IO-Link Profile

BLOB Transfer & Firmware Update

Specification

Version 1.0
June 2016

Order No: 10.082

File name: **IOL-Profile_Firmware-Update_V10_10082_Jun16.doc**

This profile specification has been developed by the IO-Link FW-Update profile group.

Any comments, proposals, requests on this document are appreciated through the IO-Link CR database www.io-link-projects.com. Please provide name and email address.
Login: *IOL-FW-Update*
Password: *Report*

Important notes:

NOTE 1 The IO-Link Consortium Rules shall be observed prior to the development and marketing of IO-Link products. The document can be downloaded from the www.io-link.com portal.

NOTE 2 Any IO-Link device shall provide an associated IODD file. Easy access to the file and potential updates shall be possible. It is the responsibility of the IO-Link device manufacturer to test the IODD file with the help of the IODD-Checker tool available per download from www.io-link.com.

NOTE 3 Any IO-Link devices shall provide an associated manufacturer declaration on the conformity of the device with this specification, its related IODD, and test documents, available per download from www.io-link.com.

Conventions:

In this specification the following key words (in **bold** text) will be used:

may:	indicates flexibility of choice with no implied preference.
should:	indicates flexibility of choice with a strongly preferred implementation.
shall:	indicates a mandatory requirement. Designers **shall** implement such mandatory requirements to ensure interoperability and to claim conformity with this specification.

Publisher:
IO-Link Community
Haid-und-Neu-Str. 7
76131 Karlsruhe
Germany
Phone: +49 721 / 96 58 590
Fax: +49 721 / 96 58 589
E-mail: info@io-link.com
Web site: www.io-link.com

CONTENTS

0 Introduction

0.1 General

The Single-drop Digital Communication Interface (SDCI) and system technology (IO-Link™[1]) for low-cost sensors and actuators is standardized within IEC 61131-9 [1]. The technology is an answer to the need of these digital/analog sensors and actuators to exchange process data, diagnosis information and parameters with a controller (PC or PLC) using a low-cost, digital communication technology while maintaining backward compatibility with the current DI/DO signals as defined in IEC 61131-2.

Any SDCI compliant Device can be attached to any available interface port of an SDCI Master. SDCI compliant devices perform physical to digital conversion in the device, and then communicate the result directly in a standard 24 V I/O digital format, thus removing the need for different DI, DO, AI, AO modules and a variety of cables.

Physical topology is point-to-point from each Device to the Master using 3 wires over distances up to 20 m. The SDCI physical interface is backward compatible with the usual 24 V I/O signalling specified in IEC 61131-2. Transmission rates of 4,8 kbit/s, 38,4 kbit/s and 230,4 kbit/s are supported.

Tools allow the association of Devices with their corresponding electronic I/O device descriptions (IODD) and their subsequent configuration to match the application requirements [2].

This document specifies the common additional protocol means for the transfer of Binary Large Objects (BLOB) and in particular the means for firmware-updates of Devices.

This document follows the IEC 62390 [3] to a certain extent.

Terms of general use are defined in IEC 61131-1 or in [4]. Specific SDCI terms are defined in this part.

0.2 Patent declaration

The IO-Link Community draws attention to the fact that it is claimed that compliance with this document may involve the use of patents concerning FW-Update in case of functional safety devices as follows, where the [xx] notation indicates the holder of the patent right:

EP 1532494 B1	[PI]	Safety control system for fail-safe control of safety-critical processes and method for running a new operating program therein

IO-Link Community takes no position concerning the evidence, validity and scope of this patent right.

The holders of the patent rights have assured the IO-Link Community that they are willing to negotiate licences either free of charge or under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with the IO-Link Community.

Information may be obtained from:

[PI]	Pilz GmbH, Felix-Wankel-Straße 2, 73760 Ostfildern, Deutschland/Germany

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The IO-Link Community shall not be held responsible for identifying any or all such patent rights.

The IO-Link Community maintains on-line data bases of patents relevant to their standards. Users are encouraged to consult the databases for the most up to date information concerning patents.

1 IO-Link^{TM} is a trade name of the "IO-Link Community". This information is given for the convenience of users of this specification. Compliance to this specification does not require use of the registered logos for IO-Link^{TM}. Use of the registered logos for IO-Link^{TM} requires permission of the "IO-Link Community".

PROGRAMMABLE CONTROLLERS —

Profile for the transfer of BLOBs and firmware-updates within systems according to IEC 61131-9

1 Scope

The single-drop digital communication interface (SDCI) technology described in part 9 of the IEC 61131 series focuses on simple sensors and actuators in factory automation, which are nowadays using small and cost-effective microcontrollers. With the help of the SDCI technology, the existing limitations of traditional signal connection technologies such as switching 0/24 V, analog 0 to 10 V, etc. can be turned into a smooth migration. Classic sensors and actuators are usually connected to a fieldbus system via input/output modules in so-called remote I/O peripherals. The (SDCI) Master function enables these peripherals to map SDCI Devices onto a fieldbus system or build up direct gateways. Thus, parameter data can be transferred from the PLC level down to the sensor/actuator level and diagnosis data transferred back in turn by means of the SDCI communication. This is a contribution to consistent parameter storage and maintenance support within a distributed automation system. SDCI is compatible to classic signal switching technology according to part 2 of the IEC 61131 series.

This document specifies the common means for the transfer of Binary Large Objects (BLOBs) between tools, programmable logic controllers (PLC), Masters and Devices as well as associated identification rules for BLOBs, securing measures for uploads, index definitions, definitions for transfer segmentation (> ISDU size), and access conflict resolutions. It is supplemented by recommendations for the encryption and compression of BLOBs.

This document specifies also the common extra protocol means for Device firmware-updates between tools, Master and Devices and the associated use cases; the necessary equipment set-up; commands and messages; file formats; identification, authentication, and validation issues; DeviceID and version rules; parameter versions and data storage rules. It is supplemented by recommendations on encryption and compression of firmware-update files, authentication via signatures, and distribution to multiple destinations within a Device.

This document contains specific IODD extensions according to this profile and instructions for an extension of the IODD-Checker.

This document contains specific test cases for the conformity testing of tools, Masters, and Devices according to this profile.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 61131-2*, Programmable controllers – Part 2: Equipment requirements and tests*

IEC 61131-9, *Programmable controllers – Part 9: Single-drop digital communication interface for small sensors and actuators (SDCI)*

3 Terms, definitions, symbols, abbreviated terms and conventions

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions in addition to those given in IEC 61131-9 apply. For the convenience of readability, major terms and definitios of IEC 61131-9 are repeated.

3.1.1
bidirectional
transfer of data from a tool via a Master to the Device and vice versa

3.1.2
binary large object
BLOB
amount of coherent data to be transferred to or from the Device

3.1.3
Bootloader
Device application responsible for *unlocking* existing firmware, handling of FW-Update binaries and permanent storage, e.g. flashing

Note 1 to entry: The technology specific Bootloader application and the IO-Link communication can be cut down to the minimum functionality required for the purpose of bootloading in case of an unsuccessful update

3.1.4
D-BLOB_Trans_Layer
communication protocol in a Device application for the transmission of data larger than the ISDU length

NOTE 1 to entry: Counterpart is either a P-BLOB_Trans_Layer or T-BLOB_Trans_Layer

3.1.5
Device
single passive peer to a Master such as a sensor or actuator

NOTE 1 to entry: Uppercase "Device" is used for SDCI equipment, while lowercase "device" is used in a generic manner.

3.1.6
download
transfer of data from a tool via a master to the Device

3.1.7
Event
instance of a change of conditions in a Device

NOTE 1 to entry: Uppercase "Event" is used for SDCI Events, while lowercase "event" is used in a generic manner.

NOTE 2 to entry: An Event is indicated via the Event flag within the Device's status cyclic information, then acyclic transfer of Event data (typically diagnosis information) is conveyed through the diagnosis communication channel.

3.1.8
firmware
entire nonvolatile software of a Device consisting of the technology specific application including the *Bootloader* and the communication stack

Note 1 to entry: The technology specific application comprises for example measuring or control algorithms, hardware drivers, and local user interfaces

3.1.9
FW-Update application
computer software *tool* for the purpose of updating a Device's *firmware*

3.1.10
HW_ID_Key
identifier within a Device and within a FW-Update file to ensure that both match

3.1.11
header
data structure containing fields for commands and flow control in a protocol

3.1.12
host
PC or PLC, hosting software tools as counterpart of Device applications

3.1.13
ISDU
indexed service data unit used for acyclic acknowledged transmission of parameters that can be segmented in a number of M-sequences

3.1.14
Master
active peer connected through ports to one up to n Devices and which provides an interface to the gateway to the upper level communication systems or PLCs

NOTE 1 to entry: Uppercase "Master" is used for SDCI equipment, while lowercase "master" is used in a generic manner.

NOTE 2 to entry: This profile has no impact on the design of Master.

3.1.15
On-request Data (OD)
acyclically transmitted data upon request of the Master application consisting of parameters or Event data

3.1.16
P-BLOB_Trans_Layer
communication protocol in PLC user programs for the transmission of data larger than the ISDU length

Note 1 to entry: Usually, the protocol is embedded in a function block (FB) written in the IEC 61131-3 programming language structured text (ST)

Note 2 to entry: Counterpart is located as D-BLOB_Trans_Layer within a Device

3.1.17
port
communication medium interface of the Master to one Device

3.1.18
Process Data (PD)
input or output values from or to a discrete or continuous automation process cyclically transferred with high priority and in a configured schedule automatically after start-up of a Master

3.1.19
ProfileIdentifier
list of supported profiles and function classes

3.1.20
segment
data structure consisting of a *header* and user data

3.1.21
T-BLOB_Trans_Layer
communication protocol in computer tools such as supervisory software for the transmission of data larger than the ISDU length

Note 1 to entry: Counterpart is located as D-BLOB_Trans_Layer within a Device

3.1.22
Technology firmware
permanently stored individual software in a Device providing control, monitoring, and data manipulation to support a particular technology

3.1.23
Tool
software means within controllers or personnel computers for the processing of *BLOBs* or firmware updates

3.1.24
Unlocking
specified sequence of SystemCommands to enable a FW-Updateby the *Bootloader*

Note 1 to entry: This feature prevents the Device from accidental access and firmware changes

3.1.25
upload
transfer of data from the Device to a tool via a Master

Note 1 to entry: The upload feature is not required for the firmware update

3.2 Symbols and abbreviated terms

BLOB	binary large object	
BM	boot mode	
FB	function block	IEC 61131-3
FW	firmware	
GUI	graphical user interface	
IOLFW	IO-Link firmware (file extension)	
ISDU	indexed service data unit	IEC 61131-9
LSO	Least significant octet	
MSO	most significant octet	
OD	on-request data	IEC 61131-9
PC	personal computer	
PCDT	parameterization configuration and diagnosis tool	IEC 61131-9
PLC	programmable logic controller	
SDCI	single-drop digital communication interface	IEC 61131-9

3.3 Conventions

3.3.1 Behavioral descriptions

For the behavioral descriptions, the notations of UML 2 [3] are used, mainly state, activity, and sequence diagrams. The layout of the associated state-transition tables is following IEC 62390 [3].

Triggers are for example external requests ("calls") or internal changes such as timeouts; [guard] are Boolean conditions for exits of states; numbered transitions describe actions in addition to the triggers within separate state-transition tables.

In this document, the concept of "nested states" with superstates and substates is used as shown in the example of Figure 1.

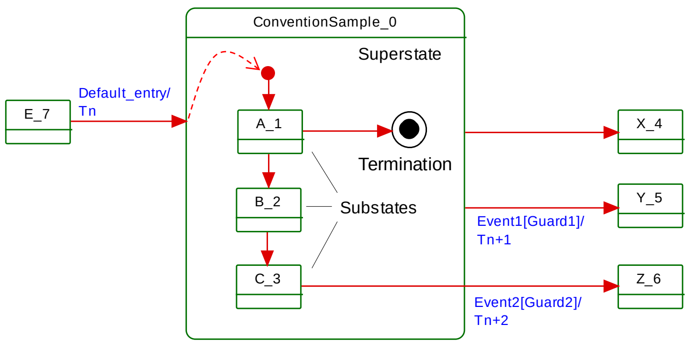


Figure 1 – Example of a nested state

UML 2 allows hierarchies of states with superstates and substates. The highest superstate represents the entire state machine. This concept allows for simplified modelling since the content of superstates can be moved to a separate drawing. An eyeglasses icon usually represents this content. Compared to "flat" state machines, a particular set of rules shall be observed for "nested states":

a) A transition to the edge of a superstate (e.g. Default_entry) implies transition to the initial substate (e.g. A_1).
b) Transition to a termination state inside a superstate implies a transition without event and guard to a state outside (e.g.X_4). The superstate will become inactive.
c) A transition from any of the substates (e.g. A_1, B_2, or C_3) to a state outside (Y_5) can take place whenever event1 occurs and guard1 is true. This is helpful in case of common errors within the substates. The superstate will become inactive.
d) A transition from a particular substate (e.g. C_3) to a state outside (Z_6) can take place whenever event2 occurs and guard2 is true. The superstate will become inactive.

In this document, the concept of "nested states" with regions is used also as shown in the example of Figure 2.

The two "lanes" (regions) can run independently or concurrently. It is possible to exit from a substate in one of the regions, for example from substate B_2, whenever Event1 occurs and guard1 is true. The superstate will become inactive (both regions).

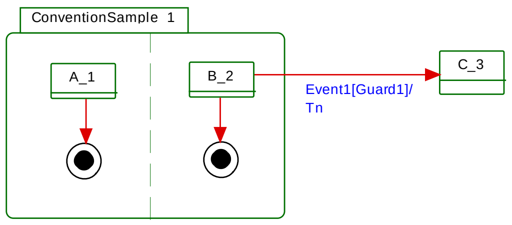


Figure 2 – Superstate with regions (And-state)

The state diagrams shown in this document are entirely abstract descriptions. They do not represent a complete specification for implementation.

3.3.2 Memory and transmission octet order

Figure 3 demonstrates the order that shall be used when transferring WORD based data types from memory to transmission and vice versa (Figure 3).

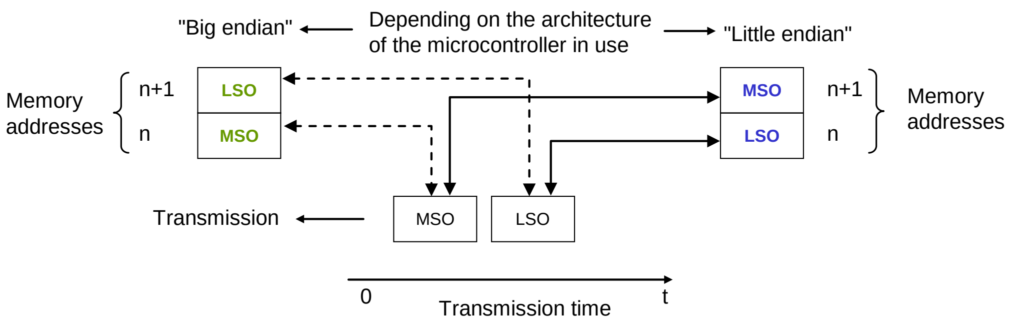


Figure 3 – Memory and transmission octet order

4 Overview of the profile

This profile consists of two parts, "BLOB" and "Firmware-Update". The first part is about the transfer of so-called *binary large objects* such as pictures taken by an optical sensor or large amount of structured data collected by a Device. Due to the limited size of ISDUs (≤ 238 octets) for the transfer of consistent records in IO-Link (see 6.4.1), it is necessary to define a segmented and controlled transfer mechanism on top of the existing ISDU mechanism.

The second part is about *firmware update* (short: FW-Update) of Devices in the field. It deals with unlocking of a Device's firmware; the necessary commands and messages; FW-Update file formats; identification, authentication, and validation issues; DeviceID and version rules; parameter versions and data storage rules.

Figure 4 shows how the additional functions "BLOB transfer" and "Firmware-Update" conform to the basic IO-Link operations (SIO, STARTUP, PREOPERATE, and OPERATE). The profile assumes a tested IO-Link communication layer according to [1], confirmed by a manufacturer declaration.

Usually, a configured Device in the field moves to the OPERATE state after power-on (cyclic Process Data exchange). From there, it is possible via particular Parameters (BLOB_ID and BLOB_CH) to initiate BLOB transfers from and to the Device in an acyclic manner. The entire transmission of a BLOB is secured by 32 bit CRC signature. The profile specifies state machines for the Device, for function blocks (FB according IEC 61131-3 programming languages) in PLCs or for computer software tools (Figure 6).

For FW-Update, a "Firmware valid" check is mandatory prior to a Device start-up until it reaches the OPERATE state.

NOTE Usually, Masters and Devices automatically run through the states in Figure 4 until the OPERATE state is taken. In principle, a FW-Update is also possible from the PREOPERATE state.

From there, additional SystemCommands allow for unlocking of the existing technology firmware within a Device. This new "unlocking" feature as part of the technology firmware is a precondition for the FW-Update of a Device. After unlocking, the Device causes a communication interrupt and after 4 Master cycles, a Bootloader took over responsibility. This one supports the download of the new firmware using the BLOB transfer mechanism.

At the end of the transfer, the new firmware is activated and after another communication interrupt and after 4 Master cycles, the new firmware can take over responsibility.

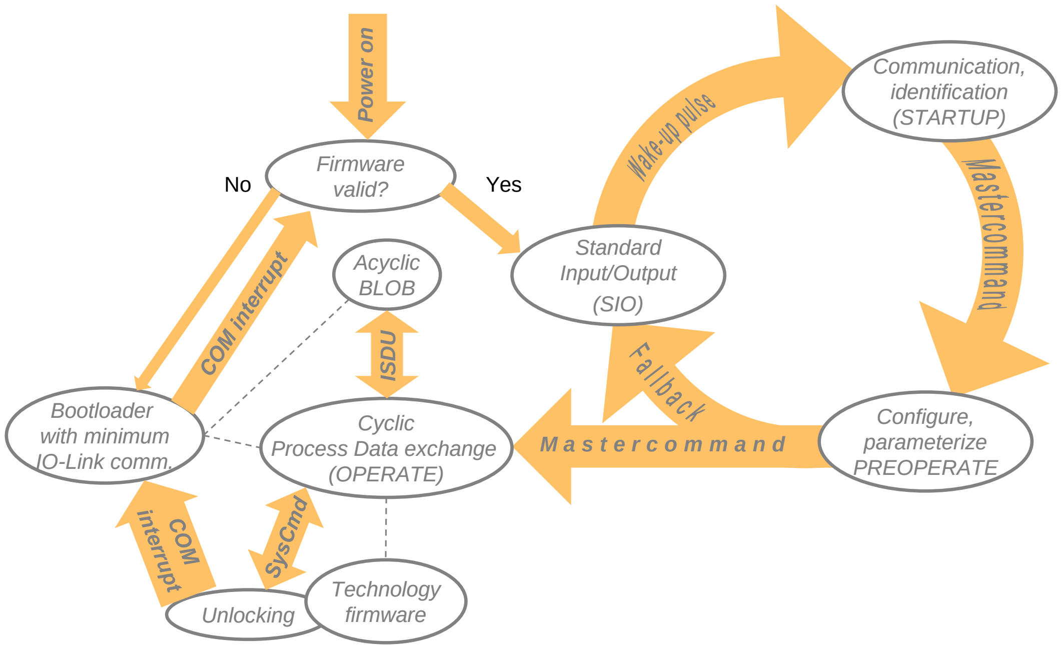

Power on
Firmware valid?
No
Yes
Wake-up pulse
Communication, identification (STARTUP)
Mastercommand
Standard Input/Output (SIO)
Acyclic BLOB
COM interrupt
ISDU
Fallback
Configure, parameterize PREOPERATE
Mastercommand
Bootloader with minimum IO-Link comm.
Cyclic Process Data exchange (OPERATE)
COM interrupt
SysCmd
Unlocking
Technology firmware

Figure 4 – IO-Link operations including BLOB transfer and FW-Update

However, in case the firmware valid test failed, the Bootloader can be re-enabled to support another update procedure.

Figure 5 shows how the additional features "BLOB transfer", "Bootloader" for FW-Update, and the "Unlocking" of the firmware fit into the Device architecture defined in clause 10 of [1].

The content of this profile is structured into the following main clauses. After this overview, clause 5 deals with Binary Large Objects (BLOBs) such as images, totalized measurements, etc. exceeding the size of one ISDU and to be transmitted in segmented manner. Clause 6 deals with firmware updates (FW-Update) of Devices using the BLOB mechanism.

Annex A describes the necessary IODD extensions for both. Annex B specifies the CRC signature calculations and Annex C optional features such as compression and encryption. Annex D provides deap insight in performance issues and Annex E and F define quality measures.

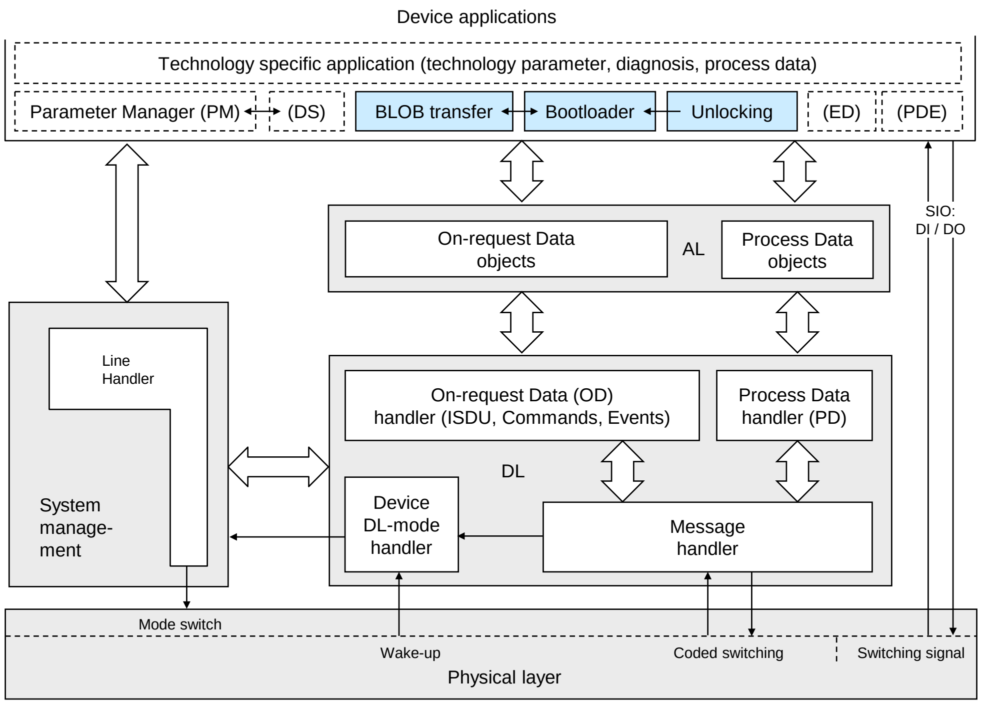


Figure 5 – Device architecture including BLOB and FW-Update

5 Profile characteristic

The IO-Link system provides a parameter "Profile Characteristic" (see [1], Table B.8) indicating the ProfileIdentifiers (PFIDs) a particular Device supports. In case of this BLOB & FW-Update profile the following PFIDs apply:

- BLOB: PFID = 0x0030
- FW-Update: PFID = 0x0031

6 Binary Large Objects (BLOBs)

6.1 Purpose and system positioning

Figure 6 illustrates the transmission of BLOBs within an automation system. "BLOB_Trans_Layers" provide the necessary protocol features between a Device and a host controller such as a PLC or a PC-based tool running for example manufacturing supervisory software.

Usually, the D-BLOB_Trans_Layer is implemented with the help of the programming language for the Device's firmware, whereas the P-BLOB_Trans_Layer is implemented in a function block (FB) with the help of an IEC 61131-3 programming language, for example "Structured Text" (ST). In the ideal case, the PLC manufacturers are supporting a standardized FB according to this document in their FB library of the engineering tool.

The T-BLOB_Trans_Layer is implemented using appropriate programming languages of that system environment, the software tool is launched and running in.

The acyclic On-request Data mechanism "ISDU" is engaged in the transmission of BLOBs (see 6.4).

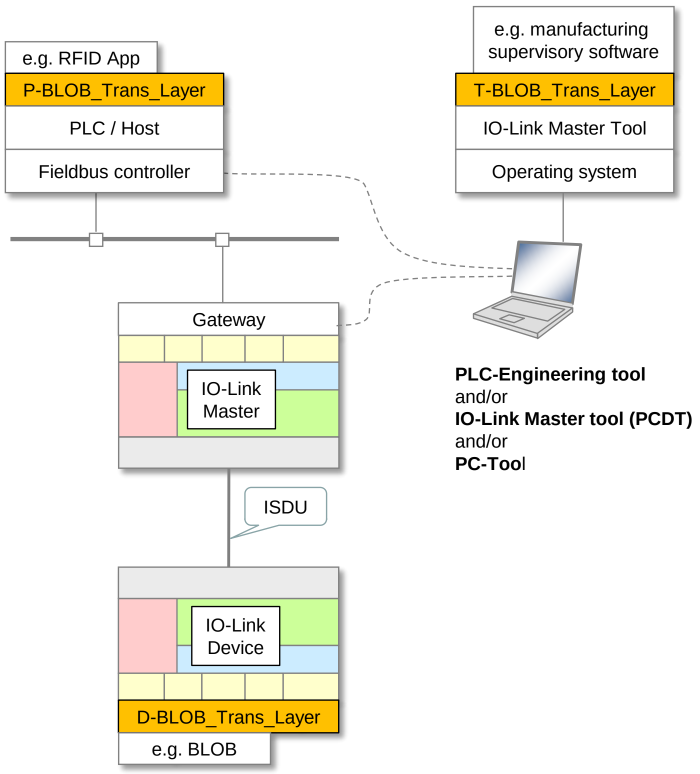


Figure 6 – BLOB transmission system

6.2 Components involved

6.2.1 PLC / Host

BLOB transfer is based on the standard IO-link BLOB protocol using ISDU communication between Master and Device. It can be implemented in a PLC user program as a P-BLOB_Trans_Layer, usually in form of a function block (FB). PLC vendors often provide common function blocks for read/write records adapted to the ISDU mechanism. This facilitates access from the PLC across the fieldbus via the Master to the Device and vice versa. Figure 6 illustrates the mechanism. The BLOB function block can be used for example to read or write RFID tags via an RFID user program within the PLC.

6.2.2 Tools

6.2.2.1 PLC-Engineering

This tool supports configuration of the IO-Link Master for fieldbus operation (network access, I/O data exchange, port configuration, etc.). It also allows development of user programs including function blocks (e.g. BLOB FB) based on IEC 61131-3 programming languages.

6.2.2.2 PC-Tools

Computer coming with communication On-request Data access to Masters (AL layer) can be used for the implementation of the BLOB transfer mechanism (T-BLOB_Trans_Layer). This enables the user to directly acquire information from a particular associated technology application within a Device or to transfer BLOBs into a Device (for example through manufacturing supervisory systems).

Such BLOBs could be FW-Update data objects. Corresponding PC-Tool software can provide user dialogs in addition to the BLOB transfer mechanism

6.2.2.3 Master tools (PCDT)

Most of the Masters are coming with a Master tool (PCDT) that provides already On-Request Data access. Thus, it is easy to extend this tool by a T-BLOB_Trans_Layer and an additional BLOB application such as FW-Update.

6.2.3 Master

All Masters according IO-link V1.1 (or a later version) can handle the BLOB transfer mechanism. No modification is required.

6.2.4 Device

The BLOB transfer mechanism supports all Devices requiring a data channel for data objects larger than the ISDU size, e.g. cameras, 3D-scanner, and measurement sensors (totalizer, data recorder). The data objects can consist of parameters, process data, diagnosis information, firmware updates, etc.

6.3 Data objects within BLOBs

6.3.1 Types of objects

BLOBs can contain various data objects. Examples are images (JPG, PNG, or other formats), text (ASCII, XML, CSV, or other formats), binaries (HEX, BIN, or other formats), etc.

This document specifies only the data object "FW-Update". It is a binary data object with unknown content (see 7.2.4).

6.3.2 Securing measures

A 32 bit CRC signature is used to ensure data integrity of the transmitted BLOB content (see Annex B).

Additional proprietary measures for compression, authentication, and encryption can be incorporated in the content also (see Annex C).

6.4 ISDU as transport vehicle

6.4.1 General

BLOBs are transferred by means of the ISDU mechanism specified in [1]. No modifications are required. The size of a BLOB can be up to 231 octets to fit into one ISDU. Larger BLOBs shall be segmented and transferred ISDU by ISDU. Figure 7 shows example ISDUs usually used for BLOB transfer (see [1], Annex A.5.7).

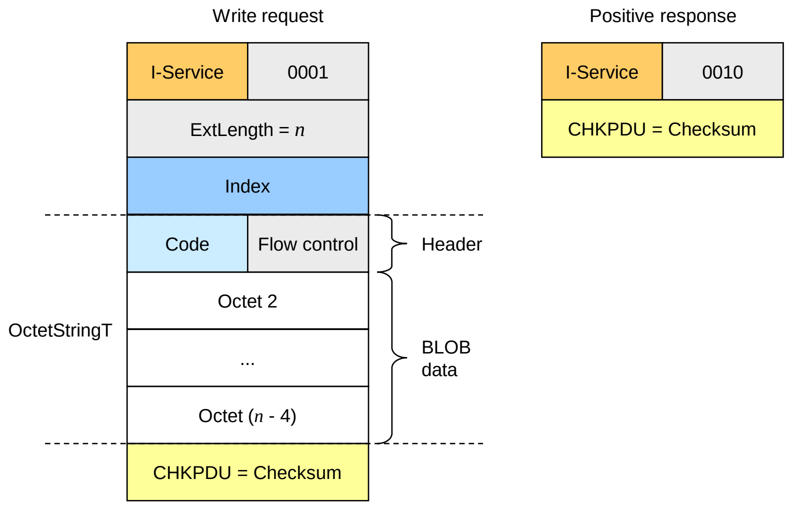


Figure 7 – Example structure of an ISDU for BLOB transfer (BLOB_CH)

Annex D provides information on the BLOB transfer performance with respect to transmission rates and BLOB segment sizes.

Since BLOB transfers can last for some time due to their extended length, SDCI communication can always be interrupted after a valid ISDU transmission of a write request. Any Tool shall be able to manage those communication interrupts.

6.4.2 Diagnosis – EventCodes

There are no profile-specific EventCodes required.

6.4.3 Acknowledgments

The CRC signature serves as acknowledgment for the transfer of the BLOB data. The ISDU mechanism provides already sufficient acknowledgments for the transfer of segments (see [1], A.5).

6.5 BLOB parameters and transfer

6.5.1 Profile related Index space

The parameters required for the BLOB transmission are specified in Table 1 using Indices reserved for profiles (see Table B.8 in [1]). Support of these parameters is mandatory for Devices providing BLOB transfer ("conditional").

Table 1 – Index assignment of the BLOB parameters

Index (dec)	Object name	Access	Length	Data type	M/O/C	Remark
…						
0x0031 (49)	BLOB_ID	R	2 octets	IntegerT	C	See 6.5.2 and Table B.8 in [1]
0x0032 (50)	BLOB_CH	R/W	variable	OctetStringT	C	See 6.5.3 and Table B.8 in [1]
…						
Key M = mandatory; O = optional; C = conditional						

Details of parameter "BLOB_ID" are specified in 6.5.2 and details of parameter "BLOB_CH" in 6.5.3. BLOB parameters are transferred using ISDU Read or Write requests.

6.5.2 BLOB_ID

The parameter BLOB_ID is used to indicate the current BLOB whose transmission is in progress. Its data type is IntegerT (16) and read only. The Device provides its available BLOB_IDs via its IODD (see A.2) and the associated parameter value within this Index.

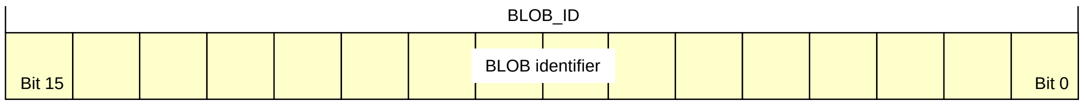


Figure 8 – Structure of BLOB_ID

Table 2 shows the coding of BLOB_ID, which implicitely indicates the read or write transmission direction.

Table 2 – Coding of BLOB_ID

Value (dec)	Definition
-32768	Not permitted
-32767 to -8193	Reserved
-8192 to -4096	Manufacturer specific (read)
-4095 to -1	Profile specific (read)
0	Idle transmission of a BLOB
1 to 4095	Profile specific (write)
4096 to 8191	Manufacturer specific (write)
8192 to 32767	Reserved

6.5.3 BLOB_CH

The parameter BLOB_CH defines the transmission channel for a particular BLOB through a particular BLOB_ID within the command BLOB_Start. It has a variable structure and always starts with an 8 bit header and a variable length of BLOB (body) data.

Figure 9 shows the structure of the header of BLOB_CH.

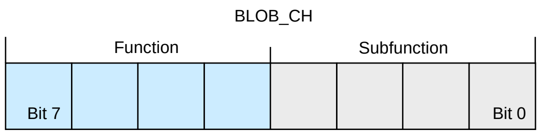


Figure 9 – Header of BLOB_CH

Table 3 shows the coding of the header of BLOB_CH. A Read request of this parameter has no associated data to the header. Therefore, the reponse of the Device to a Read request depends on the respective state of the BLOB state machine. A Read request also triggers internal state changes and thus cannot be performed twice.

Table 3 – Coding of the header of BLOB_CH

Transmission items	Read/ Write	Function	Subfunction	Definition/Parameter
–	–	0x0	0x0 to 0xF	Reserved
BLOB_Info_Read	R	0x1 (Read Info block)	0x0	Parameter: See Figure 10 (Information for the Read channel)
BLOB_Info_Write	R		0x1	Parameter: See Figure 11 (Information for the Write channel)
BLOB_Segment	R/W	0x2	0x0 to 0xF (flow control)	Counting of segments modulo 16. It starts at 0 and rolls over after 15 to 0. Parameter: Segment. NOTE
BLOB_Last	R/W	0x3	0x0	Parameter: Last segment of the BLOB.
BLOB_CRC	R/W	0x4	0x0	Parameter: CRC signature across the BLOB.
–	–	0x5 to 0xE	0x0 to 0xF	Reserved
BLOB_Abort	W	0xF (commands)	0x0	Command to abort the active transmission channel. No parameter.
BLOB_Start	W		0x1	Command to select the BLOB_ID and to establish the transmission channel. Parameter: BLOB_ID
BLOB_Finish	W		0x2	Command to finish the active transmission channel. No parameter.
-		0xF	0x3 to 0xF	Reserved for commands
NOTE Mechanism similar to [1], Table 50 – FlowCTRL definitions, COUNT				

6.5.3.1 BLOB_Info_Read

Prior to reading the parameter "BLOB_Info_Read" block, the required BLOB shall be assigned via "BLOB_Start" (see 6.5.3.8). Without an assigned BLOB, a negative Read response will be returned via an ISDU error: "0x8020"– *Service temporarily not available* (see [1], Table C.1).

A Read request to the BLOB_CH Index with the corresponding header (Function and Subfunction, see Table 3) will return the "read" properties of the active BLOB_ID within an information block as shown in Figure 10.

The "BLOB_Info_Read" block contains the BLOB length in octets, coded in UIntegerT (32).

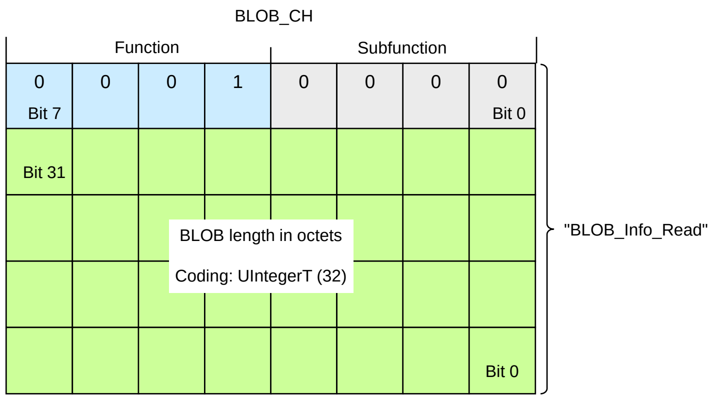


Figure 10 – Structure of "BLOB_Info_Read" block

6.5.3.2 BLOB_Info_Write

Prior to reading the parameter "BLOB_Info_Write" block, the required BLOB (BLOB_ID) shall be assigned via "BLOB_Start" (see 6.5.3.8). Without an assigned BLOB, a negative read response will be returned via an ISDU error "0x8020"– *Service temporarily not available*.

A Read request to the BLOB_CH Index with the corresponding header (Function and Subfunction, see Table 3) will return the "write" properties of the active BLOB_ID within an information block as shown in Figure 11.

The "BLOB_Info_Write" block contains the maximum BLOB size in octets, coded in UIntegerT (32) and the maximum ISDU size of a particular Device in octets, coded in UIntegerT (8).

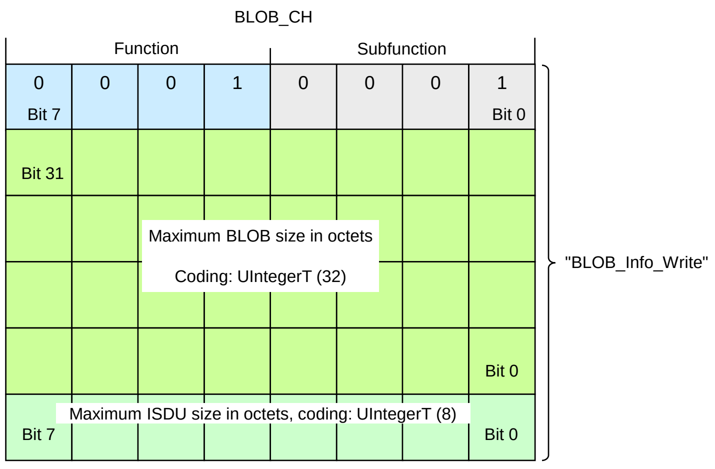


Figure 11 – Structure of "BLOB_Info_Write" block

6.5.3.3 Write BLOB_Segment

Whenever a BLOB is larger than the size of an ISDU, it will be transferred in segments, in each case filling the entire ISDU data size. The item "BLOB_Segment" indicates a particular part of the BLOB via a flow control number ("BFlowCtrl") of the segment.

A Write request to the BLOB_CH Index with the corresponding header (Function and Subfunction, see Table 3) and its body is shown in Figure 12. Data shall be transmitted as a big endian sequence, i.e. the most significant octet (MSO) shall be transmitted first, followed by less significant octets in descending order, with the least significant octet (LSB) being sent last according to [1].

The Device will return a positive Write request response.

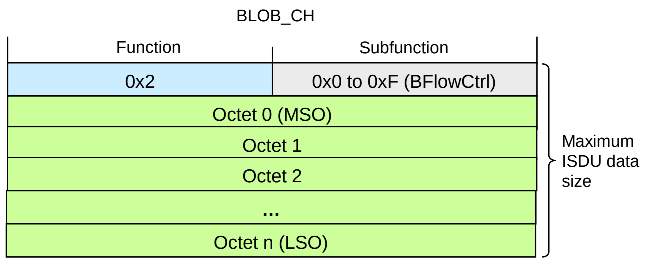


Figure 12 – Structure of "BLOB_Segment"

6.5.3.4 Read BLOB_Segment

A Read request to the BLOB_CH Index with the corresponding header (Function and Subfunction, see Table 3) will return a body as shown in Figure 12. Data shall be transmitted as a big endian sequence, i.e. the most significant octet (MSO) shall be transmitted first, followed by less significant octets in descending order, with the least significant octet (LSB) being sent last according to [1].

6.5.3.5 BLOB_Last

The item "BLOB_Last" indicates the last segment to be transferred. This segment contains the remainder of the BLOB. Padding octets shall fill up the space up to the maximum ISDU data size as shown in Figure 13.

Write and Read requests correspond to 6.5.3.3 and 6.5.3.4.

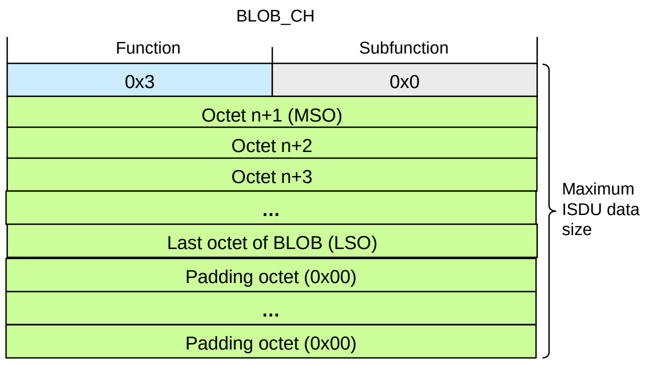


Figure 13 – Structure of "BLOB_Last"

6.5.3.6 BLOB_CRC

The item "BLOB_CRC" transmits the 32 bit CRC signature used to secure the entire BLOB data.

A Write request to the BLOB_CH Index with the corresponding header (Function and Subfunction, see Table 3) and its body is shown in Figure 14 .

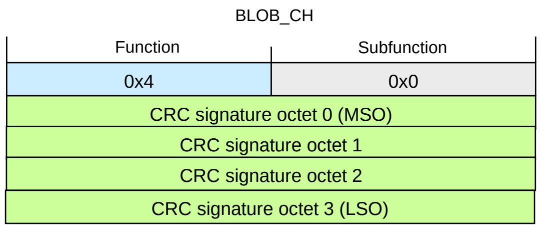


Figure 14 – Structure of "BLOB_CRC"

Writing a CRC signature will lead to a positive acknowledgment in case of correct transmission or to an ISDU error "0x8040"– *Invalid parameter set* (see [1], Table C.1).

Annex B provides information on how to calculate the BLOB_CRC signature.

A Read request to the BLOB_CH Index with the corresponding header (Function and Subfunction, see Table 3) will return a body as shown in Figure 14.

6.5.3.7 BLOB_Abort

The item "BLOB_Abort" represents a command to abort an ongoing BLOB transmission. The Write request is shown in Figure 15.

BLOB_CH

Function	Subfunction
0xF	0x0

Figure 15 – Structure of command "BLOB_Abort"

The Write request will lead to a positive acknowledgment.

6.5.3.8 BLOB_Start

The item "BLOB_Start" represents a command to launch the transmission of that BLOB whose BLOB_ID is contained in the parameter. The Write request is shown in Figure 16.

BLOB_CH

Function	Subfunction
0xF	0x1
BLOB_ID octet 0 (MSO)	
BLOB_ID octet 1 (LSO)	

Figure 16 – Structure of command "BLOB_Start"

The Write request will lead to

– a positive acknowledgment in case of an accepted command, or

– an ISDU error "0x8030"– *Parameter value out of range* in case the BLOB_ID is not supported, or

– an ISDU error "0x8036"– *Function temporarily not available* in case the transfer for this BLOB_CH is already active.

6.5.3.9 BLOB_Finish

The item "BLOB_Finish" represents a command to finalize a successful BLOB transmission. The Write request is shown in Figure 17.

BLOB_CH

Function	Subfunction
0xF	0x2

Figure 17 – Structure of command "BLOB_Finish"

The Write request will lead to a positive acknowledgment in case of an accepted command or to an ISDU error "0x8036"– *Function temporarily not available* in case the state machine is not in state "WaitOn_BLOB_complete_6 (see Figure 18).

6.5.3.10 Concurrent ISDU transfers

If any concurrent ISDU transfer besides the BLOB transfer leads to a conflict, ISDU error "0x8020"– *Service temporarily not available* shall be returned.

6.6 Protocol of BLOB transmission

6.6.1 Device BLOB state machine

Figure 18 shows the state machine of the "D-BLOB_Trans_Layer" (see 6.1). It is driven by Read or Write requests from the Host state machine (see Figure 19). However, within each state a Read_BLOB_ID will always be responded without quitting the state.

According to the conventions in 3.3.1, the nested states 1 to 6 in superstate 7 are enabled to react on errors (T18) within all of these states or on an incoming service "BLOB_Abort" (T17).

Any incorrect Read or Write request within superstate 7 will lead to the state IDLE_0 in order to synchronize with the Host state machine in Figure 19.

It is highly recommended for the Device communication in case of FW-Update operation to care for robust parameters such as a relaxed Min_Cycle_Time to prevent from communication disruptions.

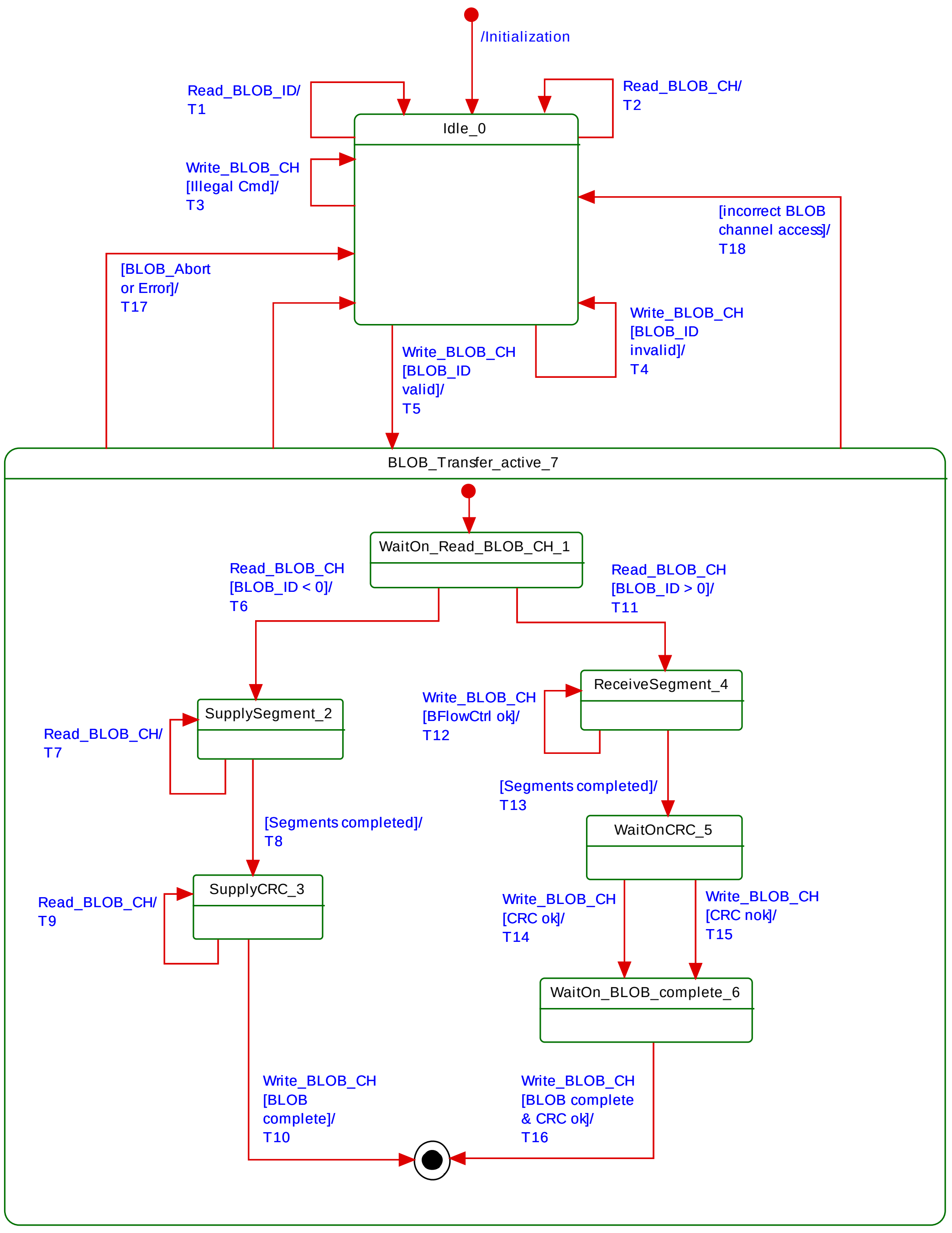

/Initialization
Read_BLOB_ID/ T1
Read_BLOB_CH/ T2
Idle_0
Write_BLOB_CH [Illegal Cmd]/ T3
[incorrect BLOB channel access]/ T18
[BLOB_Abort or Error]/ T17
Write_BLOB_CH [BLOB_ID invalid]/ T4
Write_BLOB_CH [BLOB_ID valid]/ T5
BLOB_Transfer_active_7
WaitOn_Read_BLOB_CH_1
Read_BLOB_CH [BLOB_ID < 0]/ T6
Read_BLOB_CH [BLOB_ID > 0]/ T11
ReceiveSegment_4
Write_BLOB_CH [BFlowCtrl ok]/ T12
SupplySegment_2
Read_BLOB_CH/ T7
[Segments completed]/ T13
[Segments completed]/ T8
WaitOnCRC_5
SupplyCRC_3
Read_BLOB_CH/ T9
Write_BLOB_CH [CRC ok]/ T14
Write_BLOB_CH [CRC nok]/ T15
WaitOn_BLOB_complete_6
Write_BLOB_CH [BLOB complete]/ T10
Write_BLOB_CH [BLOB complete & CRC ok]/ T16

Figure 18 – Device BLOB state machine

Table 4 shows the state transition tables of the Device BLOB state machine.

505 **Table 4 – State transition tables of the Device BLOB state machine**

STATE NAME	STATE DESCRIPTION
Idle_0	No BLOB transmission active.
WaitOn_Read_BLOB_ CH _1	Prepare BLOB info for requested BLOB_ID and direction.
SupplySegment_2	Prepare segments in ascending order upon every read request, generate flow control
SupplyCRC_3	Prepare CRC signature across transmitted BLOB content.
ReceiveSegment_4	Receive segments in ascending order upon every write request, check flow control.
WaitOnCRC_5	Receive target CRC signature across transmitted BLOB content. Compare with internally calculated CRC.
WaitOn_BLOB_complete _6	Wait on BLOB finalization via BLOB_Finish command.
BLOB_transfer_active_7	This superstate allows all states inside to react on - command "BLOB_Abort", - reading BLOB_ID to provide actual BLOB_ID, - command BLOB_Start to provide ISDU error 0x8022.

506

TRANSITION	SOURCE STATE	TARGET STATE	ACTION
T1	0	0	Response: BLOB_ID = 0.
T2	0	0	Response: ISDU error 0x8020 "Service temporarily not available"
T3	0	0	Response: ISDU error 0x8030 "Parameter value out of range"
T4	0	0	Response: ISDU error 0x8030 "Parameter value out of range"-
T5	0	1	Response: ISDU acknowledgment
T6	1	2	Response BLOB_CH: "BLOB_Info_Read"
T7	2	2	Response BLOB_CH: "BLOB_Segment" content
T8	2	3	–
T9	3	3	Response BLOB_CH: "BLOB_CRC"
T10	3	0	Set parameter BLOB-ID to IDLE
T11	1	4	Response BLOB_CH: "BLOB_Info_Write"
T12	4	4	Store BLOB segment, response: ISDU acknowledgment
T13	4	5	Calculate CRC signature across received BLOB content
T14	5	6	Response ISDU acknowlegde "No Error"
T15	5	6	Response ISDU error 0x8040 "Invalid parameter set"
T16	6	0	Set parameter BLOB-ID to IDLE
T17	1,2,3,4,5, 6	0	Set parameter BLOB-ID to IDLE; garbage collection
T18	1,2,3,4,5, 6	0	Set parameter BLOB-ID to IDLE; garbage collection. Return ISDU error 0x8030 "Parameter value out of range".

507

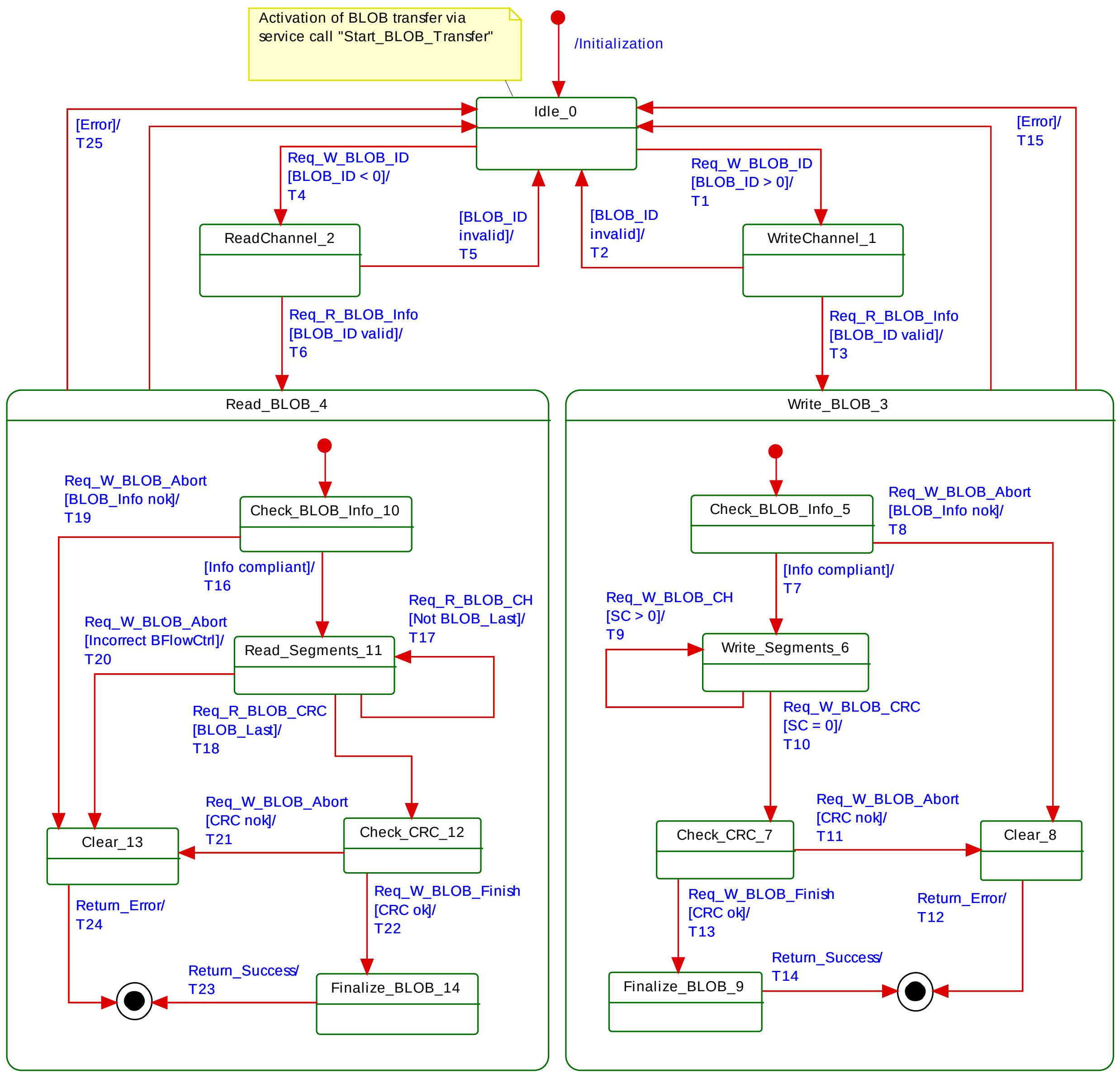
INTERNAL ITEMS	TYPE	DEFINITION
BLOB_ID	Index	See 6.5.2
BLOB_CH	Index	See 6.5.3
BLOB _Abort	Service	Item of Index BLOB_CH: See 6.5.3
BLOB _Finish	Service	Item of Index BLOB_CH: See 6.5.3
CRC	Variable	CRC signature across BLOB data
BLOB_Info_Read	Variable	Item of Index BLOB_CH: "Information", see 6.5.3
BLOB_Info_Write	Variable	Item of Index BLOB_CH: "Information", see 6.5.3
Illegal Cmd	Guard	Any access to BLOB-CH with an invalid header or data content

INTERNAL ITEMS	TYPE	DEFINITION
Incorrect BLOB channel access	Guard	Any invalid read or write access on BLOB-CH when this direction or content is not allowed
Error	Guard	Any negative ISDU response which is not handled within the super-state
BFlowCtrl	Variable	Flow control of BLOB transmission according to Table 3

6.6.2 Host BLOB state machine

Figure 19 shows the state diagram of the "P-BLOB_Trans_Layer" or "T-BLOB_Trans_Layer (see 6.1). States 5 to 9 are nested in state 4 allowing these states to react on errors (T18) within all of these states. States 10 to 14 are nested in state 3 allowing these states to react on errors (T25) within all of these states.

NOTE Req_W_BLOB_ID is a "CallTrigger" (in UML) and means: Host requests the Master to send an ISDU with "Write_BLOB_ID"



Activation of BLOB transfer via service call "Start_BLOB_Transfer"
/Initialization
Idle_0
[Error]/ T25
[Error]/ T15
Req_W_BLOB_ID [BLOB_ID < 0]/ T4
Req_W_BLOB_ID [BLOB_ID > 0]/ T1
ReadChannel_2
WriteChannel_1
[BLOB_ID invalid]/ T5
[BLOB_ID invalid]/ T2
Req_R_BLOB_Info [BLOB_ID valid]/ T6
Req_R_BLOB_Info [BLOB_ID valid]/ T3
Read_BLOB_4
Write_BLOB_3
Req_W_BLOB_Abort [BLOB_Info nok]/ T19
Check_BLOB_Info_10
[Info compliant]/ T16
Req_W_BLOB_Abort [Incorrect BFlowCtrl]/ T20
Req_R_BLOB_CH [Not BLOB_Last]/ T17
Read_Segments_11
Req_R_BLOB_CRC [BLOB_Last]/ T18
Req_W_BLOB_Abort [CRC nok]/ T21
Clear_13
Check_CRC_12
Req_W_BLOB_Finish [CRC ok]/ T22
Return_Error/ T24
Return_Success/ T23
Finalize_BLOB_14
Check_BLOB_Info_5
Req_W_BLOB_Abort [BLOB_Info nok]/ T8
[Info compliant]/ T7
Req_W_BLOB_CH [SC > 0]/ T9
Write_Segments_6
Req_W_BLOB_CRC [SC = 0]/ T10
Req_W_BLOB_Abort [CRC nok]/ T11
Check_CRC_7
Clear_8
Req_W_BLOB_Finish [CRC ok]/ T13
Return_Error/ T12
Return_Success/ T14
Finalize_BLOB_9

Figure 19 – Host BLOB state machine

Table 5 shows the state transition tables of the Host BLOB state machine.

519 **Table 5 – State transition tables of the Host BLOB state machine**

STATE NAME	STATE DESCRIPTION
Idle_0	No BLOB transmission active. Wait on activation through service call "Start_BLOB_Transfer" with arguments BLOB_ID and a pointer to the binaries of the BLOB.
WriteChannel_1	Await Device ISDU response: "BLOB_ID" OK?
ReadChannel_2	Await Device ISDU response: "BLOB_ID" OK?
Write_BLOB_3	Complete BLOB transfer to the Device. This superstate allows all states inside to react on any error and to quit.
Read_BLOB_4	Complete BLOB transfer from the Device. This superstate allows all states inside to react on any error and to quit.
Check_BLOB_Info_5	Wait on Device response: "BLOB_Info" OK? Calculate number of segments depending on BLOB size, max ISDU size and set SegmentCounter value.
Write_Segments_6	Demand Master to write BLOB segment by segment (ISDU by ISDU); generate/update BFlowCtrl. Wait on Device response: ISDU OK? Decrement SegmentCounter at each successful segment transfer. In case of ComLost the last ISDU request shall be retried at least 2 times.
Check_CRC_7	Wait on Device response: CRC OK? In case of ComLost the last ISDU request shall be retried at least 2 times.
Clear_8	Garbage collection. Prepare error message.
Finalize_BLOB_9	Wait on Device response.
Check_BLOB_Info_10	Wait on Device response: "BLOB_Info" OK?
Read_Segments_11	Demand Master to read BLOB segment by segment (ISDU by ISDU); check BFlowCtrl. Wait on Device response.
Check_CRC_12	Compare internally calculated CRC with received CRC
Clear_13	Garbage collection. Prepare error message.
Finalize_BLOB_14	Wait on Device response.

520

TRANSITION	SOURCE STATE	TARGET STATE	ACTION
Initialization	-	0	–
T1	0	1	–
T2	1	0	–
T3	1	4	–
T4	0	2	–
T5	2	0	–
T6	2	3	–
T7	5	6	–
T8	5	8	–
T9	6	6	–
T10	6	7	–
T11	7	8	–
T12	8	0	–
T13	7	9	–
T14	9	0	–
T15	5,6,7,8,9	0	Garbage collection
T16	10	11	–
T17	11	11	–
T18	11	12	–

TRANSITION	SOURCE STATE	TARGET STATE	ACTION
T19	10	13	–
T20	11	13	–
T21	12	13	–
T22	12	14	–
T23	14	0	–
T24	13	0	–
T25	10,11,12, 13,14	0	Garbage collection

INTERNAL ITEMS	TYPE	DEFINITION
BLOB_ID	Index	See 6.5.2
BLOB_CH	Index	See 6.5.3
BLOB_Abort	Service	Item of Index BLOB_CH: See 6.5.3
SegmentCounter (SC)	Variable	Preset with number of required segments and decrement to 0.
BLOB_CRC	Variable	Item of Index BLOB_CH: CRC signature across BLOB data
BLOB_Info_read	Variable	See 6.5.3
BLOB_Info_write	Variable	See 6.5.3
BLOB_Last	Variable	Last BLOB segment transmitted
Incorrect flow	Boolean	BFlowCtrl violated
ComLost	Variable	Communication failed
RetryCounter	Variable	Preset = 2
Retry	Guard	Retry = true, when ComLost and RetryCounter <>0

6.7 Access conflicts

6.7.1 Overview

The host application shall be designed in such a manner that access conflicts are avoided. The Device cannot distinguish between two different BLOB transfers.

6.7.2 Concurrent access of tools

Any Host shall delay/reject a transmission, when

- "Read BLOB_ID" does not occur in state "Idle_0",
- a "BLOB_Start" results in Error 0x8022 – *Service temporarily not available* or 0x8082 – *Channel busy.*

6.7.3 Access blocking

An ongoing BLOB transfer shall only be interrupted by the PLC or Host upon deliberate user intervention. This can occur whenever a previously interrupted BLOB transfer is restarted or a second Host starts an independent BLOB access.

The user's decision depends heavily on the BLOB currently under transmission and the circumstances.

7 Firmware-Updates

7.1 Purpose and system positioning

Figure 20 illustrates the transmission of firmware updates within an automation system with IO-Link. The "FW-Update" application and the "Bootloader" using "BLOB_Trans_Layers" provide the necessary protocol features between a Device and a computer software tool.

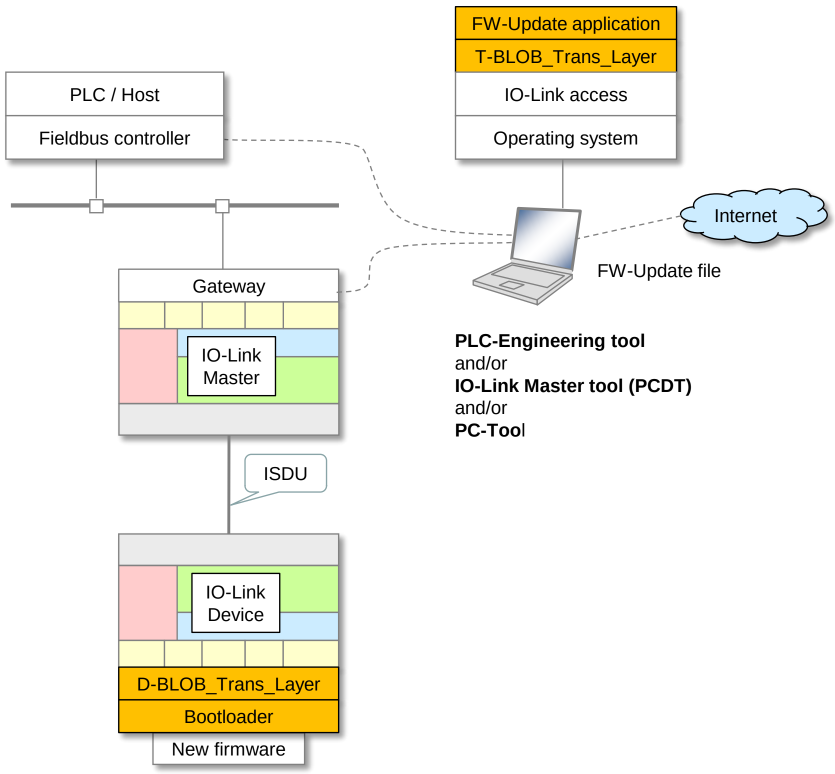


Figure 20 – FW-Update transmission system

7.2 Components involved

7.2.1 Master (PCDT) or PC tools

Either Master (PCDT) or individual PC tools carrying the BLOB transfer mechanism (see 6.2.2.2 or 6.2.2.3) can provide a FW-Update application software with particular user dialogs such as Device access and identification, FW-Update file acquisition, verification of compatibility, authorization, download, and finalization.

7.2.2 Master

All Masters according IO-link V1.1 or later can handle the FW-Update/BLOB transfer mechanism. No modification is required.

7.2.3 Device

As a precondition, the Device shall support the FW-Update profile. The technology firmware of such a modified Device is extended by a firmware check state and a bootloader. The bootloader is responsible for the BLOB transfer, the verification, the optional decryption and the flashing of the new technology firmware. In case of a FW-Update error (invalid technology firmware) the active bootloader remains enabled and a retry is possible.

7.2.4 FW-Update file

The Device manufacturer is responsible for the provision of the FW-Update file. It contains metadata (e. g. for identification and verification) and the binary data of a valid firmware. The binary data is manufacturer specific and can be encrypted or packed.

7.3 Use cases

7.3.1 Update Firmware

Figure 21 shows the use cases for the FW-Update procedure.

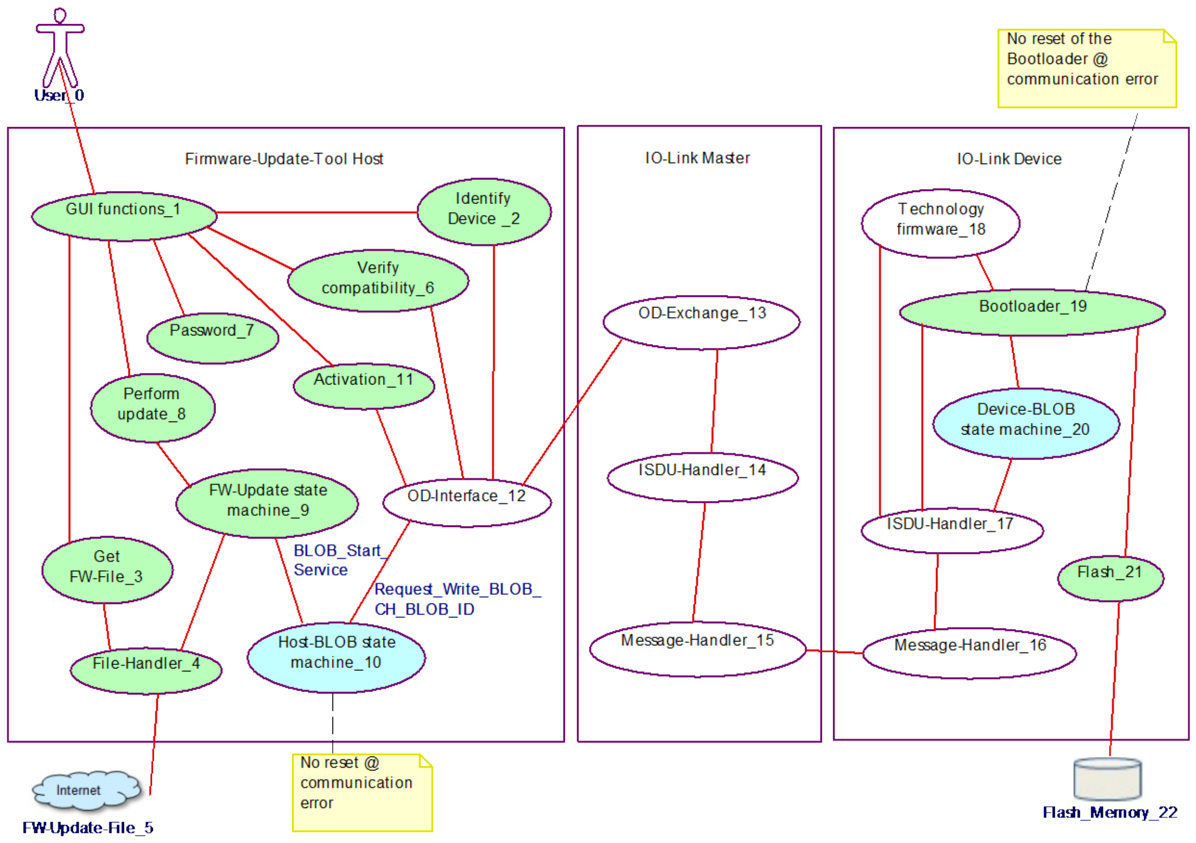


Figure 21 – Use cases of the FW-Update procedure

Table 6 shows a listing of the items in Figure 21 and references to clauses within this document or to other IO-Link specifications (bibliography).

Table 6 – Use Case reference table

No.	Item	Type	Reference	Remarks
0	User	Role description	–	Responsibility of the software tool manufacturer
1	GUI functions	Services	Clause 7.9.1	
2	Identify Device	Activity	Clause 7.5.2	
3	Get FW-File	Activity	Clause 7.5.3	
4	File-Handler	Activity	Clause 7.5.3	
5	FW-Update-File	Meta data	Clause 7.4.4	
6	Verify compatibility	Activity	Clause 7.5.4	
7	Password	Activity	Clause 7.5.5	
8	Perform update	Activity	Clause 7.5.6	
9	FW-Update state machine	State machine	7.7.2 and 7.7.4	
10	Host-BLOB state machine	State machine	Clause 6.6.2	
11	Activation	Activity	Clause 7.5.7	
12	OD-Interface	Comm-Layer	[1]	Proprietary

No.	Item	Type	Reference	Remarks
13	OD-Exchange	Gateway application	[1]	IO-Link standard
14	ISDU-Handler	Master DL	[1]	IO-Link standard
15	Message-Handler	Master DL	[1]	IO-Link standard
16	Message-Handler	Device DL	[1]	IO-Link standard
17	ISDU-Handler	Device DL	[1]	IO-Link standard
18	Technology firmware	Device application	–	Proprietary
19	Bootloader	Activity	Clause 0	
20	Device-BLOB state machine	State machine	Clause 6.6.1	
21	Flash	Activity	Clause 7.7.2	
14	Flash memory	Requirements	–	Proprietary

7.3.2 Upgrade

In this case, the firmware within a Device will be replaced by a newer version providing enhanced functionality or bug fixes. The vendor of the FW-Update file shall ensure that the new firmware is compatible and complies with all required standards, for example [1]. The upgrade can comprise the entire firmware or only parts of it.

Table 7 lists use cases of possible changes or upgrades and the corresponding implications to be considered by the designer/manufacturer.

Table 7 – Use cases of possible changes/upgrades

Use case/Change	PLC	USB-Master	Production	New DeviceID	Impact
Technology FW (bug fix)	Yes	Yes	No	No	–
Technology FW (functions)	Yes? DeviceID?	Yes	No	Yes	- New IODD - Reset DS
IO-Link stack	Yes	Yes	No	No	–
Bootloader/FW-Update	No	Yes	Yes	No	–
Parameter structure	Not recommended	Not recommended	Yes	Yes	–
New profile/version	Not recommended	Yes	No	Yes	- New IODD - Reset DS
DeviceID	–	Yes	No	Yes	- New IODD - Port config.
VendorID, VendorName	–	Yes	No	No	- New IODD
ProductID, ProductName	–	Yes	No	Yes	- New IODD
Data Storage & parameter handling NOTE	Application specific	Application specific	Application specific	Application specific	Application specific
NOTE Manufacturer should provide further information within user manual or within the "InfoMessage".					

7.3.3 Downgrade

In one case, the new firmware within the Device may lead to an incompatibility with external automation components such as a Master or function blocks (FB) within a PLC.

In another case, a user requires a previous proven-in-use or qualified firmware version within his machine application even in spare part Devices with a newer firmware.

In both cases, a rollback to the previous version is possible and the manufacturer/vendor can provide the previous version as a second FW-Update file. The downgrade shall restore all firmware parts modified by the upgrade. User parameters are available through the Data Storage mechanism.

Some microcontrollers allow flashing of a second firmware version and simple switching between both. It is the responsibility of the manufacturer to provide the switching mechanism.

7.3.4 Upload firmware

Upload of a flashed firmware is not supported.

7.4 File formats

7.4.1 General

The FW-Update file contains information specific to the FW-Update mechanism of Devices via IO-Link communication. The file internally represents a zip archive. The archive is a package that shall consist of a metadata file, binary data file with the firmware BLOB and optionally additional resource files. The metadata file describes the FW-Update data and the internal file structure of the package. All these files shall be included in a zip archive without embedded folders. The archive extension shall be "iolfw".

7.4.2 Creation of file

The FW-Update file can be created by the use of third party tools. Furthermore, it is possible to generate the FW-Update file by manually adding the metadata file, the binary data file (BLOB), and optionally additional resource files to a zip archive and renaming it according to the file naming convention (see 7.4.3). The structure and contents of the meta data file is described in 7.4.4.

7.4.3 File naming convention

The file name shall be chosen according to the naming convention shown in Figure 22:

<*Vendor name*>-<*Firmware descriptor*>-<*Date of creation*>-IOLFW<*Schema version*>.iolfw

Figure 22 – FW-Update file naming convention

The terms used in Figure 22 are specified in Table 8.

Table 8 – Items of the file naming convention

Item	Maximum length	Format	Definition
Vendor name		UTF8	Name of FW-Update file vendor, usually the Device vendor
Firmware descriptor		UTF8	Vendor specific descriptor of the file
Date of creation		YYYYMMDD	This date shall correspond to the "releaseDate" attribute in the DocumentInfo element of the metadata file
Schema version		[0-9].[0-9]	This version shall comply with the XML standard, for example "1.0".

Figure 23 shows an example of a FW-Updatefile name.

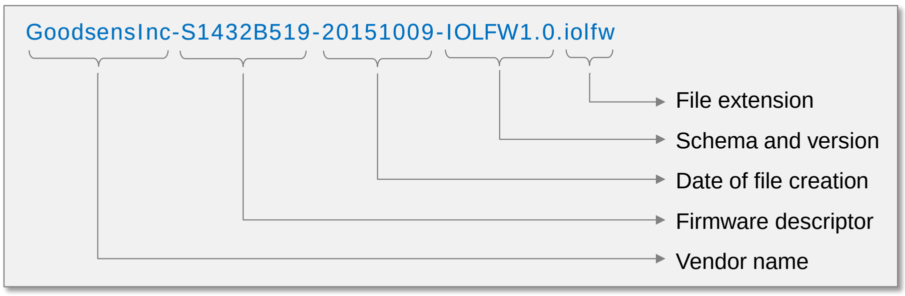


Figure 23 – Example of a FW-Update file name

7.4.4 Meta data file description

7.4.4.1 General

The metadata file format is based on extended markup language (xml). The file name shall be the same as for the entire archive except the extension that shall be "xml". The structure of the file is formally defined in the xml schema file (IOLFW1.0.xsd). The schema can be used to validate, edit and process the file in firmware update/viewer tools.

7.4.4.2 XML structure

Figure 24 shows the Meta data structure of FW-Update files.

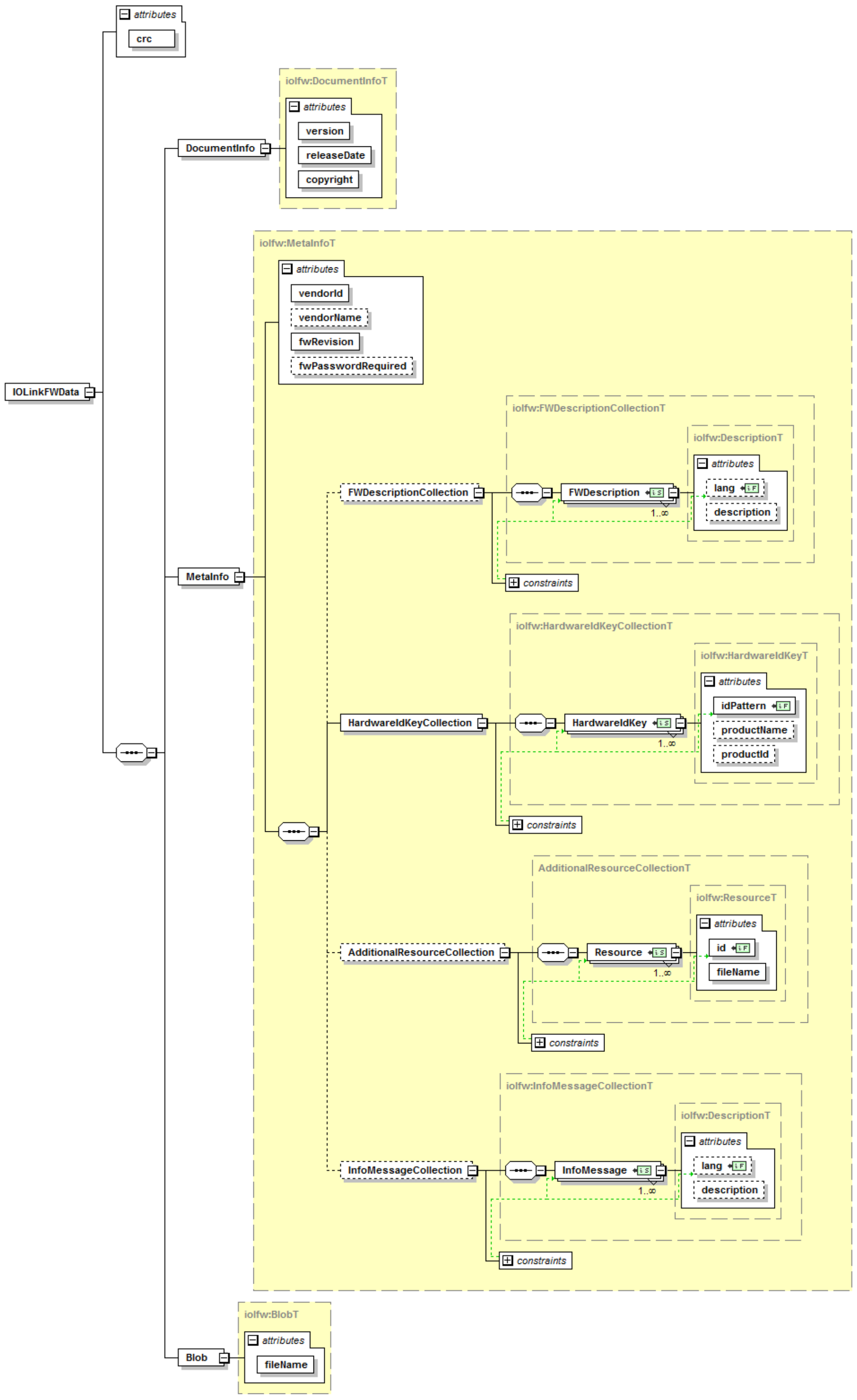


Figure 24 – Meta data XML structure

The elements in Figure 24 are defined in Table 9, Table 10, and Table 11.

7.4.4.3 Root elements

The root element of the FW-Update file is IOLinkFWData. It has one mandatory CRC attribute, which is a signature across the xml metadata file, the firmware binary file and all resource files listed in the xml element "IOLinkFWData →MetaInfo →AdditionalResourceCollection" (lower part of the yellow marked area in Figure 24). The root element shall contain the elements specified in Table 9.

Table 9 – Definition of the root elements

Item/attribute	M/O	Data type	Description
DocumentInfo	M	DocumentInfoT	This element contains common information about the file itself (version, release date, copyright).
MetaInfo	M	MetaInfoT	This element contains information about the Device firmware (Hardware IDs, revision, vendor details, description and additional resources).
Blob	M	BlobT	This element contains the binary data to be transferred to the Device via the FW-Update protocol. It has one mandatory attribute *fileName*. The *fileName* is the name (including an extension) of the file inside the firmware archive containing the binary data.
crc (signature)	M		CRC signature using the 32 bit CRC generator polynomial across the xml metadata file, the firmware binary file and all resource files listed in the xml element "AdditionalResourceCollection". The CRC polynomial shall be the same as with stamping of IODD files (see [2]).
Key M = mandatory; O = optional			

7.4.4.4 Document information

The terms used in item "DocumentInfo" in Figure 24 are specified in Table 10.

Table 10 – Definition of elements in DocumentInfo

Item	M/O	Data type	Description
version	M	string: V\d+(\.\d+){1,7})	This attribute contains the version of the FW-Update file (for example V1.02). Its actual format is vendor specific.
releaseDate	M	date: \d{4}-\d{2}-\d{2}	The date information in the FW-Update file name shall correspond to the "releaseDate" attribute in the DocumentInfo element.
copyright	M	string	Vendor-specific copyright text.
Key M = mandatory; O = optional			

7.4.4.5 Firmware meta information

The terms used in item "MetaInfo" in Figure 24 are specified in Table 11.

Table 11 – Definition of elements in MetaInfo

Item	M/O	Data type	Description
vendorId	M	unsignedShort	This attribute is the firmware vendor ID obtained from IO-Link Consortium. It shall be the same as the value encoded in the Device direct page parameters in address 0x07 and 0x08 (VendorID).
vendorName	O	string	This attribute is optional. It is recommended that it complies with the corresponding Device parameter.
fwRevision	M	string	This attribute describes the version of the firmware data. Its format is vendor specific. It should not be used to compare firmware versions by the FW-Update-Software. It is recommended

Item	M/O	Data type	Description
			that this attribute complies with the corresponding Device parameter.
fwPassword Required	O	boolean	Set this attribute to TRUE in order to request the manufacturer password during the FW-Update process. Default = "false".
FWDescription Collection	O		This element is optional. It consists of one or more FirmwareDescription elements.
FWDescription	M		This element contains any vendor specific information about the FW-Update file (release notes, for which Device types it is, etc.). The FW-Update-Software can show the information after the FW-Update file is available. It has two mandatory attributes *xml:lang* and *descriptionText*. The *xml:lang* attribute describes the language of the description according to ISO 639-1:2002 (standard two-letter format such as “en”, “de” etc.) and shall be unique across all *FirmwareDescription* inside *FirmwareDescriptionCollection*. The *descriptionText* attribute contains the text of the description.
HardwareIdKey Collection	M		This collection contains HardwareIdKeys. The ID links the FW-Update data to the Devices for which it can be used. The Devices shall have the corresponding ISDU parameter with the same ID. FW-Update-Software should read it from the Device and check whether it matches with one of the IDs in the file. The ID in the Device and one of the IDs in the file shall be the same to permit the update of the Device with the data in the file. The IDs in the file can end with a wildcard symbol "*" (idPattern attribute of each HardwareIdKey element). That means the ID in the Device shall match the string in front of "*". The characters after can differ.
HardwareIdKey	M	See idPattern	This is one of the elements in *HardwareIdKeyCollection*. It describes one of the hardware variants for which this firmware can be used for update. The element has one mandatory attribute *idPattern*. The *idPattern* attribute contains the text of this hardwareIdKey according to the pattern [A-Za-z][A-Za-z0-9 _-]*[A-Za-z0-9*]. This element can also have optionally *productName* and *productId* string attributes. It is recommended that the attributes comply with the corresponding Device parameter.
AdditionalResource Collection	O		This element is optional. It consists of one or more resource elements.
Resource	M	[a-zA-Z\d_.-]+	This element describes a file with additional resources (e.g. pictures, lookup tables etc.). It has two attributes *id* and *fileName*. The *id* attribute is a unique identifier of the resource over (?) all the other ones. The *fileName* attribute is the name of the resource file inside the firmware archive restricted by the pattern "([a-zA-Z\d_.-]+)".
InfoMessage Collection	O		It consists of one or more InfoMessage elements.
InfoMessage	M		This element contains any vendor specific user instructions which will appear after successful completion of the FW-Update process (hints for dealing with data storage, sensor recalibration instructions, etc.). It has two mandatory attributes "xml:lang" and "descriptionText". The "xml:lang" attribute describes the language of the description according to ISO 639-1:2002 (standard two-letter format such as “en”, “de” etc.) and shall be unique across all "InfoMessage" inside "InfoMessageCollection". The "InfoMessageText" attribute contains the text of the InfoMessage.
Key M = mandatory; O = optional			

7.5 Bootload management

7.5.1 Main activities

From a user's point of view, the entire firmware update (FW-Update) procedure consists of a sequence of six main activities as already indicated in Figure 21:

a) Identification of the connected Device

b) Acquisition of the FW-Update file
c) Compatibility verification between Device and file
d) Password entry and check (optional)
e) Execution of firmware update
f) Finalization and activation

Each activity is part of the FW-Updatesoftware tool and specified separately in clauses 7.5.2 to 7.5.7.The activities can be implemented for implicit automatic execution without user interaction except in case of faults or explicit execution as illustrated in Figure 34.

7.5.2 Device identification

The activity diagram of Device identification is shown in Figure 25. It refers to item 16 in the use case diagram in Figure 21.

The parameters HardwareRevision and FirmwareRevision are mandatory for this profile (see Table 12).

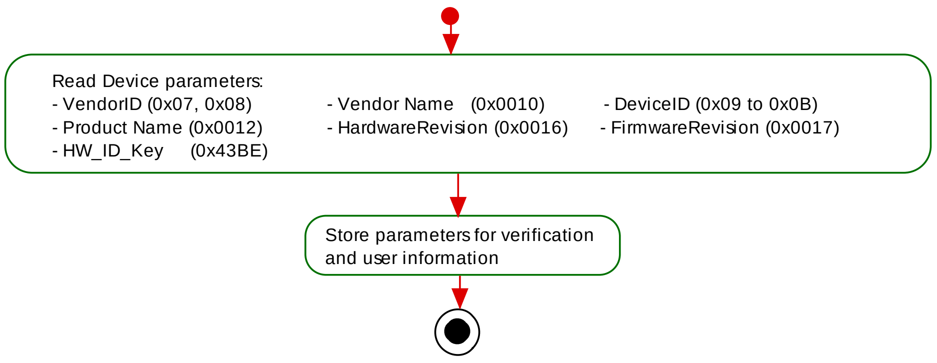

Read Device parameters:
- VendorID (0x07, 0x08)
- Vendor Name (0x0010)
- DeviceID (0x09 to 0x0B)
- Product Name (0x0012)
- HardwareRevision (0x0016)
- FirmwareRevision (0x0017)
- HW_ID_Key (0x43BE)
Store parameters for verification and user information

Figure 25 – Identification activity

The mandatory Device parameters VendorID, Vendor Name, DeviceID, Product Name, and HW_ID_Key are retrieved from the connected Device and stored locally for further activities.

7.5.3 Acquisition of the FW-Update file

Manufacturers or vendors shall provide FW-Update files in a standardized manner specified in this document (see 7.4). Figure 26 shows the activity diagram for the acquisition and the integrity check of the FW-Update file.

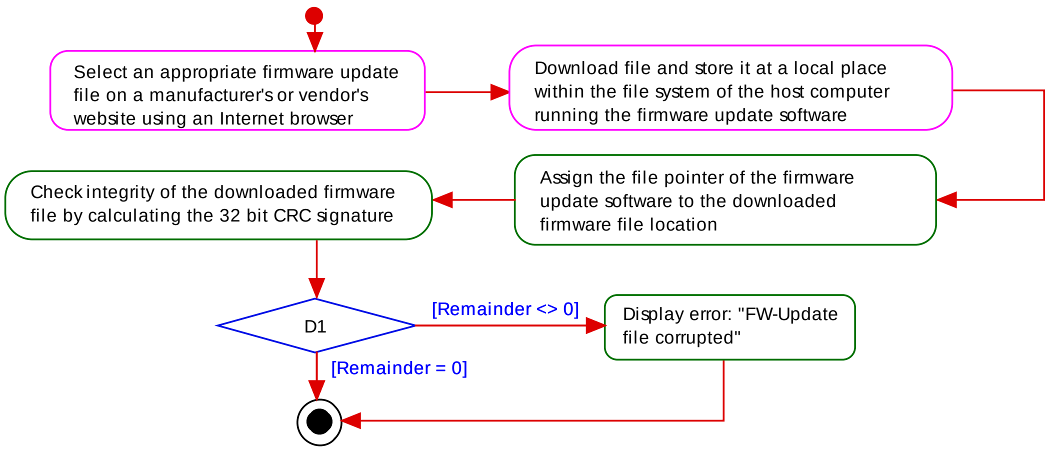

Select an appropriate firmware update file on a manufacturer's or vendor's website using an Internet browser
Download file and store it at a local place within the file system of the host computer running the firmware update software
Assign the file pointer of the firmware update software to the downloaded firmware file location
Check integrity of the downloaded firmware file by calculating the 32 bit CRC signature
D1
[Remainder <> 0]
Display error: "FW-Update file corrupted"
[Remainder = 0]

Figure 26 – FW-Update file acquisition and check

Activities in purple borders can be outside the FW-Update software tool.

7.5.4 Verification of FW-Update file compatibility

The Device parameters VendorID and HW_ID_Key are used for the compatibility check of the connected Device and the FW-Update file. The HW_ID_Key parameter of the Device shall match one of the HW_ID_Keys within the HW_ID_Key_Collection in the Meta information of the FW-Update file (see Table 10). Figure 27 shows the corresponding activity diagram.

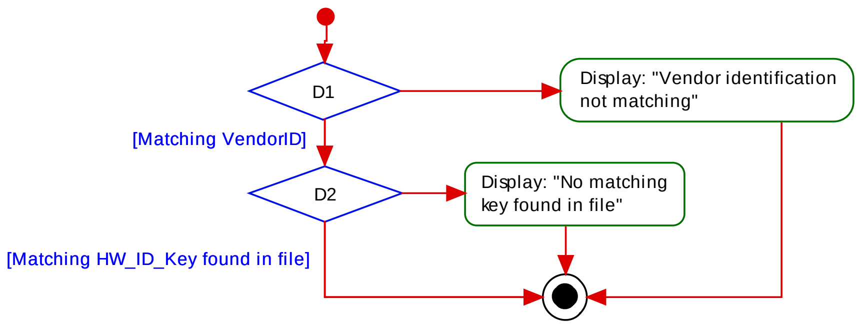

D1
Display: "Vendor identification not matching"
[Matching VendorID]
D2
Display: "No matching key found in file"
[Matching HW_ID_Key found in file]

Figure 27 – Verification of compatibility

7.5.5 Password entry and check (optional)

The Device provides an optional parameter "FW-Password" (see 7.6.7). The FW-Update software expects a user entry, whenever the attribute "fwPasswordRequired" is TRUE within the FW-Update file (see Table 11). In case of multiple Devices of same type to be updated, it is possible to store a successful password value and to skip re-entries (see 7.9.1). Figure 28 shows the corresponding activity diagram.

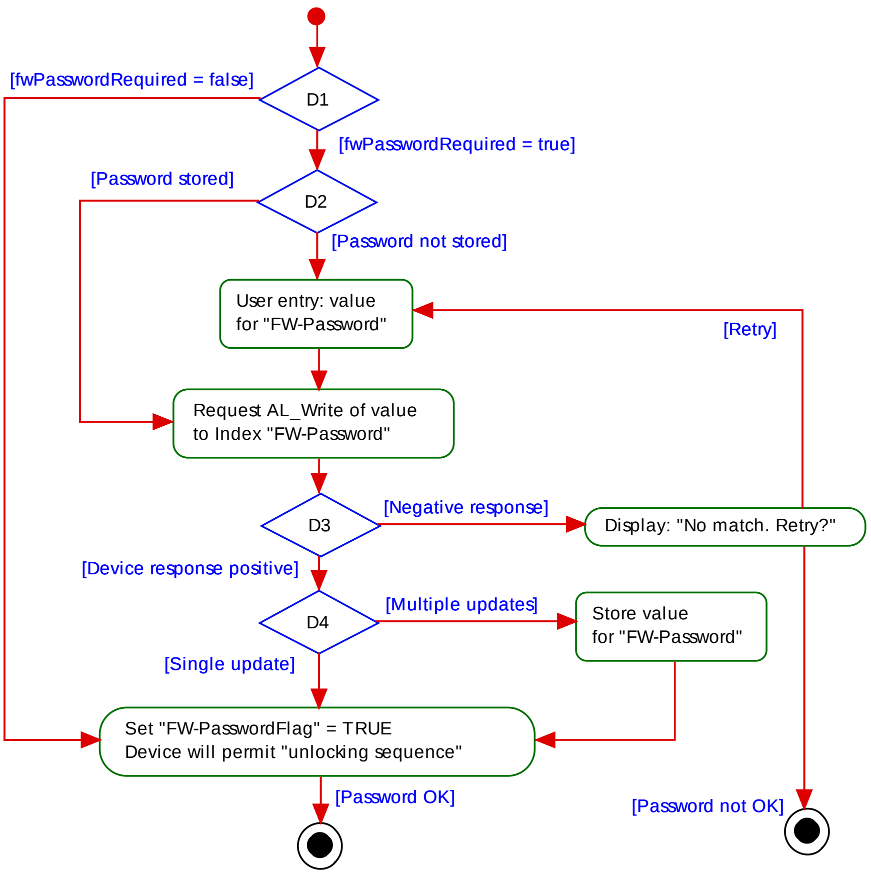

[fwPasswordRequired = false]
D1
[fwPasswordRequired = true]
[Password stored]
D2
[Password not stored]
User entry: value for "FW-Password"
[Retry]
Request AL_Write of value to Index "FW-Password"
[Negative response]
D3
Display: "No match. Retry?"
[Device response positive]
[Multiple updates]
D4
Store value for "FW-Password"
[Single update]
Set "FW-PasswordFlag" = TRUE Device will permit "unlocking sequence"
[Password OK]
[Password not OK]

Figure 28 – Password check

If password entry has been OK, the flag "FW-PasswordFlag" shall be set (see 7.7.2).

7.5.6 FW-Update via bootloader

This activity corresponds to a protocol with states and transitions and is thus specified in a separate clause 7.7.

7.5.7 Finalization and activation

If an error occurs during FW-Update the user shall be informed via a fault indication. Retries shall always begin with verification (see 7.5.4). Automatic retries shall be limited. Figure 29 shows the activity.

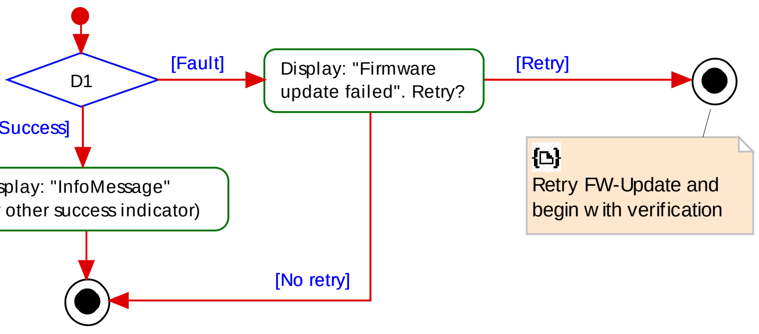

D1
[Fault]
Display: "Firmware update failed". Retry?
[Retry]
[Success]
Display: "InfoMessage" (or other success indicator)
Retry FW-Update and begin with verification
[No retry]

Figure 29 – Finalization and activation

7.6 Definitions and constraints

7.6.1 Initial Device operation (OPERATE/PREOPERATE)

Normally, a Device reaches the OPERATE state automatically without any tool intervention. Thus, the preferred state for FW-Update is the OPERATE state, where it is possible to chose the cycle time and optimize the update performance (see Annex D). If a Master port is configured to inspection level "TYPE_COMP" or "IDENTICAL", where the DeviceID shall match the configured value, the Device will stop in PREOPERATE. FW-Update is also possible in this state. However, only the fixed value in the MinCycleTime parameter (see Table B.1 in [1]) can be chosen.

7.6.2 Bootloader

As soon as the existing technology firmware of the Device is unlocked (see 0), the Device activates the Bootloader including the BLOB transfer mechanism. This means that the bootload mode is active after a communication reset of the Device.

Once the Bootloader is active, the Device shall react on the wake-up request of the Master and comply with the message handling of [1].

The Bootloader can be functionally downsized as specified in 7.10.1.

7.6.3 IODD for Bootload mode

A second IODD for FW-Update is not required. A Device comes already with an IODD containing the necessary information such as the parameter description for HW_ID_Key (see Annex A).

7.6.4 M-sequence types

In bootload mode it is mandatory to support M-sequence type "TYPE_0". This M-sequence type supports the transfer of one octet On-request Data (OD) per message, which gives it 75% overhead. This is applicable for the phases STARTUP, PREOPERATE and OPERATE.

For higher download speeds it is recommended to implement one of the following M-sequence types:

- TYPE_1_V code 6, 8 octets OD, no PD (28% overhead)
- TYPE_1_V code 7, 32 octets OD, no PD (9% overhead)

A manufacturer/vendor can decide to implement M-sequence types with PD (Process Data).

7.6.5 DeviceID versus Boot_DeviceID

In bootload mode the manufacturer shall change the DeviceID to a unique value within the manufacturer specific used range of values to indicate the Device is in bootload mode.

7.6.6 VendorID

The VendorID shall not be changed in bootload mode.

7.6.7 FW-Update specific parameters

The Device shall provide the additional parameters listed in Table 12 for the FW-Update profile ("conditional"). The parameters use Indices reserved for Device profiles and supplement Table B.8 in [1].

Table 12 – Device parameters reserved for FW-Update

Index (dec)	Object name	Access	Length	Data type	M/O/C	Definitions
…						
0x43BD (17341)	FW-Password	W	variable	StringT	O/C	64 octets (ASCII) is maximum length.
0x43BE (17342)	HW_ID_Key	R	variable	StringT	C	64 octets (ASCII) is maximum length. Pattern is [A-Za-z][A-Za-z0-9 _-]*[A-Za-z0-9*].
0x43BF (17343)	Bootmode Status	R	1 octet	UIntegerT	C	
…						
Key M = mandatory; O = optional; C = conditional						

7.6.7.1 FW-Password

This parameter shall be "write only". Device manufacturers shall set the attribute "fwPasswordRequired" = TRUE within the FW-Update file (see Table 11). The Device expects an AL_Write of the correct password value to the "FW-Password" Index prior to the unlocking of the firmware/bootloader (see 7.5.5).

7.6.7.2 HW_ID_Key

This profile-conditional read only parameter shall be used for the identification of valid FW-Update files. The HW_ID_Key will be checked against the meta information "HardwareIdKeyCollection", wherein one of the listed HW_ID_Keys should match in order to start an update process (see Table 11).

Format example: SDAT-MHS-160

7.6.7.3 BootmodeStatus

This profile-conditional read only parameter shall be used as a flag to indicate whether the Bootloader is active or inactive. Table 13 shows the coding.

Table 13 – Coding of BootmodeStatus

Code	Definition
0x00	Bootloader inactive
0x01	Bootloader active
0x02 to 0xFF	Reserved

7.6.8 FW-Update specific SystemCommands

The required FW-Update SystemCommands are listed in Table 14. This table supplements Table B.9 in [1]. A Device equipped with the Bootloader mechanism shall support these commands (see 0 and 7.7.4).

Table 14 – Coding of FW-Update SystemCommands

Command (hex)	Command (dec)	Command name	M/O/C	Definition
…				
0x50	80	BM_UNLOCK_S	C	Start unlocking sequence
0x51	81	BM_UNLOCK_F	C	Unlocking command 1
0x52	82	BM_UNLOCK_T	C	Unlocking command 2
0x53	83	BM_ACTIVATE	C	Stop communication and activate new firmware
…				

7.6.9 Unlocking sequence

Unlocking of the existing firmware is performed through a particular sequence of System-Commands sent to the Device. This sequence is designed such that accidental flashing of the Device is very unlikely to occur. All of the following steps are mandatory except password.

1) It is a precondition for the Device to be in communication (see 7.6.1).
2) The FW-Update software tool retrieved the HW_ID_Key from the Device (see 7.5.2 and 7.6.7.2) and checks it against the list in the FW-Update file (see 7.5.4).
3) Password protection is optional. Maintenance updates usually are not protected while feature enhancements will be (see 7.6.7.1).
4) In case of a match, the FW-Update software tool sends the sequence of SystemCommands shown in Figure 30 for unlocking the firmware and switching the Device into the Bootloader state (see Figure 5).

   The BM_UNLOCK_S SystemCommand is always accepted. Receiving this at any time does not generate an error and (re)starts the unlock sequence.

   There is no time-out between the SystemCommands. ISDU communication and other access are permitted while the unlocking process is ongoing.
5) In case a communication startup is detected (see state "Startup_2" in Figure 38 in [1]) before Bootloader mode is entered, the unlocking sequence shall be restarted.
6) After successful reception of the sequence, the Device shall respond with a positive acknowledgment. Upon reception of a faulty sequence, the Device shall respond with a negative acknowledgment: ISDU error 0x8036 *"Function temporarily not available, system command rejected"*.

7.6.10 Required BLOB_IDs

Table 15 shows the BLOB_IDs in use for firmware updates (see 6.5.2).

Table 15 – BLOB_IDs for FW-Update

Value (dec)	Definition
0	Idle transmission of a BLOB
1	FW-Update (write)

7.7 FW-Update protocol

7.7.1 Protocol layers

Figure 20 illustrates the transmission of firmware updates within an automation system with IO-Link. The "FWU_Trans_Layers" contain the necessary state machines using the BLOB transmission engines specified in 6.6.1 and 6.6.2 and provide the necessary protocol features between a Device and a PC-based tool.

7.7.2 Device FW-Update state machine

Figure 30 shows the state diagram of the "Bootloader_Layer" (see Figure 20). States 5 to 12 are nested in state 4 allowing these states to react on errors (T13 and T14) within all of these states (see 3.3.1).

After power-up, the Device performs a firmware check (state "0") before entering the regions (see 3.3.1) of state "Device_Operating_1".

The Device is able to receive the AL_Write service with the value of the "FW-Password" in state "DataExchange_2" (see 7.5.5). Once the written value matches the hard-coded value behind the Index of "FW-Password", the Device sets the "FW-PasswordFlag" = TRUE and is ready for the unlocking sequence.

It is important for the Device to switch into Bootmode (state 14) before any changes are applied to the internal flash memory affecting the technology application of the Device.

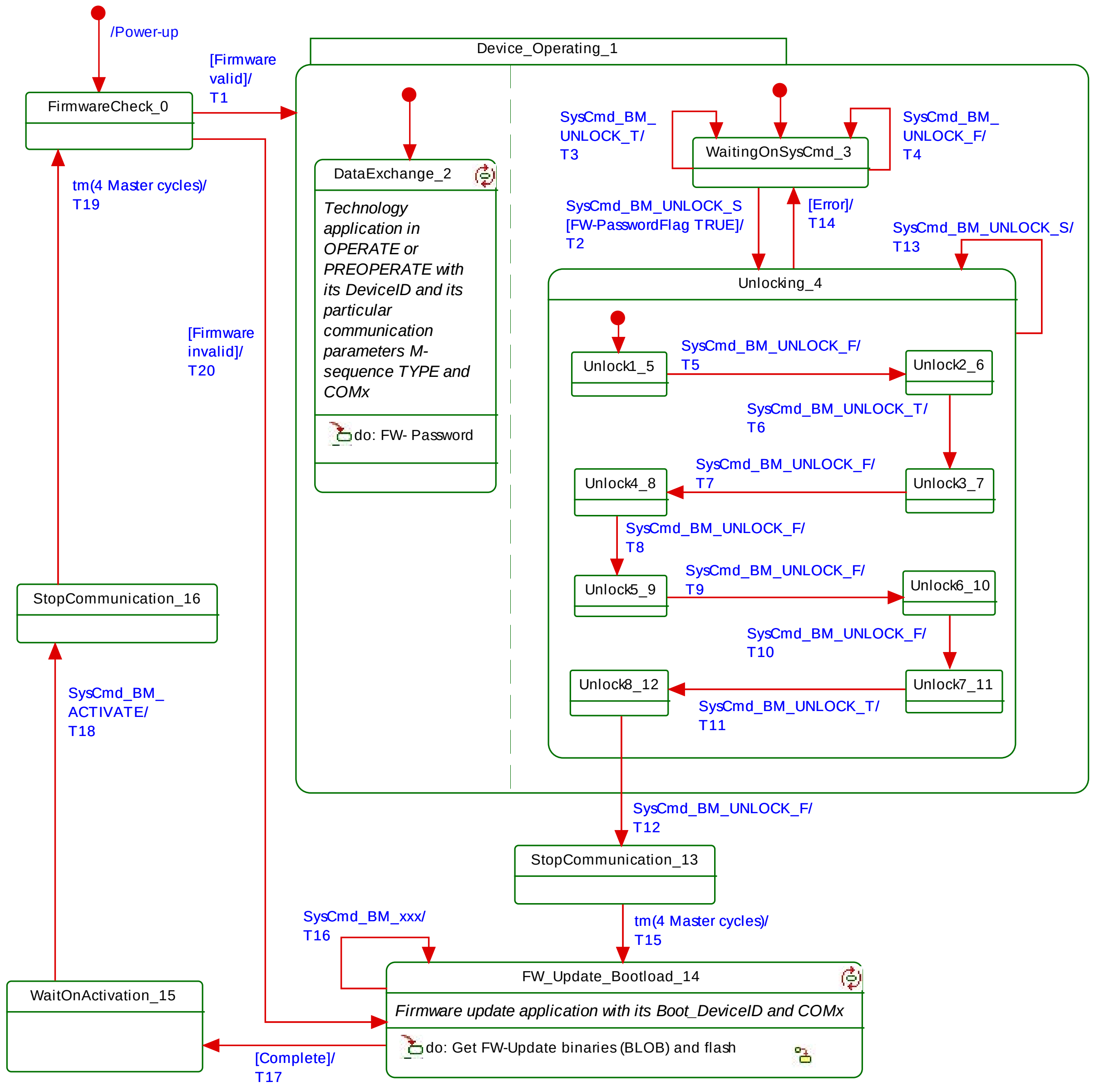


Figure 30 – Device FW-Update state machine

Not more than 2 subsequent M-sequences shall fail during flash processing to keep communication ongoing.

Any communication restart shall not affect the internal states and parameters of the BLOB transfers and the FW-Update.

The activity in state 14 "FW-Update_Bootload" is specified in 7.7.3.

Table 16 shows the state transition tables of the Device bootloader state machine.

Table 16 – State transition tables of the Device bootloader state machine

STATE NAME	STATE DESCRIPTION
FirmwareCheck_0	After power-on of the Device, the validity of the Device's firmware is checked. The check is manufacturer specific. Valid firmware leads to a regular start of the Device according to [1]. Invalid firmware initiates the Bootloader and a FW-Update is started.
Device_Operating_1	Device is in OPERATE state (or PREOPERATE). See 7.6.1.
DataExchange_2	The technology application is exchanging data while using its appropriate M-sequence TYPEs and COMx transmission rate
WaitingOnSysCmd_3	A separate additional firmware extension is started and waits on a particular System-Command "BM_Unlock_S", the first command of the unlocking sequence. Any other SystemCommand is ignored.
Unlocking_4	This superstate monitors the reception of the correct unlocking sequence. Any Sys-temCommand "BM_Unlock_S" will restart the entire sequence check. Any mismatch of received and expected SystemCommands will abandon the superstate and return to the previous state.
Unlock1_5	Expected SystemCommand is "BM_Unlock_F"
Unlock2_6	Expected SystemCommand is "BM_Unlock_T"
Unlock3_7	Expected SystemCommand is "BM_Unlock_F"
Unlock4_8	Expected SystemCommand is "BM_Unlock_F"
Unlock5_9	Expected SystemCommand is "BM_Unlock_F"
Unlock6_10	Expected SystemCommand is "BM_Unlock_F"
Unlock7_11	Expected SystemCommand is "BM_Unlock_T"
Unlock8_12	Expected SystemCommand is "BM_Unlock_F"
StopCommunication_13	The firmware extension causes the Device to stop current communication and thus forces the Master to restart communication via Wake-up.
FW_Update_Bootload_14	The Device re-establishes communication with new communication parameters (e.g. transmission rate). The Device BLOB state machine is activated. In this state the boot-loader receives segment by segment of the FW-Update binary using the BLOB trans-mission state machine (see 6.6.1). Any SystemCommand will be ignored. Received segments are passed over to the flashing mechanism (see activity diagram in 7.7.3).
WaitOnActivation_15	After correct or incorrect flashing, the Device waits on a SystemCommand "BM_ACTIVATE".
StopCommunication_16	The Bootloader causes the Device to stop current communication and thus forces the Master to restart communication via Wake-up. After 4 Master cycles, the Bootloader switches to the new technology firmware.

TRANSITION	SOURCE STATE	TARGET STATE	ACTION
T1	0	1	Device start until OPERATE (or PREOPERATE), see 7.6.1. Set "FW-PasswordFlag" = FALSE (optional).
T2	3	4	–
T3	3	3	Unexpected SystemCommand. Return ErrorCode 0x8020 -"Service tempo-rarily not available"
T4	3	3	Unexpected SystemCommand. Return ErrorCode 0x8020 -"Service tempo-rarily not available"
T5	5	6	Acknowledgment
T6	6	7	Acknowledgment
T7	7	8	Acknowledgment
T8	8	9	Acknowledgment

TRANSITION	SOURCE STATE	TARGET STATE	ACTION
T9	9	10	Acknowledgment
T10	10	11	Acknowledgment
T11	11	12	Acknowledgment
T12	12	13	Acknowledgment
T13	4	4	Received SystemCommand BM_Unlock_S. New unlocking sequence.
T14	5,6,7,8,9, 10,11,12	3	SystemCommand out of sequence. Return ErrorCode 0x8036 "Function temporarily not available, system command rejected"
T15	13	14	The firmware extension establishes a (manufacturer specific) Boot_DeviceID and optimized transmission parameters (e.g. COM3) for the download of the FW-Update binaries.
T16	14	14	Unexpected SystemCommand. Return ErrorCode 0x8020 -"Service temporarily not available"
T17	14	15	Acknowledgment. Return "Update completed"
T18	15	16	Acknowledgment. Return "Firmware check"
T19	16	0	Acknowledgment
T20	0	14	–

INTERNAL ITEMS	TYPE	DEFINITION
Acknowledgment	Variable	
FW-PasswordFlag	Bool	See 7.5.5

7.7.3 Reception of binaries and flashing activity

The activity diagram for state 14 (FW_Update_Bootload) in Figure 31 shows the following actions:

- Reception of FW-Update file binaries (BLOB transfer),
- Flashing, and
- Firmware check.

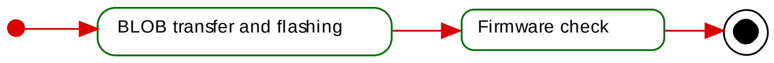


Figure 31 – BLOB and flashing activity

The activity uses the BLOB state machines in 6.6 to receive the FW-Update file binaries segment by segment until "BLOB_Last" arrives. In case the Device does not have enough memory space, each segment will be flashed after reception. Otherwise, the Device receives the entire binaries prior to flashing.

NOTE For performance reasons, the flashing of segments should not slow down the transmission of segments.

7.7.4 Master behavior

In state 13 (StopCommunication) of the Device's FW-Update state machine, the firmware extension of the Device causes a communication interrupt and switches to Bootload mode. As a consequence, the Master will restart the communication but with different parameters:

- In STARTUP phase the Master reads the Direct Parameter page 1 parameters including the VendorID and the DeviceID which is now the Boot_DeviceID.
- The Master readjusts the corresponding port to an appropriate M-sequence and COM transmission rate in order to achieve optimized performance (see Annex D).

In state 16 (StopCommunication) of the Device's FW-Update state machine, the firmware extension of the Device causes another communication interrupt and switches to the updated firmware. As a consequence, the Master will restart the communication but with original parameters: DeviceID, M-sequence, and COM transmission rate.

7.7.5 Tool FW-Update state machine

Figure 32 shows the state diagram of the "T_FWU_Trans_Layer" (see Figure 20).

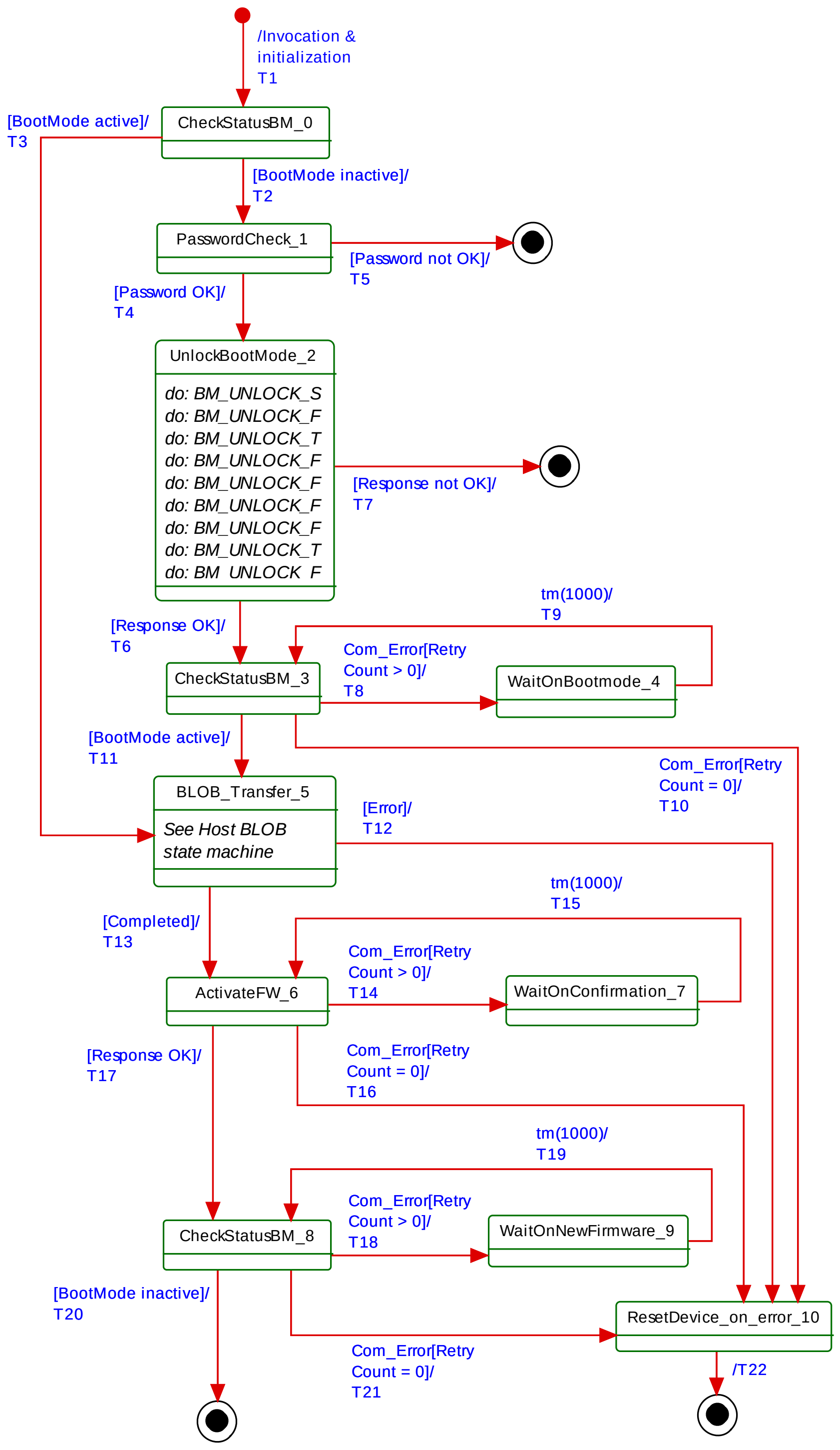


Figure 32 – Tool FW-Update state machine

The FW-Update software tool checks first whether the firmware of the Device is active and then enters the password check (option). After sending the unlocking sequence it starts the BLOB transmission (FW-Update binary). The following rules shall be observed:

- Any interruption of communication with subsequent restart of the communication shall not abort the BLOB transmission sequence.
- The host FW-Update tool shall not abort the BLOB when receiving a timed-out ISDU.
- The FW-Update can be aborted upon user request.

Table 17 shows the state transition tables of the Device FW-Update state machine.

Table 17 – State transition tables of the Tool FW-Update state machine

STATE NAME	STATE DESCRIPTION
CheckStatusBM_0	Read Bootloader status via parameter "BootmodeStatus" (see 7.6.7.3)
PasswordCheck_1	Check whether entered password matches password stored in Device (see 7.5.5)
UnlockBootMode_2	Send unlocking sequence of SystemCommands to the Device (see 0)
CheckStatusBM_3	Read Bootloader status via parameter "BootmodeStatus". If response is not OK, retry if count is not "0"
WaitOnBootmode_4	Wait 1 s (1000 ms) before return to state 3
BLOB_Transfer_5	Transfer body of the FW-Update file to the Device using the host BLOB state machine (see 6.6.2)
ActivateFW_6	New firmware activated by means of SystemCommand "BM_ACTIVATE". Retry in case of a negative response "SERV_NOTAVAIL_DEVCTRL" if count is not "0"
WaitOnConfirmation_7	Wait 1 s (1000 ms) before return to state 6
CheckStatusBM_8	Read Bootloader status via parameter "BootmodeStatus". If response is not OK, retry if count is not "0"
WaitOnBootmode_9	Wait 1 s (1000 ms) before return to state 8
ResetDevice_on_error_10	A SystemCommand 0x80 causes a move of the Device to Bootloader or technology FW

TRANSITION	SOURCE STATE	TARGET STATE	ACTION
T1	init	0	–
T2	0	1	–
T3	0	5	–
T4	1	2	–
T5	1	exit	–
T6	2	3	RetryCount = 3
T7	2	exit	–
T8	3	4	RetryCount = RetryCount -1
T9	4	3	–
T10	3	10	–
T11	3	5	–
T12	5	10	–
T13	5	6	RetryCount = 3
T14	6	7	RetryCount = RetryCount -1
T15	7	6	–
T16	6	10	–
T17	6	8	RetryCount = 3
T18	8	9	RetryCount = RetryCount -1
T19	9	8	–
T20	8	exit	–
T21	8	10	

TRANSITION	SOURCE STATE	TARGET STATE	ACTION
T22	10	exit	–

INTERNAL ITEMS	TYPE	DEFINITION
RetryCount	Variable	Retry counter, starting from "3" and decrementing to "0"
BootMode active/inactive	Flag	Parameter "BootModeStatus" (see 7.6.7.3).
Error	Variable	Possible errors: see BLOB state machines.
Com_Error	Variable	Master specific

7.7.6 Sequence charts

Figure 33 shows the sequence chart of the entire FW-Update procedure. For the sake of better readability, the involvement of the Master is not shown during and after the BLOB transfer.

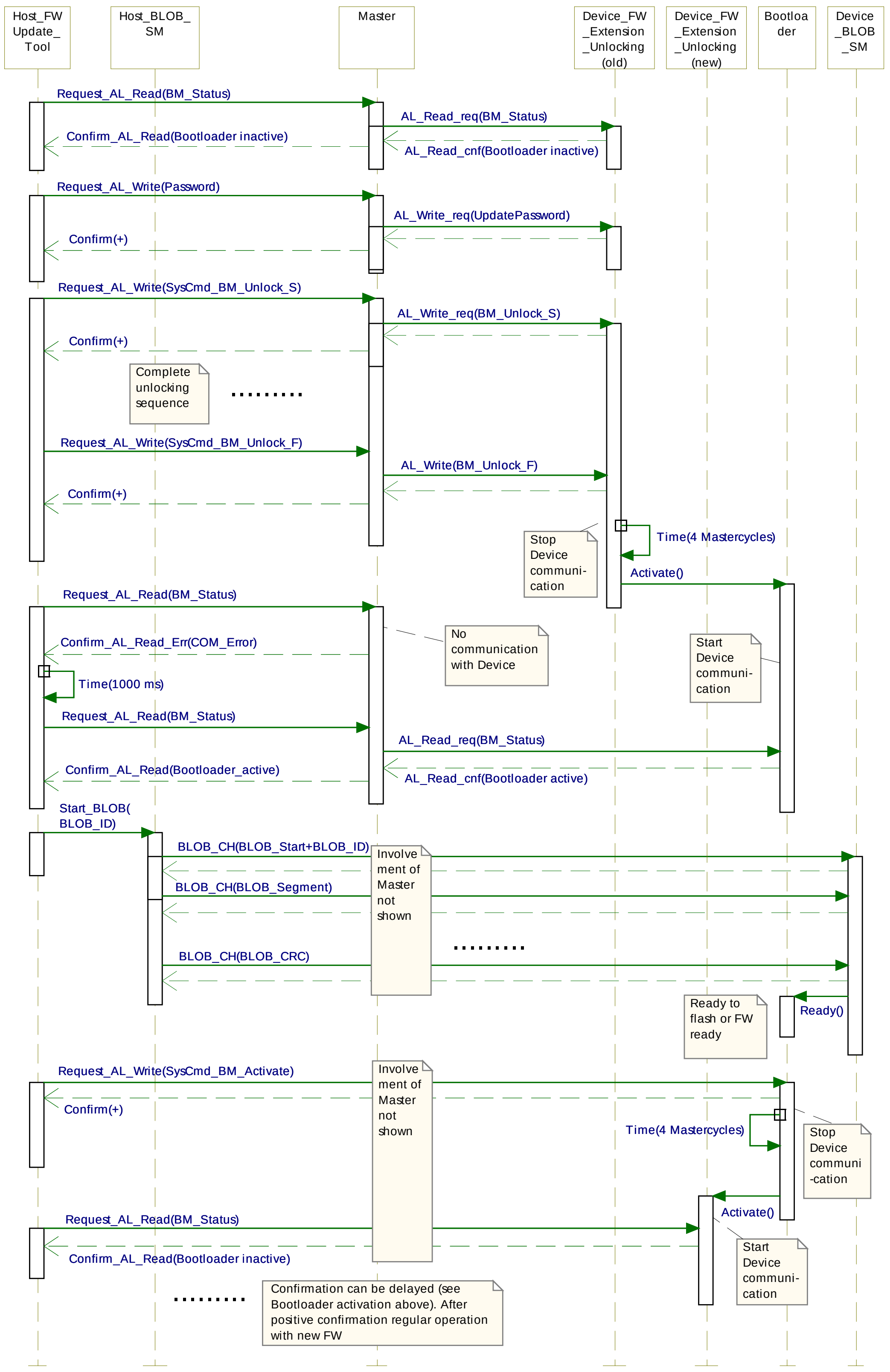

Host_FW Update_ Tool
Host_BLOB_ SM
Master
Device_FW _Extension _Unlocking (old)
Device_FW _Extension _Unlocking (new)
Bootloader
Device _BLOB _SM
Request_AL_Read(BM_Status)
AL_Read_req(BM_Status)
Confirm_AL_Read(Bootloader inactive)
AL_Read_cnf(Bootloader inactive)
Request_AL_Write(Password)
AL_Write_req(UpdatePassword)
Confirm(+)
Request_AL_Write(SysCmd_BM_Unlock_S)
AL_Write_req(BM_Unlock_S)
Confirm(+)
Complete unlocking sequence
Request_AL_Write(SysCmd_BM_Unlock_F)
AL_Write(BM_Unlock_F)
Confirm(+)
Stop Device communi- cation
Time(4 Mastercycles)
Activate()
Request_AL_Read(BM_Status)
No communication with Device
Confirm_AL_Read_Err(COM_Error)
Start Device communi- cation
Time(1000 ms)
Request_AL_Read(BM_Status)
AL_Read_req(BM_Status)
Confirm_AL_Read(Bootloader_active)
AL_Read_cnf(Bootloader active)
Start_BLOB(BLOB_ID)
BLOB_CH(BLOB_Start+BLOB_ID)
Involvement of Master not shown
BLOB_CH(BLOB_Segment)
BLOB_CH(BLOB_CRC)
Ready()
Ready to flash or FW ready
Request_AL_Write(SysCmd_BM_Activate)
Involvement of Master not shown
Confirm(+)
Time(4 Mastercycles)
Stop Device communi -cation
Activate()
Request_AL_Read(BM_Status)
Start Device communi- cation
Confirm_AL_Read(Bootloader inactive)
Confirmation can be delayed (see Bootloader activation above). After positive confirmation regular operation with new FW

Figure 33 – Sequence chart of the FW-Update procedure

7.8 Validation/responsibility

The FW-Update tool manufacturer is responsible for the proper function of the tool and its conformity with this specification via manufacturer declaration based on the tests in [6] and in Annex E.

The Device manufacturer is responsible for the proper function of a Device after a FW-Update and the conformity with the IO-Link specifications via manufacturer declaration based on the tests in [6] and in Annex E. The tests include the unlocking and the Bootloader mechanisms.

The manufacturer provides information about appropriate corrective actions in case an update failed, e.g. recovery, customer service, etc.

7.9 Common look and feel

7.9.1 Graphical user interface (GUI) and functions

Figure 34 illustrates exemplary the GUI functions shown in 7.3. The tab "Firmware" shall only be enabled or available if a Device supports FW-Update. The layout is conceptual and not mandatory. The aspect of multiple updates of several Devices of the same type is not shown.

It is possible to split the functions into a PC part for FW-Update file acquisition and pre-verification. A dedicated tool can then perform the second part comprising verification and FW-Update start. It is up to the manufacturer of the tool to provide appropriate indication (e.g. LED), for example the verification result and the update progress.

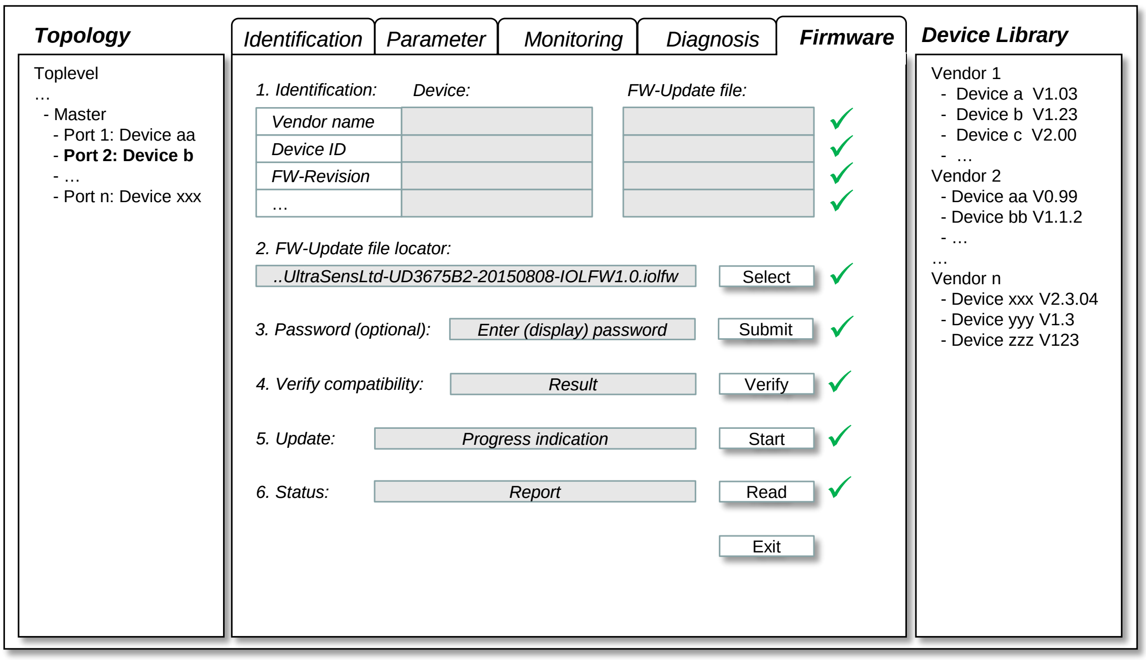


Figure 34 – Exemplary illustration of the GUI functions

The connected Device is identified in step 1 and relevant parameters are displayed (see fields below "Device:"). The FW-Update file is selected in step 2. This example assumes files that are already acquired via Internet, data media, or other means.

If the attribute "fwPasswordRequired" is set TRUE, the entry of a password is necessary in step 3. In case of multiple updates the password is stored for the sake of ease of use (see 7.5.5 for details). Selection of another FW-Update file overwrites password entries.

A crosscheck between the Device parameters and the items in "HardwareIdKey" (Table 11) is performed in step 4 and leads to the display of the items in step 1 (see fields below "FW-Update file:").

The update is performed in step 5 via the "Start" button and progress is indicated. The button is only activated if the verification in step4 has been successful. The progress indicator shall be displayed as a bar or "%" for the transfered share of the total of octets.

A detailed report can be retrieved via the "Read" button in step 6. This report can comprise information about the successful update or error information. It can also comprise an individu-al "InfoMessage" provided by the manufacturer (see Table 11).

The FW-Update application can be finalized via the Exit button. This button is disabled during an ongoing firmware update process.

7.9.2 Multi-language terms

Table 18 shows the terms and the definitions that should be used whenever a graphical user interface is implemented in a FW-Update tool.

Text for other languages can be retrieved from the IO-Link website (www.io-link.com).

Table 18 – Human interface terms

Term	Definition
Identification	Device parameters involved are specified in 7.5.2
Device	[1]
FW-Update file	See 7.4
VendorID	[1], Table B.1
Vendor Name	[1], Table B.8
DeviceID	[1], Table B.1
Product Name	[1], Table B.8
HardwareRevision	[1], Table B.8
FirmwareRevision	[1], Table B.8
HW_ID_Key	See 7.6.7.2
FW-Revision	[1], Table B.8
Locator	Directory path to the FW-Update file
Select	Browse to FW-Update file
Password	See 7.5.5
Submit	Transmit entry
Verify	See 7.5.4
Compatibility	See 7.5.4
Result	Detailed verification information: success or mismatches
Progress indicaton	See 7.9.4
Start	Initiate Bootloader and update process
Status	Detailed update information: success or abort
Report	User information: "InfoMessage", see Table 11
Read (print)	Detailed success report with relevant parameters or error/abort report
Exit	Quit FW-Update. Disabled during an ongoing firmware update process.

7.9.3 Error displays of FW-Update tools

Table 19 shows a list of possible error displays of FW-Update tools.

Table 19 – Error displays of FW-Update tools

Operational phase	Error/status display	Corrective action (Help)
Startup/connection	Device does not support firmware update (Profile)	Read and check parameter "Profile-Characteristic"
	Firmware update version not supported	Read and check parameter "Profile-Characteristic"
File select	Firmware update file is corrupted and cannot be opened	Contact customer/sales service
	Tool does not support firmware update file	Contact customer/sales service

Operational phase	Error/status display	Corrective action (Help)
	version	
	Selected file is not a firmware update file	Look for a file with extension .iolfw
Password	Password required	Enter password
	Password incorrect	Correct entry
Verification	Connected Device not supported by file	Look for appropriate file or contact customer/sales service
	Status: Connected Device supported, check manually	Check parameter by parameter
BLOB and ISDU	Invalid BLOB_ID	BLOB_ID shall be "1" or "0" (see Table 15)
	Incorrect BLOB_ID	Check IODD (see Annex A)
	ISDU transaction failed	Retry, then contact customer/sales service
Update/status	Status: Update may take several minutes	Abort or wait
	Status: Update completed	Read/print report ("InfoMessage"), see Table 11
	Update failed	Retry, then contact customer/sales service
	Update denied due to an active process	Retry later
	Time-out	Retry, then contact customer/sales service
	CRC error	Retry, then contact customer/sales service
	Activation of new firmware failed	Repeat and/or contact customer/sales service

7.9.4 Indications

The following rules apply for indications:

a) Progress indicator shall show % of full length of the data object to be transmitted.

b) Successful completion shall be indicated by a green symbol (preferably check mark).

c) Faults shall be indicated on displays by a red cross ⊗.

d) A Device's indicator should indicate an unsuccessful FW-Update if possible. The indication shall be described in the user manual.

7.10 Recommended strategies

7.10.1 Design aspects

The Device shall behave in Bootload mode as Device communicating with any Master that conforms with [1]. However, in Bootload mode functional downsizing of the Device to a certain degree with respect to [1] is possible.

The layer structure of a Device is shown in Figure 5 of this profile and in Figure 35 below ([1]).

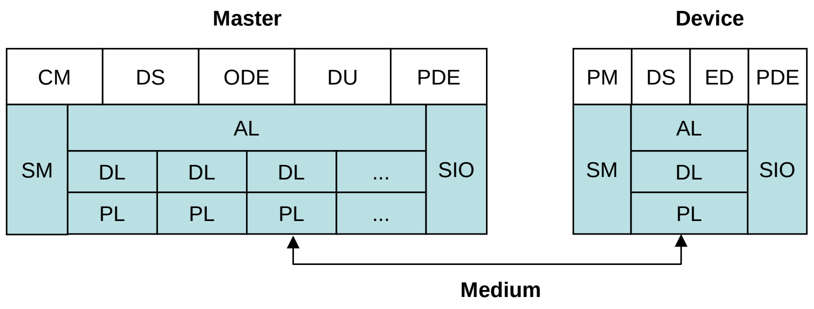


Figure 35 – Layer structures of Master and Device

For conformance, the Physical Layer (PL), the Data Link Layer (DL) and the Application Layer (AL) shall be implemented in Bootload mode (BM). The System Management (SM) and the SIO-Mode shall be implemented as far as required for the wake-up to switch between operational modes and to switch back into the regular technology firmware of the Device (TFW).

Regarding Device applications and features, only the Parameter Manager (PM) shall be present in Bootload mode. The applications

- Data Storage (DS),
- Event Dispatcher (ED), and
- Process Data Exchange (PDE)

are not required and should not to be implemented in Bootload mode.

The features

- Device access locks,
- Dynamic parameters, and
- Block parameterization

are not required and can be omitted in Bootload mode.

The following tables provide specifications of items which affect Devices supporting the BLOB & FW-Update profile.

Table 20 shows the affected items of the Direct Parameter page 1.

Table 20 – Affected items of the Direct Parameter page 1

Item	Index	Ref	BM	TFW	Comment
VendorID 1 + 2	0x07, 0x08	[1] , Table B.1	M	M	Identical in BM and TFW
DeviceID 1 + 2 + 3	0x09, 0x0A, 0x0B	[1] , Table B.1	M	M	Not identical in BM and TFW
Key for BM and TFW	M = mandatory, O = optional, C = conditional, – = not permitted				

Table 21 shows the affected MasterCommands.

Table 21 – Affected MasterCommands

Item	Index	Ref	BM	TFW	Comment
MasterIdent	0x95	[1] , Table B.2	M	M	Standard behavior
DeviceIdent	0x96	[1] , Table B.2	M	M	Standard behavior

Item	Index	Ref	BM	TFW	Comment
DeviceStartup	0x97	[1] , Table B.2	M	M	Standard behavior
ProcessDataOutput Operate	0x98	[1] , Table B.2	–	M	Ignored in BM
DeviceOperate	0x99	[1] , Table B.2	M	M	Standard behavior
DevicePreoperate	0x9A	[1] , Table B.2B1.2	M	M	Standard behavior
Key for BM and TFW	M = mandatory, O = optional, C = conditional, – = not permitted				

Table 22 shows the affected ISDUs.

Table 22 – Affected ISDUs

Item	Index	Ref	BM	TFW	Comment
HW_ID_Key	0x43BE	6.6.7	M	C	–
BLOB_ID	0x0031	5.5.1	M	O	–
BLOB_CH	0x0032	5.5.1	M	O	–
Bootmode_Status	0x43BF	6.6.7	M	C	–
Update-Password	0x43BD	6.6.7	–	C	–
Profile Characteristic	0x000D	[1] , Table B.8	M	O	See clause 5
Hardware Revision	0x0016	[1] , Table B.8	M	M	Can be different according to company policy
FirmwareRevision	0x0017	[1] , Table B.8	M	M	Refers in BM to the revision of the Bootloader
ProductName	0x0012	[1] , Table B.8	M	M	–
ProductId	0x0013	[1] , Table B.8	O	O	–
Key for BM and TFW	M = mandatory, O = optional, C = conditional, – = not permitted				

Table 23 shows the affected SystemCommands.

Table 23 – Affected SystemCommands

Item	Index	Ref	BM	TFW	Comment
Device reset	0x80	[1] Table B.9	M	O	BM: reset blob transfer
BM_UNLOCK_S	0x50	6.6.8	–	M	–
BM_UNLOCK_F	0x51	6.6.8	–	M	–
BM_UNLOCK_T	0x52	6.6.8	–	M	–
BM_ACTIVATE	0x53	6.6.8	M	–	–
Key for BM and TFW	M = mandatory, O = optional, C = conditional, – = not permitted				

Table 24 shows the affected services.

Table 24 – Affected services

Item	Index	Ref	BM	TFW	Comment
FW-Password check		6.7.2	–	C	Mandatory for FW-Update profile
FW validity check		6.7.2	M	C	–
BLOB_Transfer		5.5	M	O	–
Firmware flashing		–	M	O	–
Key for BM and TFW M = mandatory, O = optional, C = conditional, – = not permitted					

7.10.2 Distribution to multiple destinations within the Device

It is within manufacturer's responsibility to manage multiple destinations within the Device via one single BLOB (FW-Update file).

7.10.3 Compatibility levels

Manufacturer has a possibility to achieve compatibility via the parameter HW_ID_Key and the meta information within the FW-Update file (see 7.4.4.5). Further compatibility levels are within manufacturer's responsibility if needed.

Annex A
(normative)
IODD extensions

A.1 Overview

The IODD syntax is defined within the IODD specification and is enforced by the companion XML schema file. Thus, any change to the IODD syntax requires a new version of the IODD specification and schema.

Creating a new IODD version is a complex, time consuming task. To make things worse, each new version causes labor for the tool manufacturers and may increase complexity for users. As a consequence, new versions of the IODD specification and schema are only published in case of major updates.

Therefore, the IODD definitions for BLOB transfer and FW-Update are done in two steps. An intermediate step allows this profile to go ahead and work with IODDs based on the existing IODD V1.1 syntax. The longterm step is recommended for the proper integration into a future version of the IODD specification and schema.

Information that shall be present in IODDs of this profile:

- Device supports BLOB transfer,
- Indices to be used for the BLOB transfer channel,
- BLOBs supported, either read and/or write,
- Purpose of the supported BLOBs,
- Device supports firmware update,
- SystemCommands for Bootmode,
- Profile-specific variables.

Information that shall *not* be present in IODDs of this profile:

- Content of BLOBs,
- FW-Update binaries (this is contained in the FW-Update file),
- Passwords (can be read by everone)

A.2 Binary Large Objects (BLOBs)

BLOBs are transferred via a transfer channel described by a pair of indices: One used for the identification of the BLOB and one used for the actual segmented data transfer. A Device could have more than one BLOB transfer channels for simultaneous transfer of more than one BLOB at a time. Within this profile, only one channel is defined, using the indices BLOB_ID 0x0031 (49) and BLOB_CH 0x0032 (50). These indices are reserved for profiles.

The Device adheres to this profile and supports BLOB transfer (but not necessarily FW-Update), whenever the attribute Features/@profileCharacteristic is present and contains the value 0x0030.

```
<Features ... profileCharacteristic="...0x0030 ..."/>
```

The following variable shall be inserted into the VariableCollection section:

```
<Variable index="49" accessRights="ro" id="V_BLOB_ID">
    <Datatype xsi:type="IntegerT" bitLength="16">
    (see next paragraphs for detailed explanations)
    </Datatype>
    <Name textId="TN_BLOB_ID"/>
</Variable>
```

The following shall be inserted into the ExternalTextCollection/PrimaryLanguage section:

```
    <Text id="TN_BLOB_ID" value="ID of the BLOB that is currently transferred"/>
```

NOTE 1: Text for other languages can be retrieved from the IO-Link website (www.io-link.com).

The variable "V_BLOB_ID" shall not be referenced by any Menu.

The index for BLOB_CH is not described in the IODD. The IODD syntax cannot describe complex communication protocols using a single index as transport channel. Tools shall assume the presence of BLOB_CH, whenever "V_BLOB_ID" is present.

Which BLOB IDs are supported by the Device is expressed by enumerating the allowed values of V_BLOB_ID as SingleValues within the DataType element. Negative values are used for BLOBs readable from the Device, and positive values are used for writing BLOBs to the Device. Zero shall always be defined (BLOB idle).

Example:

```
        <SingleValue value="-4096">
            <Name textId="TN_BLOB_ID_Manufacturer_Read"/>
        </SingleValue>
        <SingleValue value="0">
            <Name textId="TN_BLOB_ID_Idle"/>
        </SingleValue>
        <SingleValue value="4096">
            <Name textId="TN_BLOB_ID_Manufacturer_Write"/>
        </SingleValue>

```

The texts referenced by Name/@textId shall describe the usage of the supported BLOBs, e.g.

```
    <Text id="TN_BLOB_ID_Manufacturer_Read" value="Calibration values (read)"/>
    <Text id="TN_BLOB_ID_Idle" value="Idle, no BLOB transfer active"/>
    <Text id="TN_BLOB_ID_Manufacturer_Write" value="Calibration values (write)"/>
```

NOTE 2: Text for other languages can be retrieved from the IO-Link website (www.io-link.com).

NOTE 3: See the example IODD "IO-Link-14-BLOB-Transfer-20160225-IODD1.1.xml" at the IO-Link website (www.io-link.com).

NOTE 4: In a future version of the IODD specification, a syntax describing the index 50 (BLOB_CH) as "used for a protocol" could be specified.

A.3 FW-Update

The IODD of the Device shall only reflect the behavior of the Device in normal operation, not within the Bootload mode. A Device manufacturer could create a manufacturer-specific IODD describing the behavior in Bootload mode. However, this IODD shall only be used for in-house testing and shall not be distributed to customers.

The Device adheres to this profile and supports FW-Update, whenever the attribute Features/@profileCharacteristic is present and contains the value 0x0031.

```
        <Features … profileCharacteristic="…0x0031 …"/>
```

Furthermore, the Device shall support the SystemCommand values required for activating the Bootload mode. This can be expressed by inserting SingleValues to the V_SystemCommand variable:

```
    <StdVariableRef id="V_SystemCommand">
        <SingleValue value="80">
            <Name textId="TN_SystemCommand_BM_UNLOCK_S"/>
        </SingleValue>
        <SingleValue value="81">
            <Name textId="TN_SystemCommand_BM_UNLOCK_F"/>
        </SingleValue>
        <SingleValue value="82">
            <Name textId="TN_SystemCommand_BM_UNLOCK_T"/>
        </SingleValue>
    </StdVariableRef>
```

The SystemCommand "BM_Activate" is not available in normal operation and thus shall not be described here.

Activating the boot load mode via a sequence of writes of these values to the variable V_SystemCommand is reserved for built-in functionality of tools. The values shall not be writeable individually by the user. Therefore IODDs shall not contain Buttons for V_SystemCommand which use these values as buttonValue.

The texts referenced by Name/@textId shall be:

```
     <Text id="TN_SystemCommand_BM_UNLOCK_S" value="Start unlocking sequence"/>
     <Text id="TN_SystemCommand_BM_UNLOCK_F" value="Unlocking command 1"/>
     <Text id="TN_SystemCommand_BM_UNLOCK_T" value="Unlocking command 2"/>
```

NOTE 6: Text for other languages can be retrieved from the IO-Link website (www.io-link.com).

The following profile specific variable shall be described whenever the firmware password feature is supported:

```
    <Variable index="17341" accessRights="wo" id="V_FW-Password">
        <Datatype xsi:type="StringT" encoding="UTF-8" fixedLength="16"/>
        <Name textId="TN_FW-Password"/>
    </Variable>
```

The value of "fixedLength" is manufacturer specific.

The following two profile specific variables are mandatory:

```
    <Variable index="17342" accessRights="ro" id="V_HW_ID_Key">
        <Datatype xsi:type="StringT" encoding="UTF-8" fixedLength="16"/>
        <Name textId="TN_HW_ID_Key"/>
    </Variable>
```

The value of "fixedLength" is manufacturer specific.

```
    <Variable index="17343" accessRights="ro" id="V_BootmodeStatus">
        <Datatype xsi:type="UIntegerT" bitLength="8">
            <SingleValue value="0">
                <Name textId="TN_BootmodeStatus_Inactive"/>
            </SingleValue>
            <SingleValue value="1">
                <Name textId="TN_BootmodeStatus_Active"/>
            </SingleValue>
        </Datatype>
        <Name textId="TN_BootmodeStatus"/>
    </Variable>
```

The texts referenced by Name/@textId shall be:

```
    <Text id="TN_FW-Password" value="Firmware password"/>
    <Text id="TN_HW_ID_Key" value="Hardware Identification Key"/>
    <Text id="TN_BootmodeStatus" value="Bootmode status"/>
    <Text id="TN_BootmodeStatus_Inactive" value="Bootloader is inactive"/>
    <Text id="TN_BootmodeStatus_Active" value="Bootloader is active"/>
```

NOTE 7: Text for other languages can be retrieved from the IO-Link website (www.io-link.com).

NOTE 8: See the example IODD "IO-Link-15-Firmware-Update-20160615-IODD1.1.xml" at the IO-Link website (www.io-link.com).

A.4 IODD checker V1.1.x

When defining the intermediate part (see A.1) of the IODD definitions for this profile, care shall be taken to not violate the rules of a particular IODD Checker version.

The IODD Checker V1.1.x (x>3) allows manufacturers to use profile-specific indices and system command values within their IODD. It issues a warning that these indices or values are reserved for profiles and that further rules may apply. These profile-specific rules are currently not checked by the IODD Checker.

Annex B
(normative)
Calculation of CRC signatures

B.1 Overview of CRC-32 signatures

Hamming distance and properness for all required data lengths are important characteristics to select a particular generator polynomial.

If a generator polynomial g(x) = p(x)*(1 + x) is used, where p(x) is a primitive polynomial of degree (r – 1), then the maximum total block length is $2^{(r-1)} - 1$, and the code is able to detect single, double, triple and any odd number of errors (see [22]).

Figure 20 shows the topology to be considered for the investigations on efficiency of the CRC signature (layered transmission protocols). It should be unlikely that the CRC generator polynomial used for BLOB transfer matches the CRC generator polynomial used in the underlying transmission systems, for example IO-Link, fieldbus, or PC connection.

Table B.1 shows the characteristics of the chosen CRC-32 generator polynomial.

Table B.1 – CRC-32 generator polynomial

Polynomial		Data length (bits)	Hamming distance	Properness	Referenz	Remark
"Normal"	"Reversed"					
0x741B8CD7	0xEB31D82E	< 16360	≥ 6	n/a	[22]	~ 2 kB
		< 114663	≥ 4			~ 14 kB

NOTE Representations: "Normal": high order bit omitted, most-significant bit leftmost; "Reversed": high order bit omitted, but most-significant bit rightmost. "Reversed" allows for algorithm without branch (see Figure B.1.

B.2 Implementation considerations

B.2.1 Overview

The designer has two choices to implement the CRC signature calculation. One is based on an algorithm using XOR and bit shift operations while the other is faster using octet shifts and lookup tables.

B.2.2 Bit shift algorithm (32 bit)

For the 32-bit CRC signature, the reversed form 0xEB31D82E is used as the generator polynomial. The number of data bits may be odd or even. Figure B.1 shows the bit shift algorithm in "C" programming language.

```
uint32_t crc32_bitwise(char *data, size_t length, uint32_t previousCrc32 = 1)
{
  uint32_t crc = ~previousCrc32;
  int j;
  const uint8_t* current = (const uint8_t*) data;
  while (length-- > 0)
  {
    crc ^= *current++;
    for (j = 0; j < 8; j++)
    {
      crc = (crc >> 1) ^ (-(int32_t)(crc & 1) & 0xEB31D82E);
    }
  }
  return ~crc;
}
```

Figure B.1 – Bit shift algorithm in "C" language

The variables used in Figure B.1 are specified in Table B.2.

Table B.2 – Definition of variables used in Figure B.1

Variable	Definition
data	octet buffer containing the data to be protected
length	number of octets in data (starting with index 0) used for crc signature processing
previousCrc32	CRC signature value before new data are applied; seed value = 1
0xEB31D82E	polynomial in reversed presentation (most-significant bit rightmost)
crc	updated CRC signature value

B.2.3 Octet shift and Look-up tables (32 bit)

The corresponding function "crc32_octetwise" in "C" language is shown in Figure B.2. This function can be 20 times faster. However, the lookup table requires memory space.

```
uint32_t crc32_octetwise(uint8_t const * current, uint32_t length,
uint32_t previousCrc32 = 1, const uint32_t crc32Lookup[256])
{
    uint32_t crc = ~previousCrc32;
    while (length-- > 0)
        crc = (crc >> 8) ^ crc32Lookup[(crc & 0xFF) ^ *current++];
    return ~crc;
}
```

Figure B.2 – CRC-32 signature calculation using a lookup table

The variables used in Figure B.2 are specified in Table B.3.

Table B.3 – Definition of variables used in Figure B.2

Variable	Definition
current	octet buffer containing the data to be protected
length	number of octets in data (starting with index 0) used for crc processing
previousCrc32	CRC signature value before new data are applied
crc32Lookup	table lookup function with argument
crc	updated CRC signature value

The function in Figure B.2 uses the lookup table in Table B.4.

Table B.4 – Lookup table for CRC-32 signature calculation (hexadecimal)

CRC-32 lookup table (0 to 255)							
00000000	9695C4CA	FB4839C9	6DDDFD03	20F3C3CF	B6660705	DBBBFA06	4D2E3ECC
41E7879E	D7724354	BAAFBE57	2C3A7A9D	61144451	F781809B	9A5C7D98	0CC9B952
83CF0F3C	155ACBF6	788736F5	EE12F23F	A33CCCF3	35A90839	5874F53A	CEE131F0
C22888A2	54BD4C68	3960B16B	AFF575A1	E2DB4B6D	744E8FA7	199372A4	8F06B66E
D1FDAE25	47686AEF	2AB597EC	BC205326	F10E6DEA	679BA920	0A465423	9CD390E9
901A29BB	068FED71	6B521072	FDC7D4B8	B0E9EA74	267C2EBE	4BA1D3BD	DD341777
5232A119	C4A765D3	A97A98D0	3FEF5C1A	72C162D6	E454A61C	89895B1F	1F1C9FD5
13D52687	8540E24D	E89D1F4E	7E08DB84	3326E548	A5B32182	C86EDC81	5EFB184B
7598EC17	E30D28DD	8ED0D5DE	18451114	556B2FD8	C3FEEB12	AE231611	38B6D2DB
347F6B89	A2EAAF43	CF375240	59A2968A	148CA846	82196C8C	EFC4918F	79515545
F657E32B	60C227E1	0D1FDAE2	9B8A1E28	D6A420E4	4031E42E	2DEC192D	BB79DDE7
B7B064B5	2125A07F	4CF85D7C	DA6D99B6	9743A77A	01D663B0	6C0B9EB3	FA9E5A79

CRC-32 lookup table (0 to 255)							
A4654232	32F086F8	5F2D7BFB	C9B8BF31	849681FD	12034537	7FDEB834	E94B7CFE
E582C5AC	73170166	1ECAFC65	885F38AF	C5710663	53E4C2A9	3E393FAA	A8ACFB60
27AA4D0E	B13F89C4	DCE274C7	4A77B00D	07598EC1	91CC4A0B	FC11B708	6A8473C2
664DCA90	F0D80E5A	9D05F359	0B903793	46BE095F	D02BCD95	BDF63096	2B63F45C
EB31D82E	7DA41CE4	1079E1E7	86EC252D	CBC21BE1	5D57DF2B	308A2228	A61FE6E2
AAD65FB0	3C439B7A	519E6679	C70BA2B3	8A259C7F	1CB058B5	716DA5B6	E7F8617C
68FED712	FE6B13D8	93B6EEDB	05232A11	480D14DD	DE98D017	B3452D14	25D0E9DE
2919508C	BF8C9446	D2516945	44C4AD8F	09EA9343	9F7F5789	F2A2AA8A	64376E40
3ACC760B	AC59B2C1	C1844FC2	57118B08	1A3FB5C4	8CAA710E	E1778C0D	77E248C7
7B2BF195	EDBE355F	8063C85C	16F60C96	5BD8325A	CD4DF690	A0900B93	3605CF59
B9037937	2F96BDFD	424B40FE	D4DE8434	99F0BAF8	0F657E32	62B88331	F42D47FB
F8E4FEA9	6E713A63	03ACC760	953903AA	D8173D66	4E82F9AC	235F04AF	B5CAC065
9EA93439	083CF0F3	65E10DF0	F374C93A	BE5AF7F6	28CF333C	4512CE3F	D3870AF5
DF4EB3A7	49DB776D	24068A6E	B2934EA4	FFBD7068	6928B4A2	04F549A1	92608D6B
1D663B05	8BF3FFCF	E62E02CC	70BBC606	3D95F8CA	AB003C00	C6DDC103	504805C9
5C81BC9B	CA147851	A7C98552	315C4198	7C727F54	EAE7BB9E	873A469D	11AF8257
4F549A1C	D9C15ED6	B41CA3D5	2289671F	6FA759D3	F9329D19	94EF601A	027AA4D0
0EB31D82	9826D948	F5FB244B	636EE081	2E40DE4D	B8D51A87	D508E784	439D234E
CC9B9520	5A0E51EA	37D3ACE9	A1466823	EC6856EF	7AFD9225	17206F26	81B5ABEC
8D7C12BE	1BE9D674	76342B77	E0A1EFBD	AD8FD171	3B1A15BB	56C7E8B8	C0522C72
NOTE This table contains 32 bit values in hexadecimal representation for each value (0 to 255) of the argument a in the function crc32Lookup [a]. The table should be used in ascending order from top left (0) to bottom right (255).							

B.2.4 Seed value

The seed value shall be "1" (see *previousCrc32 = 1* in Figure B.1 and Figure B.2).

Annex C
(informative)
Compression, authentication, and encryption

C.1 Compression

C.1.1 General

The advantage of compressing data objects is the smaller size leading to a faster BLOB download via the relatively slow IO-Link transmission.

There is no need to receive the complete compressed firmware image before starting to decompress and flash it; several algorithms are available which only need read access to the already decompressed part of the firmware image, and only very little additional memory. Most of the complexity of data compression is within the compressor, which typically resides on a PC with ample resources, while the decompressor is fairly simple. Several algorithms are so simple that decompression is fast enough to keep pace with the transmission without additional delays.

Depending on the available memory (Flash and RAM) and computing power, some algorithms are more suitable than others (see [8]).

The principle of data compression (applies for both lossy and lossless compression)

The compressor has a model of the data. This model is used to predict the next few bytes to be compressed. The incoming data is compared with the prediction, and only the difference (the “surprise”) needs to be stored. The better the model, the better the prediction, and the better the data can be compressed. The model can be predefined /static, if the kind of data to be compressed is known in advance. But most of the time, the model is derived from the already processed data, and dynamically updated. So the compressor does an extrapolation of the already received data to predict the next data. The difference to the prediction is then coded in a compact form. For example with the "Deflate" algorithm (C.1.3), the model is "repeated sequences of byte-strings" (Lempel-Ziv) and Huffman coding.

The decompressor also has the model, which it updates the same way the compressor does, from the already decompressed data. Step-by-step, it generates a prediction and then decodes the compressed stream and applies the differences to the prediction to regenerate the bytes of the original uncompressed stream.

The following clauses C.1.2 to C.1.4 contain some suggestions for compression algorithms, sorted in increasing requirements for memory and computing power.

C.1.2 LZRW (Lempel-Ziv-Ross Williams)

This family of algorithms "LZRW" was invented by Ross Neil Williams (see [9], [10] and Dr. Ross’s compression crypt in [11] with source code). For example, the decompressor for LZRW1 is 22 lines of "C" code, the while loop within is 12 lines of code. Apart from the source and destination pointers, only a few local variables are required.

C.1.3 Deflate (Lempel-Ziv 77 + Huffman Coding)

The "Deflate" algorithm is explained in [12] and specified in RFC 1951 [13]. It was invented by Phil Katz for PKZIP (see [14]). Source code is available for the zlib compression library [15]. If only one stream has to be compressed, the ZLIB format can be used (see RFC 1950 in [16]). For combining multiple streams into one compressed stream, the gzip format can be used (see RFC 1952 in [17]).

"Deflate" has a better compression than "LZRW". However, it is more complex and slower.

C.1.4 LZO (Lempel-Ziv-Oberhumer)

The "LZO" algorithm was invented by Markus F.X.J. Oberhumer and is explained in [18]. Source code is available in [19].

"LZO" is much faster than "Deflate", but more complex and requires a buffer of 8 kilobytes (or 64 kilobytes) for the decompressor.

C.2 Authentication via signatures

Adding a CRC signature to the binary code is a very effective measure against *unintended* modification (random corruption) of the firmware image. It allows detection of firmware alterations ever since the CRC signature was last calculated. However, since the CRC algorithm is specified and well-known, anyone can maliciously alter the data and recalculate the CRC signature such that the checks return OK.

A more sophisticated measure is required to secure the firmware against *intended* modifications, for example tampering, hacking, or malicious attacks. In this case, a cryptographic signature should be added to ensure authenticity of the firmware image without modification other than by the manufacturer.

How to create a cryptographic signature?

In a first step, the data stream to be secured can be optionally transformed to a "normalized" form. For example, if the data stream is known to be XML code, white space and comments would be filtered out and numbers would be normalized by removing leading zeros, trailing zeroes, and space characters.

In a second step, the transformed data stream is processed using a cryptographic hash algorithm. A property of hash algorithms is the ease of computing a hash value for a given stream of data. However, it is nearly impossible to generate another stream of data producing the same hash.

In a third step, an asymmetric encryption algorithm is used. Asymmetric means, it uses a pair of keys, a private one and and a public one. What has been encrypted with one key can only be decrypted with the other key. It is nearly impossible to decrypt a data stream only knowing the key used for encryption. Thus, the computed hash value from the second step is now encrypted with the private key of the signer.

The overall cryptographic signature attached to the data stream finally contains the hash value, the encrypted hash value, the public key of the signer, and meta information about the used algorithms for the three steps, including meta information about the purpose of the signature.

How to check the signature?

The receiver performs the same transformation and processing of the hash value of the data stream with the algorithms indicated within the signature. The check fails if the hash value deviates from the one stored in the signature.

Decryption of the encrypted hash value contained in the signature can be performed using the public key stored in the signature, and using the encryption algorithm indicated in the signature. The check fails if the result deviates from the hash value within the signature.

In case of passed checks, the receiver knows that the data hasn't been changed ever since someone with this public key signed it. However, the authenticity of this public key is not yet known. Thus, the public key stored in the signature shall be verified. Either, the public key has already been distributed via a secure channel and is therefore well-known, or some secure protocol is used to check the key online. The online check is also useful in cases where the key pair had been withdrawn due to a compromised private key.

An overview of algorithms and advice, which one to use and which one not to use, can be retrieved from [20].

In the context of FW-Update, the signature would be created by the Device manufacturer and added to the firmware image prior to its release to the customer. After downloading the firmware to the Device, the bootloader would check the signature and deny the activation of the firmware if not valid.

Conclusions?

e) A transformation step in case of a firmware image is not necessary.

f) There are no special requirements for the hash algorithm and the asymmetric encryption algorithm. It may be sufficient for boot-loader to support only one fixed hash and encryption algorithm due to restricted memory space. Thus, meta information about algorithms in use can also be omitted.

g) It may also be sufficient for boot-loader to contain the public key of the manufacturer for a comparison:
- No possibility for an online check
- The key is not a secret
- If the key in the boot-loader can be altered, the downloaded firmware can be altered also

h) There is no provision for the withdrawal of a key pair in case of a compromised private key since the public key is built-in. Therefore, the key pair used for the authenticity of firmware images should not be used for any other purposes. It can be reasonable using separate key pairs for each Device type or Device family to limit the damage.

i) Recommendation is to use authentication only in case of a threat of inofficial or tampered firmware.

C.3 Encryption of FW-Update files

Encryption of FW-Update files can be desirable to prevent from an analysis of a Device's function. Algorithms can be found in [20].

However, it is recommended not to use encryption since the Device requires decrypted code to perform its intended functions. The key for decryption must be permanently available within the Device and thus would be accessible for an intruder.

Annex D
(informative)
Performance estimations

D.1 Assumptions

Clause 6.4.1 describes how the ISDU mechanism of IO-Link is used for the transfer of BLOBs and in particular of FW-Updatebinaries which can be one up to several 100*10^3 octets. The time required for a BLOB transfer via ISDU dominates the duration of a FW-Update provided the time of an upperlevel fieldbus system is negligible. This estimation focuses only on IO-Link specific parameters. All other related parameters are not considered. This means that the estimations in this Annex E are considerably shorter than in reality.

The length of a binary large object (BLOB) to be transferred between a Tool and a Device is called $Length_{\mathrm{BLOB}}$ (see 6.5.3.1). The payload of an ISDU can be in this case up to 231 octets ("ExtLength"). However, a particular Device defines its maximum ISDU payload via a parameter and therefore a BLOB is usually transferred in a segmented manner using the data type OctetStringT as illustrated in Figure 7.

Segment by segment of the BLOB is placed as OctetStringT into the BLOB data area of the Write-request of the Master and transferred via the On-request Data communication channel using an appropriate M-sequence type. The Device returns its positive response also via the On-request Data communication channel (see [1]).

D.2 Estimated time for a BLOB transfer

The time required for a BLOB transfer is given by

– the number of ISDUs "N_{ISDU}", multiplied by

– the number of M-sequences "N_{Mseq}" required to transfer one ISDU, and multiplied by

– the cycle-time "t_{CYC}" for one ISDU transfer.

The result of this performance estimation is shown in equation (1) that allows evaluating the duration of a BLOB transfer "t_{BLOB}".

$$t_{BLOB} = N_{ISDU} \times N_{Mseq} \times t_{CYC} \qquad (1)$$

D.3 Required number of ISDU transfers

The length of the BLOB, divided by the length of the OctetStringT (decremented by the header), and rounded up to the next integer value determines the number "N_{ISDU}" of ISDUs required for a BLOB transfer as shown in equation (2).

$$N_{ISDU} = \left\lceil \frac{Length_{BLOB}}{Length_{OctetStringT} - 1} \right\rceil \qquad (2)$$

D.4 Number of M-sequences per ISDU

ISDUs are cyclically transferred in OPERATE mode using the On-request Data (OD) channel. Its capacity depends on the M-sequence type. TYPE_1_V with 8 or 32 octets of OD are recommended for the BLOB transfer in case of firmware update. For the number of M-sequences "N_{Mseq}" per ISDU transfer, the number of M-sequences for both the write request and the read request are required (see Figure 7).

For the write request this number is "ExtLength" of the ISDU (length of OctetStringT + 4) divided by the octets of OD (8 or 32). For the read request this number is 2 divided by the octets of OD (8 or 32). The quotients are rounded up to the next integer value and summarized as shown in equation (3).

$$N_{Mseq} = \left\lceil \frac{ExtLength}{OD} \right\rceil + \left\lceil \frac{2}{OD} \right\rceil \qquad (3)$$

Table E.1 lists the number of M-sequences required for various lengths of OctetStringT values and for different M-sequence types.

Table E.1 – Number of M-sequences per ISDU

Length of OctetStringT	TYPE_0 (OD = 1)	TYPE_1_V (OD = 8)	TYPE_1_V (OD = 32)
28	34	5	2
60	66	9	3
92	98	13	4
124	130	17	5
156	162	21	6
188	194	25	7
220	226	29	8

D.5 Recommended Master cycle times

The duration of an M-sequence depends on the transmission time of UART frames and certain delay times as shown in [1], Annex A.3.6. The (Master) cycle time cannot be shorter than the M-sequence time since there is an idle time between two M-sequences.

The estimation assumes a negligible idle time and the following parameter values for equation A.6 in [1]:

– m: Number of UART frames sent to the Device (corresponds to OD)

– n: Number of UART frames returned by the Device (corresponds to OD)

– t_A: Delay between end of stop bit of last UART frame sent and begin of start bit of first UART frame received (assumed value: 10 T_{BIT})

– t_1: Delay between two UART frames sent (assumed value: 1 T_{BIT})

– t_2: Delay between two UART frames received (assumed value: 3 T_{BIT})

Table E.2 shows recommended minimum cycle times with repect to transmission rates and M-sequence types.

Table E.2 – Recommended cycle times (t_{CYC})

Transmission rate (bit/s)	M-sequence types		
	TYPE_0 (OD = 1)	TYPE_1_V (OD = 8)	TYPE_1_V (OD = 32)
COM1	16,0 ms	33,6 ms	94,4 ms
COM2	2,9 ms	5,1 ms	12,8 ms
COM3	1,4 ms	1,7 ms	3,0 ms
NOTE The timings have been extended by 1 ms due to internal Master processing times.			

D.6 Conclusions

Annexes D.3 to D.5 provide the necessary values for equation (1) in Annex D.2 to calculate the timings for BLOB transfers.

Figure E.1 and Figure E.2 show the minimum timings for COM3 and COM2 transmissions at various configurations. "OD 8" refers to an M-sequence type with 8 octets of M-sequence data; "OctetStringT 220" refers to an OctetStringT length of 220 octets.

At a COM3 transmission rate, 250 000 octets require a minimum of 20 to 35 seconds.

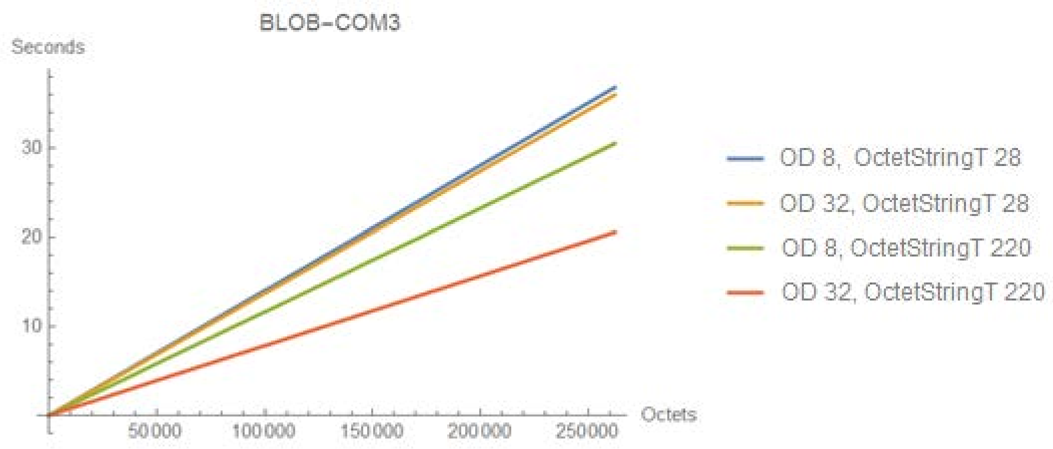


Figure E.1 – BLOB transfer timings for COM3

At a COM2 transmission rate, 30 000 octets require a minimum of 12 to 22 seconds.

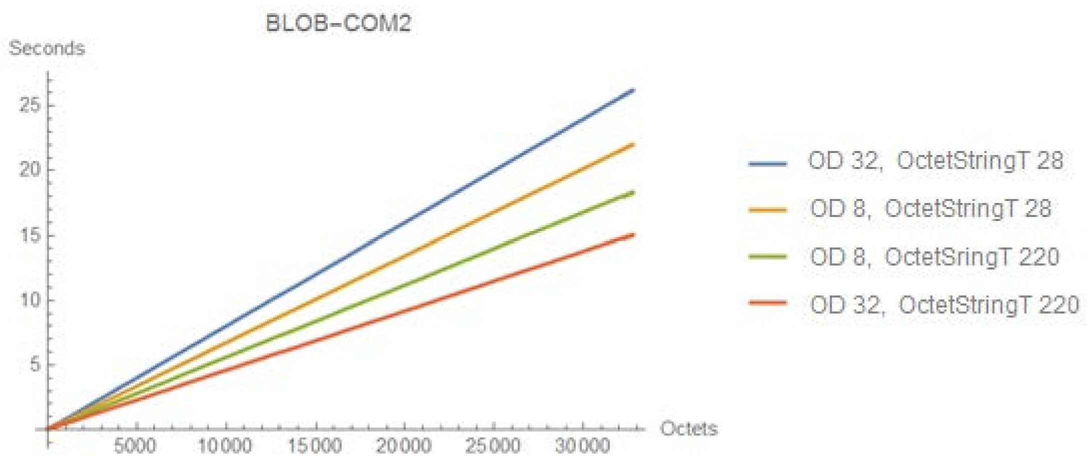


Figure E.2 – BLOB transfer timings for COM2

As expected, it turns out that the performance be best with large OctetStringT lengths and large On-request Data communication channels.

However, if for some reasons a shorter OctetStringT needs to be chosen (e.g. because of RAM limitations in the Device), it is preferable for COM2 Devices to choose an M-sequence type with 8 octets OD. It should be noted that selecting an M-sequence type with only 1 octet OD slows down the transfer speed by approximately a factor of 8. The timings provided in the diagrams shall be considered as lower limit. In real implementations, additional delays for transfer and storage shall be expected that can increase these timings by factors.

In case the Device needs the segment to be flashed immediately, a "busy" message is sent. "N_{Mseq}" increases by a rounded up $t_{\mathrm{flash}}/t_{\mathrm{CYC}}$. "$t_{\mathrm{BLOB}}$" then follows equation (4):

$$t_{BLOB} = t_{CYC} \times \left\lceil \frac{Length_{BLOB}}{Length_{OctetString} - 1} \right\rceil \times \left(\left\lceil \frac{ExtLength}{OD} \right\rceil + \left\lceil \frac{2}{OD} \right\rceil + \left\lceil \frac{t_{flash}}{t_{CYC}} \right\rceil \right) \qquad (4)$$

With typical flash (erase) times of 40 ms, t_{BLOB} increases by 275 % for COM3 and by 50 % for COM2.

Annex E
(normative)
Profile specific test cases and tools

E.1 Overview

The profile specific test cases and tools will be defined at a later state of the project, when implementations are more advanced and possible consequences can be overlooked. However, for the sake of sufficient quality in the field, this task shall be started by end of 2016 at the latest.

E.2 Binary Large Objects (BLOBs)

tbd.

E.3 Firmware-update

tbd.

E.4 Test tools

E.4.1 Requirements for a Device tester

tbd.

E.4.2 Requirements for a BLOB FB tester

tbd.

E.4.3 Requirements for an Update tool tester

tbd.

E.4.4 Requirements for an IODD Checker dedicated to the BLOB & FW-Update profile

In a future IODD Checker version, covering the particular requirements for the BLOB & FW-Update profile, the following checks could be added.

For BLOB transfer, detected by Features/@profileCharacteristic containing "0x0030":

- A variable with Index 49,
- Which is present and has the expected Id, Name/@textId and DataType,
- And which has no ValueRange assigned.
- The SingleValues are within the allowed ranges.
- The SingleValue with value "0" is present and has a Name with the expected textId.
- A variable with index 50 is not present.
- The texts referenced by the above mentioned textId shall be as expected.

If BLOB transfer is not supported:

- A variable with Index 49 or 50 is not present.

For FW-Update, detected by Features/@profileCharacteristic containing "0x0031":

- The SystemCommand variable,
- Which is referenced,
- And which has SingleValues for values 80 to 82, each one having Name with the expected textId.

- It does not have SingleValue for value 83.
- A variable with index 17341, when present, has the expected Id, Name/@textId and DataType.
- A variable with index 17342,
- Which is present, has the expected Id, Name/@textId and DataType.
- A variable with index 17343,
- Which is present, has the expected Id, Name/@textId and DataType,
- And has SingleValues for the values "0" and "1", each one having Name with the expected textId.
- The texts referenced by the above mentioned textId shall be as expected.

If FW-Update is not supported:

- The SystemCommand variable shall not contain SingleValues with values in the range 80 to 83.
- A variable with Index 17341 to 17343 is not present.

If the Checker detects and checks BLOB transfer and FW-Update, the general warning regarding use of profile-specific Indices/values shall be suppressed.

E.5 Manufacturer declaration

tbd.

Annex F
(informative)
Information on conformity testing of profile Devices

Information about testing profile Devices for conformity can be obtained from the following organization:

IO-Link Community
Haid-und-Neu-Str. 7
76131 Karlsruhe
Germany
Phone: +49 (0) 721 / 96 58 590
Fax: +49 (0) 721 / 96 58 589
E-mail: info@io-link.com
Web site: http://www.io-link.com

Bibliography

[1] IO-Link Community, *IO-Link Interface and System*, V1.1.2, July 2013, Order No. 10.002 or
IEC 61131-9, *Programmable controllers – Part 9: Single-drop digital communication interface for small sensors and actuators (SDCI)*

[2] IO-Link Community, *IO Device Description (IODD)*, V1.1, July 2011, Order No. 10.012

[3] IEC/TR 62390:2005, *Common automation device profile guideline*

[4] IEC 60050 (all parts), International Electrotechnical Vocabulary

NOTE See also the IEC Multilingual Dictionary – Electricity, Electronics and Telecommunications (available on CD-ROM and at <http://domino.iec.ch/iev>).

[5] IO-Link Community, *IO-Link Communication*, V1.0, January 2009, Order No. 10.002

[6] IO-Link Community, *IO-Link Test Specification*, V1.1.2, July 2014, Order No. 10.032

[7] IO-Link Community, *IO-Link Smart Sensor Profile*, V1.0, October 2011, Order No. 10.042

[8] http://www.faqs.org/faqs/compression-faq/

[9] https://en.wikipedia.org/wiki/Ross_Williams_%28computer_scientist%29

[10] https://en.wikipedia.org/wiki/LZRW

[11] http://www.ross.net/compression/

[12] https://en.wikipedia.org/wiki/DEFLATE

[13] https://tools.ietf.org/html/rfc1951

[14] https://en.wikipedia.org/wiki/Phil_Katz

[15] https://en.wikipedia.org/wiki/Zlib

[16] https://tools.ietf.org/html/rfc1950

[17] https://tools.ietf.org/html/rfc1952

[18] https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Oberhumer

[19] http://www.oberhumer.com/opensource/lzo/

[20] http://www.internet-sicherheit.de/fileadmin/docs/service/tipps_zur_internet-sicherheit/sicherheit_von_verschluesselungsalgorithmen/2014-05-26_Cryptographic_Algorithms_A4.pdf

[21] https://en.wikipedia.org/wiki/Cyclic_redundancy_check

[22] Koopman, P., *32-Bit Cyclic Redundancy Codes for Internet Applications,* The International Conference on Dependable Systems and Networks (DSN), 2002

[23] Castagnoli, G., Braeuer, S. & Herrman, M., *Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits*, IEEE Transactions on Communications, Vol. 41, No. 6,June 1993.

[24] http://create.stephan-brumme.com/crc32/

____

IO-Link