry point into the DataModel that allows host access to all Semantic Identifiers. It is a list of SemanticMap tuples.
StatusList	This is an entry point into the DataModel that allows host access to all field device status. It is a list of all the Variables that contain status information (i.e., data from Command 48, device status, communication status and the configuration change counter).
RuntimeDataList	This is an entry point into the DataModel that allows host access to all process values. It is a list of the process value tuples (i.e. the Device Variables) with references to Variable for units, value, data quality, etc. In this case the list has the same entries as ProcessValueStatusList.

The SemanticMapList contains all the SEMANTIC_MAP definitions from the EDD. These definitions are added to the DeviceInfo using the tags found in Table 8.

This list will not exist if no Semantic Maps are found in the EDD.

Table 8. Tags for Semantic Maps

SemanticMap	The start of an element in the SemanticMapList. Each SemanticMap consists of a SemanticIdList and a SematicItemList
SemanticIdList	The start of the list of Semantic Ids
SemanticId	The Semantic ID. Fundamentally each Semantic ID is a text string
SematicItemList	A list of references to the associated EDD Variables for this Semantic Map entry.
SemanticItem	A reference to an EDD Variable. .

The SemanticItem references and EDD VARIABLE and normally the metadata associated with the that VARIABLE will be used by the DeviceInfo Application. However, two modifiers to a SemanticItem are allowed and either may be (optionally) preset

- A "CONSTANT_UNIT" may be specified for displaying as the engineering unit associated with SemanticItem; and
- An enumeration list may be specified. If present, the descriptions in the list are used when the SemanticItem takes on the value corresponding to a specified enumeration entry.

3.4 Command Specifications

DeviceInfo is designed to allow Applications to parse HART Commands without needing any prior knowledge, or even without purchasing HART Protocol Specifications. The keywords (tags) used to structure the command metadata are shown in Table 9.

Key to command specifications are the RequestData and ResponseData lists in the command. These lists satisfy three objectives: (1) enables the binary stream to be decomposed into data-items and (2) structures related data items into cohesive aggregated data sets for easier Application understanding.

DeviceInfo allows the Commands to be parsed (decomposed) into constituent data-items. For example, Command 3 consists of the loop current (a simple Variable reference) and set of four of Dynamic Variables (references, along with an index, to ProcessValues). To parse the Command 3 the Host Application must sequence through the list of references in the ResponseData list. Each reference resolves to a Variable that has a VarSizeof. VarSizeof specifies how many bytes from the binary stream to consume for this Variable. This allows the binary stream to be parsed as the list of references in ResponseData is sequenced.

DeviceInfo structures the data for easier Application consumption. As can be seen in Figure 3, the ResponseData can be a reference to {Variable, ProcessValue, or ProcessValueStatus}. In DeviceInfo, Dynamic Variables (from Command 3) are represented as ProcessValue sets containing {UnitVariable, ValueVariable}. In other words, a ProcessValue reference is placed in the ResponseData list for each Dynamic Variable. This enables an Application to explicitly recognize the units code associated with a process value.

Table 9. Tags for Command Specifications

Communications	The section of the DeviceInfo file specifying commands
Command	All command descriptions begin with the tag Command
CommandNumber	Identifies the command number
CommandType	Identifies the type of command {READ, WRITE, COMMAND}
RequestData	Tag indicating the data in the command request (sent to ➔ the device)
ResponseData	Tag indicating the data in the command response (sent from 🡐 the device)
ResponseCodeList	Lists the command Response Codes
ResponseCode	Specifies an individual Response Code
ResponseCodeValue	The value of the Response Code
ResponseCode Description	The description text.
ResponseCodeStatus	Indicates command execution Success, a Warning or an Error result.

A ResponseCodeStatus of Warning means the Command was executed but not exactly as specified. To a Host Application this means the Field Device executed the Command just with some small adjustment. A warning is returned, for example, when the process value has not been updated since it was last read. i.e., the process variable is being read faster than the device updates it. The Host Application can generally ignore a Warning. An Error means the command was not executed.

4. DEVICEINFO FILE NAMES

In HART, the Expanded Device Type and Device revision specify a single, fixed set of commands and data. Consequently, the DeviceInfo filename consists of the Expanded Device Type and the Device Revision. The EDD revision is irrelevant to DeviceInfo and, therefore, not included in the filename.

The DeviceInfo filename consist of six hexadecimal digits representing the 2-byte Expanded Device Type and the 1-byte Device Revision number. This is followed by ".HDI"[8], ".core" or ".nodeset" and the “json" or ".xml” suffix (see Figure 4).

```
1a8902.HDI.core.json       1d0401.HDI.core.json       3f0601.HDI.core.json
1a8902.HDI.core.xml        1d0401.HDI.core.xml        3f0601.HDI.core.xml
1a8902.HDI.nodeset.xml     1d0401.HDI.nodeset.xml     3f0601.HDI.nodeset.xml
1a9901.HDI.core.json       1f1e01.HDI.core.json       4d0104.HDI.core.json
1a9901.HDI.core.xml        1f1e01.HDI.core.xml        4d0104.HDI.core.xml
1a9901.HDI.nodeset.xml     1f1e01.HDI.nodeset.xml     4d0104.HDI.nodeset.xml
1d0101.HDI.core.json       3f0301.HDI.core.json       4d0105.HDI.core.json
1d0101.HDI.core.xml        3f0301.HDI.core.xml        4d0105.HDI.core.xml
1d0101.HDI.nodeset.xml     3f0301.HDI.nodeset.xml     4d0105.HDI.nodeset.xml
1d0201.HDI.core.json       3f0401.HDI.core.json       4d0205.HDI.core.json
1d0201.HDI.core.xml        3f0401.HDI.core.xml        4d0205.HDI.core.xml
1d0201.HDI.nodeset.xml     3f0401.HDI.nodeset.xml     4d0205.HDI.nodeset.xml
1d0301.HDI.core.json       3f0501.HDI.core.json       4d0301.HDI.core.json
1d0301.HDI.core.xml        3f0501.HDI.core.xml        4d0301.HDI.core.xml
1d0301.HDI.nodeset.xml     3f0501.HDI.nodeset.xml     4d0301.HDI.nodeset.xml
```

Figure 4. Sample DeviceInfo Filenames

[8] HART DeviceInfo

ANNEX A. DEVICEINFO GENERATION REQUIREMENTS

DeviceInfo is designed to provide context and semantics for HART runtime data. Specifically, DeviceInfo allows measurement and optimization applications to apply human readable labels and descriptions to the binary data communicated via HART. Several assumptions are fundamental to DeviceInfo:

- Field Devices are HART compliant;
- The Field Device's runtime data are returned in standard HART Commands; and
- Target host applications must be able to easily utilize DeviceInfo files;

In other words, DeviceInfo files include metadata corresponding to Universal and Common Practice Commands and the Field Device's implementation of those commands **must** comply with HART Protocol requirements. DeviceInfo cannot be generated if the field device edds do not comply with this requirement.

This annex provides background information about DeviceInfo treatment of conditionals and strings encountered in field device EDDs

A.1. Conditionals

No Conditionals are included in DeviceInfo files. For Universal and Common Practice Commands the use of conditionals is limited to Unit Codes and Display formats. The DeviceInfo file provides the first DisplayFormat string encountered.

Some devices include Unit Code conditionals to simplify user configuration of units. DeviceInfo files include all Unit Codes from all possible conditional evaluations. Per HART Specifications, Field Device must return the Unit Code currently configured and the Unit Code returned can then be looked up in the DeviceInfo file.

A.2. Strings

To simplify Host Application implementation, only English strings are provided in DeviceInfo file metadata All strings are provided in UTF-8 format.

ANNEX B. DEVICEINFO FILE SYNTAX SUMMARY

B.1. DeviceInfo

device-info
revision identification diagnostics communications data-model status-set run-time-set semantic-map-list identity-set

revision:
SDIRevision *major-revision . minor-revision*

major-revision:
integer

minor-revision:
integer

identification:
device-type device-type-string manufacturer manufacturer-string device-revision

device-type:
ExpandedDeviceTypeCode *integer*

device-type-string:
ExpandedDeviceTypeString *string*

manufacturer:
ManufacturerCode *integer*

manufacturer-string:
ManufacturerString *string*

device-revision
DeviceRevision *integer*

disgnostics:
file-name edd-errors edd-warnings edd-revision

file-name:
SourceEncodedDDPath *string*

edd-errors:
EDDErrors *string*

edd-warnings:
EDDWarnings *string*

edd-revision:
EDDRevision *integer*

symbol-name:
SymbolName *string*

symbol-number:
SymbolNumber *integer*

B.2. Data Model

data-model:
DataModel *variable-list process-value-list process-value-status-list*

B.2.1. Process Value

process-value-list:
ProcessValueList *process-values*

process-values:
process-value
process-value process-values

process-value:
ProcessValue *list-index units-variable value-variable*

list-index:
Index Const *integer*

units-variable:
UnitsVariable *description help data-reference*

value-variable:
ValueVariable *description help data-reference*

B.2.2. Process Value with Status

process-value-status-list:
ProcessValueStatusList *symbol-name symbol-number label help process-value-statuses*

process-value-statuses:
process-value-status
process-value-status process-value-statuses

process-value-status:
ProcessValueStatus *list-index label help var-classification units-variable value-variable*
process-data-quality limit-status device-family-status

var-classification:
DeviceVariableClassification *description help data-reference*

process-data-quality:
ProcessDataQuality *description help data-refefence data-mask*

limit-status:
LimitStatus *description help data-reference data-mask*

device-family-status:
DeviceFamilyStatus *description help data-reference data-mask*

label:
  Label *string*

help:
  Help *string*

description:
  Description *string*

B.2.3. Variables

variable-list:
  VariableList *variables*

variables:
  variable
  variable variables

variable:
  Variable *symbol-name symbol-number var-label var-help var-size var-type*

var-label:
  VarLabel *string*

var-help:
  VarHelp *string*

var-size:
  VarSizeof *integer*

var-type:
  float-type var-float$_{opt}$
  unsigned-type var-unsigned$_{opt}$
  bit-enum-type var-bit-enum
  enum-type var-enum
  date-type
  packed-type
  latin-1-type

date-type:
  VarType Date

packed-type:
  VarType Packed

Latin-1-type:
  VarType Latin-1

float-type:
  VarType Float

var-float:
  VarFloat *display-format-string*
  VarFloat *display-format-string var-constant-unit*
  VarFloat *var-constant-unit*

```
var-constant-unit
   ConstantUnit string

display-format-string:
   DisplayFormat printf-string

printf-string:
   string

unsigned-type:
   VarType Unsigned

var-unsigned:
   VarUnsigned display-format-string

bit-enum-type:
   VarType BitEnum

var-bit-enum:
   VarBitEnum bit-enum-specs

bit-enum-specs:
   bit-enum-spec
   bit-enum-spec bit-enum-specs

bit-enum-spec:
   BitEnumSpec bit-mask bit-description bit-help_opt

bit-mask:
   BitMask integer

bit-description:
   BitDescription string

bit-help:
   BitHelp string

enum-type:
   VarType Enum

var-enum:
   VarEnum enum-specs

enum-specs:
   enum-spec
   enum-spec enum-specs

enum-spec:
   VarEnumSpec enum-value enum-description enum-help_opt

enum-value:
   EnumValue integer
```

```
enum-description:
   EnumDescription string

enum-help:
   EnumHelp string
```

B.3. Run Time Data List

```
run-time-set:
   RuntimeDataList process-value-statuses-reference

process-value-statuses-reference:
   process-value-status-reference
   process-value-status-reference process-value-statuses-reference

process-value-status-reference:
   Reference SymbolName process-value-status-name-string
   Reference SymbolName process-value-status-name-string reference-index

process-value-status-name-string:
   string
```

B.4. Status List

```
status-set:
   StatusList status-item-list

status-item-list:
   status-reference
   status-reference status-item-list

status-reference:
   Reference SymbolName data-name-string
   Reference SymbolName data-name-string reference-index
```

B.5. Semantic Map

```
semantic-map-list:
   SemanticMapList semantic-maps

semantic-maps:
   semantic-map
   semantic-map semantic-maps

semantic-map:
   SemanticMap symbol-name symbol-number semantic-id-list sematic-item-list

semantic-id-list:
   SemanticIdList semantic-ids
```

```
semantic-ids
    semantic-id
    semantic-id semantic-ids

semantic-id:
    SemanticId string

sematic-item-list:
    SematicItemList semantic-items

semantic-items:
    semantic-item
    semantic-item semantic-items

semantic-item:
    SemanticItem Reference SymbolName variable-name-string
    SemanticItem Reference SymbolName variable-name-string var-enum
    SemanticItem Reference SymbolName variable-name-string var-constant-unit
```

B.6. Communications

```
communications:
    Communications command-list

command-list:
    command
    command command-list

commmand:
    Command symbol-name symbol-number command-number command-type request-data
            response-data response-code-list

command-number:
    CommandNumber integer

command-type:
    CommandType one-of { READ WRITE COMMAND ]

request-data:
    RequestData data-item-list

response-data:
    ResponseData data-item-list

response-code-list:
    ResponseCodeList response-codes

Response-codes:
    response-code
    response-code response-codes

response-code:
    ResponseCode response-code-value response-code-description

response-code-value:
    ResponseCodeValue integer

response-code-description:
    ResponseCodeDescription string

data-item-list:
    data-reference
    data-reference data-item-list

data-reference:
    Reference SymbolName data-name-string
    Reference SymbolName variable-name-string data-mask
    Reference SymbolName data-name-string reference-index

reference-index:
    Index data-reference
    Index Const integer

variable-name-string:
    string

data-name-string:
    string

data-mask:
    Mask integer
```

ANNEX C. REFERENCES

This Annex provides a list of reference documents for developers.

C.1. HART Communications Protocol Specifications

HART Communications Protocol Specification. HCF_SPEC-013, FCG TS20013

Command Summary Specification. HCF_SPEC-099, FCG TS20099

Universal Command Specification. HCF_SPEC-127, FCG TS20127

Common Practice Command Specification. HCF_SPEC-151, FCG TS20151

Common Tables Specification. HCF_SPEC-183, FCG TS20183

C.2. Related Documents

FieldComm Group. *HART DeviceInfo - NodeSet Files*. FCG AG21074

FieldComm Group. *HART-IP Client Guide*. FCG AG21072

Bowden Romilly, *HART® Technology: A Technical Overview*, 2018. ISBN-13: 978-1549862274. See https://www.amazon.com/HART®-Technology-Technical-Romilly-Bowden/dp/1549862278

An example HART-IP client is available on GitHub

FieldComm Group. "Windows HART-IP Client"
https://github.com/FieldCommGroup/WindowsHartIpClient.

ANNEX D. DEFINITIONS, SYMBOLS, AND ABBREVIATIONS

This annex summarizes the vocabulary / jargon used throughout this document.

D.1. Definitions

Device ID A 24bit number returned by HART field devices that is unique for every instance of a given Expanded Device Type. Expanded Device Type + Device ID for the device's Unique ID.

Device Revision The integer returned in byte 5 of Identity Commands. This defines the revision level of the command set supported by the field device including the device-specific commands.

Device Variables See Process Values.

Dynamic Variables The four process values returned in Command 3.

Expanded Device Type The integer returned in bytes 1-2 of Identity Commands (see the Command Summary Specification). This defines the command set supported by a device.

Host Application A computer program interacting with a device and providing functionality to the user. May include master functions or depend on others for the master function.

Identity Commands Host Applications use Identity Commands to identify a field device and begin a communication session with the field device. There are five Identity Commands:

Command 0	Read Unique Identifier
Command 11	Read Unique Identifier Associated With Tag
Command 21	Read Unique Identifier Associated With Long Tag
Command 73	Find Device
Command 75	Poll Sub-Device

All of these commands return the same Response Data Bytes. These commands return the data items necessary to

Determine the command set and data items supported by the field device;
Generate a long frame address; and
Route commands to the appropriate field device.

JSON Java Script Object Notation - Derived from Java Script as a simple mechanism for sharing data. A widely used standard in live web/cloud applications

NodeSet NodeSets are written in XML and specify the interface between an OPC UA server and client.

Process Value A measurement or a setpoint

Runtime Data Runtime data reflect the current process conditions and changes continuously as the connected process varies.

Simple Device A Field Device exposing only Dynamic Variables.

Status Data Status Data indicates the health of the field device, its sensors and its Process Values.

Unique Identifier (ID) The concatenation of the Expanded Device Type and Device ID. These data, when combined, uniquely identify a specific field device. No two devices ever manufactured shall have the same combination of these values.

Units Code An one byte integer that indicates the engineering units (e.g. millibars, meters per second, or degrees Celsius) for the associated data item. In HART, all floating-point numbers have a specified or implied Units Code.

XML Extensible Markup Language. A computer language designed for transferring electronic documents. Derived from SGML - Standard Generalized Markup Language

D.2. Symbols and Acronyms

EDD	Electronic Device Description
ERP	Enterprise Resource Planning
IT	Information Technology
JSON	Java Script Object Notation
LSB	Least Significant Byte. The LSB is always the last byte transmitted over a HART data link
MSB	Most Significant Byte. The MSB is always the first byte transmitted over a HART data link
OT	Operational Technology
PAM	Plant Asset Management
PDU	Protocol Data Unit. The packet of information being communicated.
XML	Extensible Markup Language

ANNEX E. DEVICEINFO FILE REVISION RULES

The revision number of the DeviceInfo File Format is included in every DeviceInfo file and must track the revision of the data model shown in Figure 3. DeviceInfo Model (R.2.2). The revision (xx.yy) includes a major (xx) and minor revision (yy). The major revision shall be incremented when the structure of the DeviceInfo file itself is changed.

The minor revision shall be incremented for any other change. In other words, simple additions are a minor revision. For example, a new data type could be added to Variable as a minor revision.

Client Applications must support forward compatibility. In other words, Client Applications must read the revision number from the DeviceInfo file it is loading. If the Client Application supports the major revision the file must be successfully loaded and utilized. Any additions (e.g., supplementary metadata) encountered when loading a specific DeviceInfo file can be safely discarded without impacting Client Application operation.

ANNEX F. REVISION HISTORY

F.1. Revision 2.2.

This revision:

- Added support for semantic identifiers (SemanticMapList).
- Added syntax for remaining HART data types (Packed, Latin-1, Date) and the DeviceIdentity tag.
- Added syntax Label, Description and Help to ProcessValueStatus.

Since are no structural changes (only additions) this is a minor revision.

F.2. Revision 2.1.

Added new tags: RuntimeDataList, StatusList. Since this is not a structural change (only additions) this is a minor revision

F.3. Revision 2.0 - Initial Revision.

Structural Change (thus major revision). Added "DataModel" tag. Variables, ProcessValueList, ProcessValueStatusList now members of DataModel

F.4. Revision 1.0 - Initial Revision.