FDT®3.0

Technical Specification

Version 1.00.00

This specification is the intellectual property (IP) of the FDT Group, AISBL. Copyright 2020 by the FDT Group AISBL. All rights reserved. No portions nor the totality of this specification may be reproduced in any form or medium nor further distributed in any form or medium without the express written permission of the FDT Group.

FDT Group – Document No. <20-0001>

Publisher:

FDT Group AISBL
5 Industrieweg • 3001 Heverlee • Belgium

www.fdtgroup.org
info@fdtgroup.org

This specification is the intellectual property (IP) of the FDT Group, AISBL. Copyright 2020 by the FDT Group AISBL. All rights reserved. No portions nor the totality of this specification may be reproduced in any form or medium nor further distributed in any form or medium without the express written permission of the FDT Group.

The possession of this specification does not, by itself, convey any right to use or reproduce any portion of the specification or to make or have made any products or services contemplated, suggested or enabled by the specification. You are hereby notified that such products and services may be covered by valid patents or copyrights of the FDT Group, its members or other licensors.

The necessary nonexclusive licenses to use or reproduce portions of the specification to make or, have made such products or services may be obtained only from the FDT Group. Contact the FDT Group at info@fdtgroup.org for more information and to secure the necessary licenses and other applicable artifacts.

The FDT Group makes no warranty or assurances as to the completeness, fitness, inerrancy, suitability or efficaciousness of this standard for any geopolitical area, market segment, network, protocol, application, product or service.

The FDT Group reserves the right to modify, enhance, abbreviate, abridge, consolidate or withdraw this standard at any time without further notification.

“FDT” and the FDT Group logo are registered trademarks of the FDT Group, AISBL.

History

Rev.	Author	Change Description	Date
1.00.00	WG Architecture & Specification	Public Release FDT3.0	2020-06-04

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

CONTENTS

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

INTRODUCTION

This document is an interface specification for developers of FDT (Field Device Tool) components for function control and data access within a client/server architecture. The specification is a result of an analysis and design process to develop standard interfaces to facilitate the development of servers and clients by multiple vendors that need to interoperate seamlessly.

With the integration of fieldbuses into control systems, there are a few other tasks which need to be performed. In addition to fieldbus-specific and device-specific tools, there is a need to integrate these tools into higher-level system-wide planning tools or engineering tools. In particular, for use in extensive and heterogeneous control systems, the unambiguous definition of engineering interfaces that are easy to use for all those involved is of great importance.

A device-specific software component, called DTM (Device Type Manager), is supplied by the field device manufacturer with its device. The DTM is integrated into engineering tools via the FDT interfaces defined in this specification. The approach to integration, in general, is open for all kind of fieldbuses and thus meets the requirements for integrating different kinds of devices into heterogeneous control systems.

Figure 1 shows how FDT3 Technical specification is related to the IEC 62453 series.

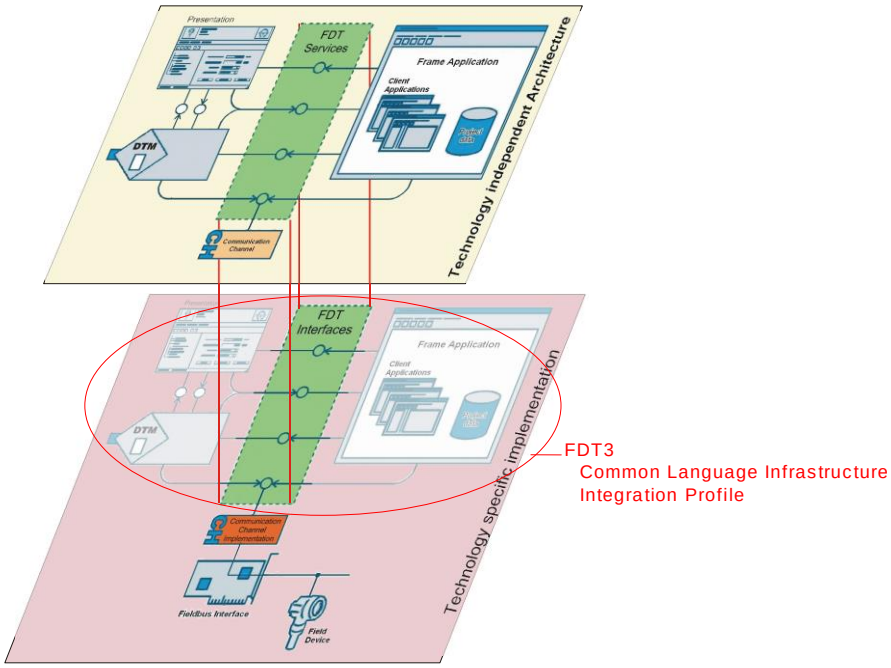


Figure 1 — Relation of FDT3 to the IEC 62453 series

The document structure is:

- Chapter 3 explains the used terms, definitions and conventions
- Chapter 4 introduces the general concepts of FDT3.0
- Chapter 5 describes the technical concepts used to implement FDT3.0 and how FDT concepts are mapped to .NET Standard
- Chapter 6 provides an overview of the FDT Objects, their interfaces and behavior
- Chapter 7 presents an overview of the FDT3.0 datatypes
- Chapter 8 shows the interaction of FDT Objects at runtime

- Chapter 9 explains rules related to installation and deployment of DTMs
- Chapter 10 explains how FDT life cycle concepts are implemented
- Chapter 11 shows examples for Frame Application architectures

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

1 Scope

This document defines how the common FDT principles are implemented based on the Microsoft .NET technology and web technologies for graphical user interfaces The specification includes the object behavior and object interaction via .NET interfaces and JavaScript APIs. Emphasis has been placed on support of distributed Frame Application architectures.

This document specifies FDT version 3.0.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

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 61784 (all parts), *Industrial communication networks – Profiles*

IEC 62453-1:2009, *Field Device Tool (FDT) interface specification – Part 1: Overview and guidance*

IEC 62453-2:2009, *Field Device Tool (FDT) interface specification – Part 2: Concepts and detailed description*

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

3 Terms, definitions, symbols, abbreviated terms and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 62453-1, IEC 62453-2, MSDN® and the following apply.

3.1.1
action
execution of a function which may involve several calls to interface methods of different FDT Objects

3.1.2
asynchronous methods
methods that trigger execution of asynchronous operations

Note: See also 5.8.7

3.1.3
asynchronous operation
operation that is performed while the FDT object (client) that has requested the operation does not wait for the result, but the client is notified when the operation is finished

3.1.4
bulk data
device-node-specific persisted data, which is stored besides DTM instance data

Note: Example for bulk data: accumulated historical data, used for trend analysis.

3.1.5
bulk operation
operation to perform one or more tasks at a group of devices nodes

Note: Examples for bulk operation: up- or download for a group of devices, parameter adjustment for a group of devices or report generation for a group of devices

3.1.6
Child DTM
DTM instance in an FDT Project, which is classified by its relation to a Parent DTM

Note: Any DTM which uses FDT communication may be classified as Child DTM (i.e. Device DTM, Gateway DTM, Module DTM and BTM)

3.1.7
clone DTM instance
process of creating a new device node in the FDT topology based on an existing device node (includes copying DTM instance(see 3.1.11) and resetting device-node-specific DTM data)

Note: The identification attributes of the device are changed.

3.1.8
Communication Channel
component representing access to a fieldbus segment or to other means of communication

3.1.9
compatibility

a) feature of a component (hardware or software) that enables it to be interoperable with another component.

b) feature of a component (hardware or software) that enables it to replace another version of the component

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

3.1.10
compatibility attributes
attributes used to find compatible components, to replace components or to validate compatibility after a component replacement. Compatibility attributes are required to check whether a component is compatible with another component.

Note: Compatibility attributes are used to define compatibility in regard to meaning b) of compatibility.

3.1.11
copy DTM instance
process of creating a new device node in the FDT topology based on an existing device node (includes loading the original DTM dataset to initiate the new DTM instance)

Note: The identification attributes of the device are not changed.

3.1.12
copy device node in FDT Topology
(see 3.1.11 copy DTM instance)

3.1.13
DataTransaction
a transaction regarding the data of a DTM (persistent or device data)

3.1.14
delegate
a reference type that can be used to encapsulate a method

3.1.15
device configuration
process of changing data related to device-specific characteristics/basic behavior

Note: Example for such characteristics may be the structure of a remote I/O or the type of measurement procedure like absolute pressure or differential pressure

3.1.16
device data
configuration data that resides on the device

3.1.17
device parameterization
process of changing device-specific data in order to adjust application-specific behavior

3.1.18
device node
node in the topology, which represents a device

Note: A DtmDeviceType is assigned to a device node, which is instantiated to operate the device in online or offline modes. See Figure 25

3.1.19
device type check
process of checking the device type when a DTM is going online with a connected physical device

Note: The DTM shall reject to go online if the connected physical device type is not supported. The check shall be based on same information as in DeviceIdentInfo. See 6.3.2.3

3.1.20
DD
device description

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

3.1.21
DD-Interpreter DTMs
DTMs which interpret device descriptions at runtime.

3.1.22
DTM
software component containing device-specific application software, including DTM Business Logic, DTM User Interface and related objects (e.g. Communication Channel)

Note: Older FDT specification documents used the term "DTM" for the object DTM Business Logic as well as for the whole component consisting of DTM BL, DTM UI and channels.

3.1.23
DTM Business Logic (DTM BL)
part of the DTM, which contains all the functionality to access storage and communication and which manages the instance data of a DTM

3.1.24
DTM Identifier
identifier, which is used to identify a DTM (DTM BL class)

Note: In order to uniquely identify a DTM BL class, the property DtmInfo.Id is used.

3.1.25
DTM instance modal
prevents user interaction with other windows of the DTM instance (see modal window)

3.1.26
DTM User Interface (DTM UI)
part of the DTM, which is displayed to a human user

3.1.27
DtmType
either DtmDeviceType, DtmModuleType or DtmBlockType. All DtmTypes provide identification in DtmInfo class.

Note: DtmType is described by TypeInfo.

3.1.28
DtmDeviceType
element of a DTM software supporting one or more device types.

Note: DtmDeviceType is described by DeviceTypeInfo class.

3.1.29
DtmModuleType
element of a DTM software supporting one or more device module types.

Note: DtmModuleType is described by class ModuleTypeInfo.

3.1.30
DtmBlockType
element of a DTM software supporting one or more block types.

Note: DtmBlockType is described by class BlockTypeInfo.

3.1.31
FDT Object
object defined by FDT (e.g. Frame Application, DTM Business Logic, DTM User Interface, Communication Channel)

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

3.1.32
FDT Protocol Annex
document defining support for a communication protocol for FDT

Note: Examples for such documents are “PROFIBUS protocol annex” and “HART protocol annex”. Within IEC 62453 the standard parts with numbers 3xy define support for communication protocols.

3.1.33
FDT Application Profile Annex
document defining support for a type of application for FDT

Note: An example for such a document is the “PLC Tool interface” (defined for FDT1.2). Other such documents may be defined at a later time also for FDT3.

3.1.34
fieldbus message
data in a protocol-specific telegram

3.1.35
Frame Application modal
prevents user interaction with windows of the Frame Application (see modal window)

3.1.36
hardware platform
hardware on which FDT software is executed

Note: Different hardware platforms may be supported that are based on different architectures and display formats, for example PC and others

3.1.37
identification attributes
attributes which describe the identity of a component. These attributes are typically displayed to users or used to validate and ensure compatibility of components.

3.1.38
incompatibility
situation where a component is not interoperable or where a component cannot replace another component

3.1.39
instance data
configuration data that resides in the DTM instance

3.1.40
lifetime of DTM instance
time span of executing a DTM BL (from state ‘created’ till state ‘released’)

3.1.41
Link
logical relation of a DTM to a physical device (not the communication connection)

3.1.42
modal window
prevents user interaction with all windows of the process

3.1.43
monolithic DTM
a single DTM that represents the complete device with all its modules

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

NOTE There are also other concepts representing modules of a device in this specification, for example Module DTM and BTM

3.1.44
online data
configuration data that resides on the device and can be accessed by communicating with the device

NOTE: online data may be a subset of device data (i.e. it may be that not all device data is accessible by communicating with the device)

3.1.45
operation
procedure that may involve one or more method calls between FDT Objects

3.1.46
Parent DTM
DTM instance in an FDT Project, which is classified by its relation to a Child DTM

Note: any DTM which provides FDT communication may be classified as Parent DTM (e.g. Communication DTM, Gateway DTM, Composite Device DTM)

3.1.47
passive device
device that does not communicate

3.1.48
pattern
a standard solution to common problems in software design

3.1.49
platform
combination of hardware platform, target platform and target CLR, that defines the environment for execution of FDT software

3.1.50
project
a generic term for the sum of information related to a set of devices

Note: The definition of project is specific for a Frame Application.

3.1.51
proxy object
an object which functions as a representative of another object

Note1: The proxy pattern is often used as software design pattern.

Note2: FDT3 uses the proxy pattern for interaction between DTM BL and DTM UI, between DTM BLs, between DTM BL and Communication Channels, for supporting multiple CLI implementations and versions in a Frame Application and for providing backward compatibility to FDT1.2.x[1][2] and FDT2[3].

3.1.52
reassign
Change the TypeInfo assigned to a device node

a) from one TypeInfo.Id to the same TypeInfo.Id in a different DTM

b) change the TypeInfo.Id to another TypeInfo.Id within the same DTM or to another DTM

3.1.53
reassignment
process of assigning a different DtmType to a device node with assigned DtmType

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Note: It is possible that for the previously assigned DtmType already a dataset exists. This dataset should be considered in the reassignment.

3.1.54
replacing installation
installation of a new version of a DTM which replaces a currently installed DTM version

Note: A Frame Application is notified about the installation, but the DtmDeviceTypes do not need to be reassigned. DTM Updates (see update) and DTM Upgrades (see upgrade) replace installations of older versions of the DTMs.

3.1.55
revision
identification of modification of non-FDT components, e.g. device firmware or device hardware

Note1: Not all fieldbus specifications supported by FDT and/or device types provide a version identification which allows to derive compatibility statements.

Note2: In contrast to a version, revisions require additional fieldbus-specific or device-type-specific knowledge to derive compatibility or interchangeability predictions.

3.1.56
SemanticInfo
identifier that provides a reference to semantics defined in a specific context

Note1: The reference is provided by the SemanticId, the context is provided by the ApplicationDomain that accompanies the SemanticId

Note2: There may be several semantics provided for an information item, e.g. a parameter may be described in a fieldbus profile as well as in a device profile (e.g. for drives), that is why several SemanticInfo entries may be provided for an information item.

3.1.57
set point
target value that an automatic control system will aim to reach

Note: For example a boiler control system may have a temperature set point, which is the temperature the control system aims to attain

3.1.58
Sibling DTMs
DTM instances in an FDT Project, which are classified by their relation to the same Communication Channel

3.1.59
surrogate process
a process hosting an object on behalf of client processes

Note: A surrogate process can have other qualities than the client process. E.g. it can be used to load different CLI implementations (e.g. .NET Core, .NET Framework) and versions.

3.1.60
synchronous operation
operation that is performed while the object that requested the operation is waiting for the result

3.1.61
target CLR
common language runtime, which defines the environment for execution of FDT software

Note: An example for target CLR is the CLR 4.0.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

3.1.62
target platform
the native data size supported by the machine and operation system, on which the FDT software is executed

3.1.63
update
process to replace a component with a later (up to date) revision (update revision) that includes error corrections.

3.1.64
update revision
(minor) revision of a component that includes error corrections and small enhancements

Note: In comparison to an upgrade revision an update revision includes no major functional enhancements or new features . An Update Revision shall be backwards compatible to previous revisions of the same component.

3.1.65
upgrade
process to replace a component with a later revision that includes functional enhancements and/or new features (upgrade revision).

3.1.66
upgrade revision
revision of a component that includes functional enhancements and/or new features compared to a previous revision of the component

Note: An Upgrade Revision shall be backwards compatible to previous revisions of the same component.

3.1.67
version
an instance of a software product derived by modification or correction of a preceding software product instance (see [28]).

Note1: The format of a version is: Major.Minor[.release[.build]] for more information see Annex D.

Note2: Version is used in FDT3 for identification of FDT software components and for corresponding compatibility attributes.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

3.2 Abbreviations

For the purpose of this document, the abbreviations given in IEC 62453-1, IEC 62453-2 and the following apply.

API	Application Programming Interface
BL	Business Logic
BTM	Block Type Manager
CLI	Common Language Infrastructure
CLR	Common Language Runtime
CLS	Common Language Specification
CSS	Cascading Style Sheet
DCS	Distributed Control System
DLL	Dynamic Link Library
DTM	Device Type Manager
FA	Frame Application
FDT®	Field Device Tool
GUI	Graphical User Interface
GUID	Globally Unique Identifier (a UUID)
HART®	Highway Addressable Remote Transducer
HTML	Hyper Text Markup Language
IO	Input / Output
LCID	Locale ID
MIME	Multipurpose Internet Mail Extensions
MSDN®	Microsoft Developer Network
OPC	Open Packaging Convention
OPC UA	Open Platform Communications Unified Architecture
PLC	Programmable Logic Controller
RID	Runtime id
UI	User Interface
WPF	Windows Presentation Foundation
XDR	XML data reduced
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformations

3.3 Conventions

The conventions for the UML notation used in this document are defined in Annex I.

This document specifies requirements to software. Different levels of requirement may be recognized by the used wording.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Wording	Indicates
'shall', 'has to', 'have to' ,or 'Mandatory'	No exceptions allowed.
'should' or 'Recommended'	Strong recommendation. It may make sense in special exceptional cases to differ from the described behavior.
'conditional'	Function or behavior shall be provided, depending on defined conditions.
'can' or 'Optional'	Function or behavior may be provided.

Further conventions are:

Convention	Indicates
Note:	Indicates text (in small letters), which does not express requirements, but provides additional information.
<MethodName>	Angle brackets are used to indicate a reference to an asynchronous method Note: If looking for definition in Annex B: Such methods are implemented as pair of BeginMethodName()/EndMethodName()
Editor note	Magenta text in a green frame is used to provide editorial information. This information will disappear in the final version of the document (e.g. by replacement)

Code examples provided in this document are intended for illustration of the described concepts. They should not be used as is. Developers of FDT software should consider where the developed code is applied and design the software accordingly. For exact specification of protocol-specific implementations, refer to the FDT Protocol Annex documents.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

4 Concepts of FDT

4.1 Introduction

This section defines the general concepts of FDT3.0. The object model, use cases and general solution approaches are explained.

FDT3.0 is an evolution of FDT, based on FDT2.1. For a comparison of FDT3.0 with FDT2.1 refer to Annex G.

NOTE 1: FDT2 is used as designator applying to both FDT2.0, and FDT2.1.

NOTE 2: FDT3 is used as designator applying to FDT3.0.

4.2 FDT object model

Figure 2 provides an overview of how the FDT Objects (defined in IEC 62453-2) are implemented in FDT3 and how their relationship to each other is implemented. The FDT Objects are implemented as .NET objects. WebUIs, as a specific type of User Interface, are implemented based on HTML and JavaScript. See chapter 5.2 and 5.3 for the supported HTML and JavaScript versions respectively.

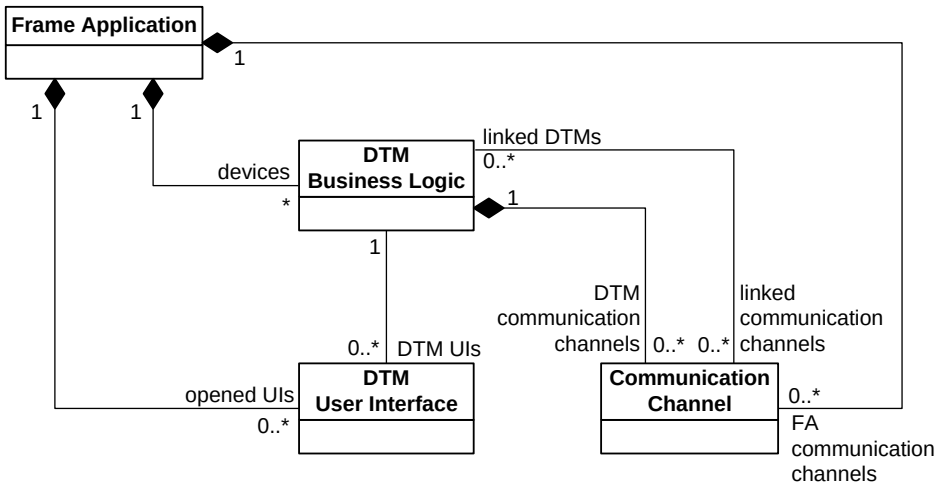


Figure 2 — FDT3 Object Model

If the Frame Application is a distributed software system, the Frame Application is responsible to organize the instantiation of the objects (based on a vendor-specific implementation).

4.3 FDT Frame Application (FA)

4.3.1 General

Frame Applications are able to handle any type of device by integrating corresponding DTMs without the need for device or fieldbus-specific knowledge. Depending on the intended use Frame Applications may provide different appearance and features (e.g. standalone configuration or engineering system of a DCS). Typically, the Frame Application comprises client applications focusing on specific aspects such as configuration, observation, IO planning, and using the services provided by the DTMs.

A Frame Application is the runtime environment for the DTMs and provides interfaces which enable the DTM Business Logic and the DTM User Interfaces to interact with its environment. In addition, the Frame Application manages the interaction between the DTM Business Logic and the DTM User Interface by providing a standard messaging interface (see Figure 3).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

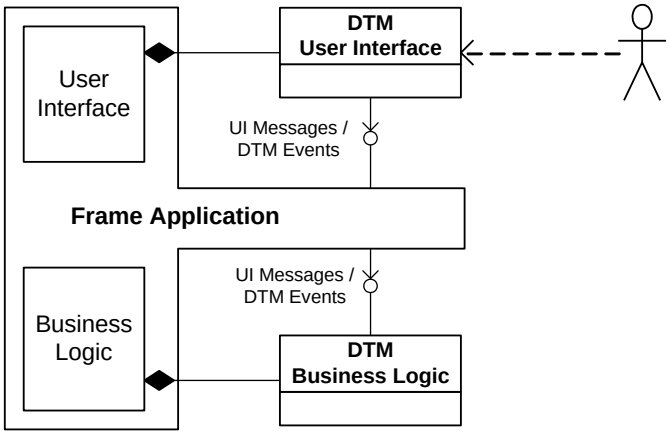


Figure 3 — Frame Application

The messaging interface is used for the transport of DTM-specific messages and events. Contents and format of the messages are proprietary and not understood by the Frame Application.

The Frame Application User Interface represents all functions of the frame related to user interface. The Frame Application Business Logic represents all frame functions related to business logic. Both are not specified by FDT, but are implementation-specific parts of the Frame Application (e.g. functional parts or structural parts). These two parts can be comprised in one single application or in separated applications, for example in a server and client application.

Frame Applications can have no, one or multiple Frame Application User Interfaces.

The Frame Application Business Logic part is responsible to execute the DTM Business Logic. It provides services which enable the DTM Business Logic to:

- persist data in the Frame Application persistence storage (see 4.14.1),
- communicate with associated device,
- request displaying of further user interfaces (e.g. user dialogs, additional DTM User Interface),
- browse the FDT topology and interact with other DTMs,
- inform the Frame Application regarding events (error / trace messages, progress etc.),
- interact with the DTM User Interface.

The Frame Application user interface part makes the DTM services available to the users, for example the functions and user interfaces supported by a DTM (see 4.5). It hosts the DTM User Interfaces as part of its own user interface and provides services to:

- interact with the DTM Business Logic (see 4.4)
- request displaying of further user interfaces (e.g. user dialogs, additional DTM User Interface)
- browse the FDT topology and interact with other DTMs
- inform the Frame Application regarding events (error / trace messages, progress etc.)

4.3.2 System communication

A Frame Application may have built-in communication capabilities, provided through Communication Channels. Frame Applications without built-in communication capabilities use Communication DTMs (see 4.7.3). Figure 4 shows the relation between the Frame Application and Communication Channel object.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

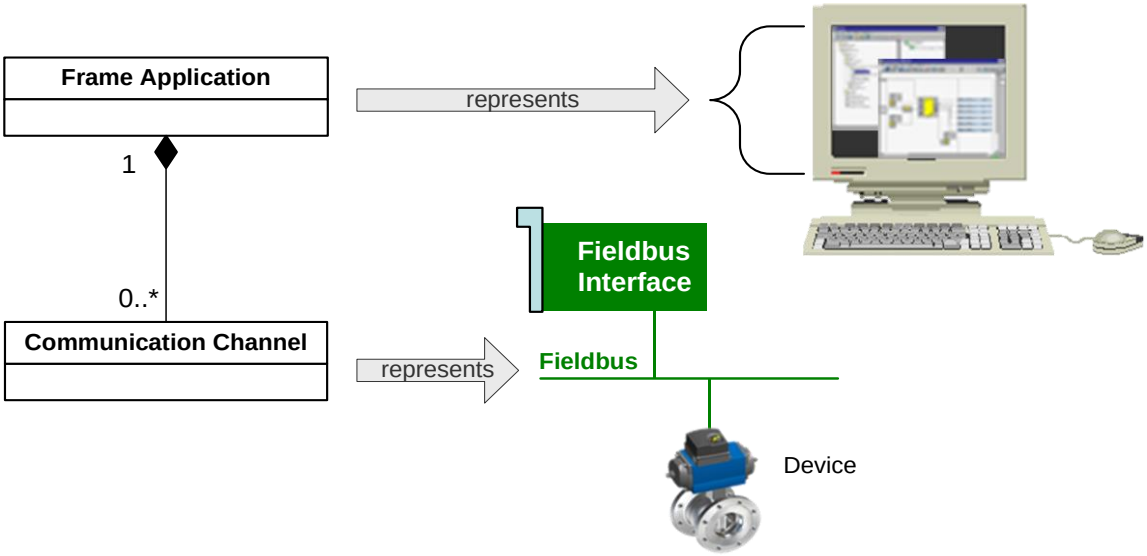


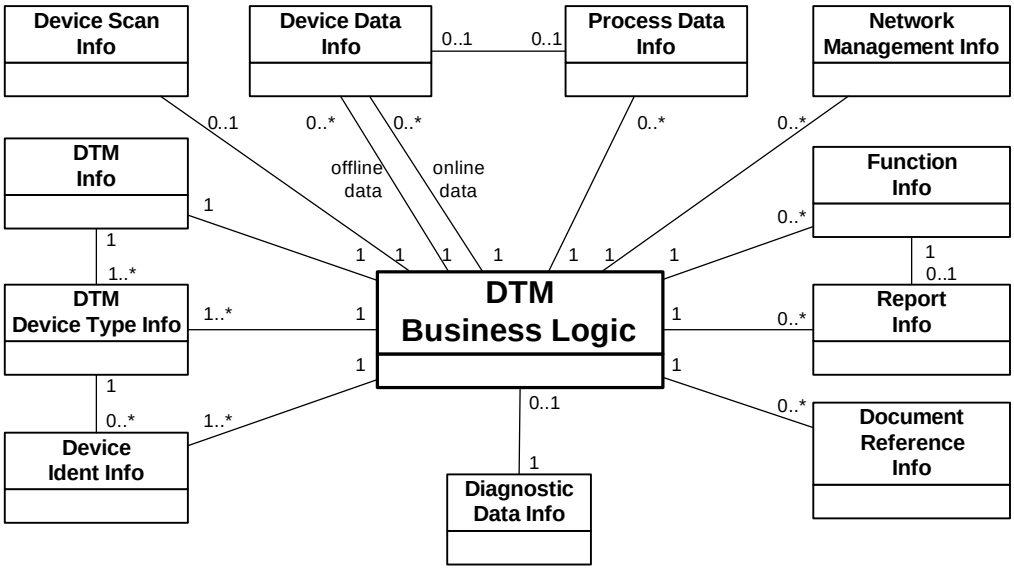
Figure 4 — Frame Application with integrated Communication Channel

4.4 DTM Business Logic

4.4.1 General

The DTM Business Logic (DTM BL) encapsulates device-specific functions and protocol-specific definitions. Thus a Frame Application is able to handle any type of device by integrating corresponding DTM Business Logic without the need for device-specific or fieldbus-specific knowledge.

The Frame Application interacts with the DTM Business Logic through defined interfaces (see Annex B). Figure 5 shows the information objects which are used by the interface definitions implemented by the DTM Business Logic.



NOTE: Since this section describes the general concept of FDT, the actual implementation in a DTM may differ. (E.g. if a network protocol uses network management, it will be mandatory to provide network management information.)

Figure 5 — DTM Business Logic

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

4.4.2 DTM, DTM Device Type, and Device Ident Info

A DTM supports interfaces to provide information about itself and the devices which are supported by the DTM. This information can be used by the Frame Application to create libraries, select a DTM during creation of the topology, or for simple validations.

In order to increase performance in creation of libraries and selection of DTMs, the information shall be provided by a separate class (the so called DtmInfoBuilder class). The DtmInfoBuilder is installed together with the DTM. It implements the same interface to provide information as defined for a DTM (see IDtmInformation in Annex B). The main advantage of DtmInfoBuilder is, that it can be used without instantiating the DTM.

Note: By using the DtmInfoBuilder it is possible to adapt the available TypeInfos depending on various conditions (e.g. DD-Interpreter DTMs may provide TypeInfos depending on installed DD files).

Figure 6 shows the information datatypes that are provided by a DTM in order to support these Frame Application function and how this info datatypes are used to describe the supported devices (see 7.4 and 7.5 for detailed description).

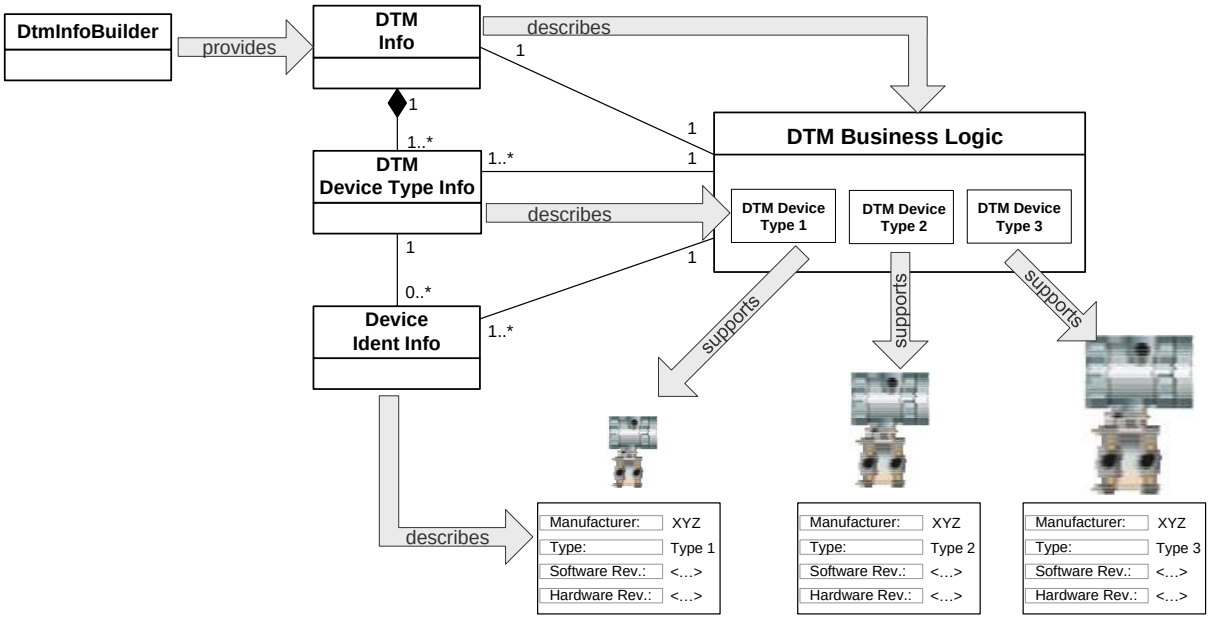


Figure 6 — DTM, Device Type and Device Ident Info

NOTE: Since this section describes the general concept of FDT, the actual implementation in a DTM may differ. (E.g. Since the scan method is important for recognizing devices, Device DTMs shall provide Device Ident Info.)

The representation for a particular physical device type within the DTM is called DTM Device Type. A DTM may provide one or more DTM Device Types. The concrete design and implementation of the DTM Device Types is not in scope of FDT.

Information about physical device types, which can be handled by the DTM Device Types is returned by the service IDtmInformation.GetDeviceIdentInfo (see definition of IDtmInformation in Annex B). Such information is for example manufacturer, type, hardware and embedded software version of the device. The DTM may even return regular expressions for some specific device identification elements to signal that the DTM Device Type can be used for all devices for which the expression matches (e.g. the term '.*' for the hardware version may signal that the DTM Device Type supports all hardware versions).

The information returned by service IDtmInformation.GetDeviceIdentInfo is fieldbus-specific and therefore defined by the document describing the protocol profile integration in FDT3. However, FDT defines the means to transform the information into a protocol-independent format to enable Frame Applications without protocol-specific knowledge to use it.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

4.4.3 Device Data Info

A DTM supports interfaces to read and write device parameters stored in the DTM instance data (offline data) and directly in the connected device (online data). This data is represented by DataInfo (see 4.15.1 for detailed description).

A DTM has to expose a defined set of device parameters which are publicly available (see 4.15 for detailed description). Parameters are provided in a bus neutral structure allowing their use without knowledge of the fieldbus protocol.

4.4.4 Process Data Info

Process data provided by devices (e.g. IO signals) are integrated into the functional planning of the control system. The process data related information for the integration of the device into the control system like datatype, signal direction, engineering units, and ranges is provided by the DTM Business Logic for each DTM Device Type (Figure 7), but may also depend on the device instance configuration.

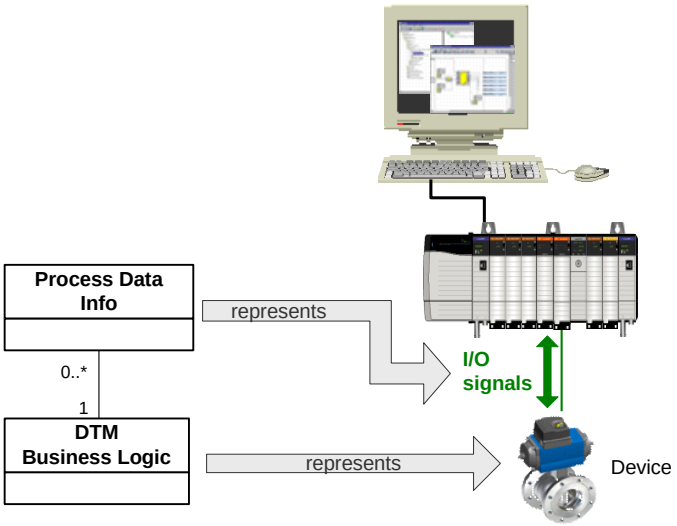


Figure 7 — Process Data Info

The process values provided by a device, both the number and type, may depend on the configuration of the DTM. Thus the number of available Process Data Info objects may also depend on the device configuration. A Frame Application is able to inform the DTM that further configuration changes shall be prohibited because the process data is already integrated into functional planning of the control system. In this case the DTM shall not allow any changes which will affect the definition of the available process values.

The process information is protocol specific. Each FDT Protocol Annex defines a derived class defining which information shall be contained. However, the properties in the base class Process Data Info provide common information (e.g. IO signal name, tag etc.) in a protocol-independent format to enable Frame Applications without protocol-specific knowledge to integrate the process variables into the system (see 7.11.1 for detailed description).

If the process data value is also available as Device Data Info, then the corresponding element is referenced by the Process Data Info element (see IOSignalRefs in 7.11.1).

A Process Data Info element may have a relation to a Communication Channel. This is a typical case for a Gateway DTM for a remote IO (see 4.7.4). The relation is represented within the Process Data Info element (see IOSignalRef in 7.11.1).

NOTE: The Process Data Info replaces Process Channel as defined in FDT1.2.x

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

4.4.5 Diagnostic Data Info

A DTM provides a service to retrieve the status of the related device. The status is encoded in a protocol independent way, according to [10][8].

4.4.6 Network Management Info

The DTM supports an interface to read and write network management information which can for example be used for address management and bus master configuration (see 8.11.1).

The Network Management Info is protocol specific. It may contain device bus-address, tag and additional protocol-specific configuration settings. Each FDT Protocol Annex defines a derived class defining the protocol-specific record. However, the base class Network Management Info provides common information (e.g. bus-address, tag) in a protocol-independent format to enable Frame Applications without protocol-specific knowledge to use it (see 7.13 for detailed description).

4.4.7 Function Info

Each DTM may provide a set of functions comprised by

- DTM User Interfaces,
- DTM functions without DTM User Interface (CommandFunctions) and
- references for external documents.

The Function Info object provides the information about the functions such as name, status (enabled / disabled), etc. (see 7.14 for more detailed description).

A function should provide printable information. A Frame Application may call the documentation interface (see [5] chapter 7.2.7.1) of the DTM to retrieve printable information. This interface returns a corresponding Report Info object which holds the printable information. The Report Info object may indicate the relation to the function.

4.4.8 Report Info

If a DTM provides instance data and/or online data, then the DTM shall implement a device-type-specific reporting of the provided data for documentation and archiving purposes. The DTM BL may implement different types of reports that each cover a distinct subset of the instance or online data of a device.

The Report Info exposes a list of report types supported by a DTM (see 7.18). The list may be grouped. The list is static over the lifetime of a DTM instance, there are no dependencies on the current application context.

4.4.9 Document Reference Info

A DTM may provide references to external documents, which are displayed by the Frame Application. These references may be provided as part of the TypeInfo (as static information) or as part of the Function Info (may change dynamically). The Frame Application may provide an own viewer for the documents or rely on the condition that external software is installed for the document type. It is recommended, that DTMs use common document formats (recommended formats are PDF and HTML).

Document references may be provided as local path or as URL. Frame Applications may apply restrictions to displaying documents from URLs for security reasons.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

4.5 DTM Functions

4.5.1 DTM User Interface

A DTM User Interface (DTM UI) shall be a web-based user interface (DTM WebUI).

The contents and layout of the DTM WebUIs is device specific, but shall follow the DTM Style Guide[6].

4.5.2 Function access control

The Frame Application can restrict the invocation of functions provided by a DTM and restrict DTM transition to certain states (e.g. restrict transition to state Online by not calling the method EnableCommunication2()).

The Frame Application will get the list of the functions provided by the DTM in IFunction.FunctionInfo property. If possible the DTM should expose all functions available in the DTM in all states (see DTM states in 6.3.2). The FunctionItem.Enabled property will indicate if the functions are not applicable for the current DTM state. It is mandatory for a DTM to expose all functions that are available in the current configuration (e.g. the current user level). A DTM may change the list of functions, if the configuration changes.

If the DTM instance was terminated and loaded again, then a DTM shall expose the same set of functions for a device node. If the DTM is updated the list of the functions may change, the Frame Application may detect an update of a DTM (see 10.4) and shall check if the list of functions changed.

The number and type of listed functions may change, as well as the status of the functions in regard to availability and visibility may change.

The DTM shall indicate any changes by IFunction.FunctionsChanged event. The Frame Application shall update the list of functions accordingly. Frame Application should hide the functions with Hidden property set to TRUE.

When a user interface for the DTM is invoked, the DTM shall not allow switching the context from within the DTM WebUI.

For Example:
If a Diagnostic Function is invoked by the Frame Application, the DTM will present the diagnostic information in the user interface. The user should not be allowed to invoke the configuration screen from within the DTM WebUI for the diagnostic function without the permission from the frame. The user shall be able to invoke the configuration screen from the functions exposed to the Frame Application if the permissions in the Frame Application allow it.

If the MainOperation Function is invoked by the Frame Application, the DTM will present all available functions within the user interface. The user may invoke the Configuration screen from within the DTM WebUI as well as the Diagnostic function or any other integrated function.

4.5.3 User interaction in DTM WebUIs

If a user changes values in DTM WebUIs, those changes are applied after confirmation to the respective DTM or device (see [6]).

Status information for a DTM WebUI (data source, summary parameter modification state, UI operation mode, activation of service mode) can be displayed (see [6]).

Note: This behavior is modelled according to the general user experience in web-based GUIs. It is based on the expectation that a web user interface may lose connection to its data source at any time.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

4.5.4 Command functions

Command functions are used to execute actions (commands) either on the DTM BL. Command functions in context of the DTM BL shall not have an GUI.

A Command function may have parameters. The information about the parameters, which is provided by the DTM BL, may include default values of the parameters. The actual parameter values are passed when the Command function is executed.

One example for command functions is: Reset device.

4.5.5 Static Function

Static Function is a concept that allows processing information from a device without executing the DTM.

For observation and monitoring of a plant often it is necessary to retrieve not much data from many devices. In such use cases the effort to start and manage many DTMs may be too high. On the other hand in such use cases it may be not sufficient to read a single item from the device. Sometimes the data read from the device needs to be processed or to be combined with additional information in order to calculate the correct information.

A Static Function is part of a DTM package, the DTM provides information regarding the Static Function. A Static Function has no access and no relation to the data of a DTM.

StaticFunctions are executed using a StaticFunction Provider object. The Frame Application is responsible for creating the Static Function Provider objects. There shall be only one Static Function Provider object per device and it is initialized with the necessary communication data. On one StaticFunction Provider object the execution of a Static Function may be started multiple times in parallel. It is the responsibility of the StaticFunction Provider to process the functions sequentially in the order they were started.

It is recommended that a DTM provides Static Functions to support well defined use cases as defined by FDT protocol project groups and Profile Working groups.

Examples of Static Functions are:

- GetDeviceStatus
  Provides the current device status in FDT format (see Fdt.Dtm.DeviceStatus)
- GetProcessValue
  Provides the current process value of the device (input or output) in normalized format
- GetOperationTime
  Provides the operation time of a device in .NET Time format.

See Annex L for definitions of standard Static Functions, which provide support for common use cases.

4.6 Communication Channel

A Communication Channel provides access to the fieldbus and is communication technology dependent. It represents the entry point to a fieldbus network, a vendor-specific backplane bus or point-to-point connection. A Communication Channel may also represent logical communication to blocks within a device. A Communication Channel may be provided by a Frame Application (see 4.3.2) or by a DTM Business Logic (see 4.7.3).

A Communication Channel provides interfaces that are independent from the communication hardware and the fieldbus protocol. The interfaces provide methods to

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

- scan the fieldbus for available devices,
- connect to a device,
- send communication messages and
- disconnect from a device.

The interface methods take base arguments which are extended for protocol-specific communication by FDT Protocol Annex specifications (see Figure 8).

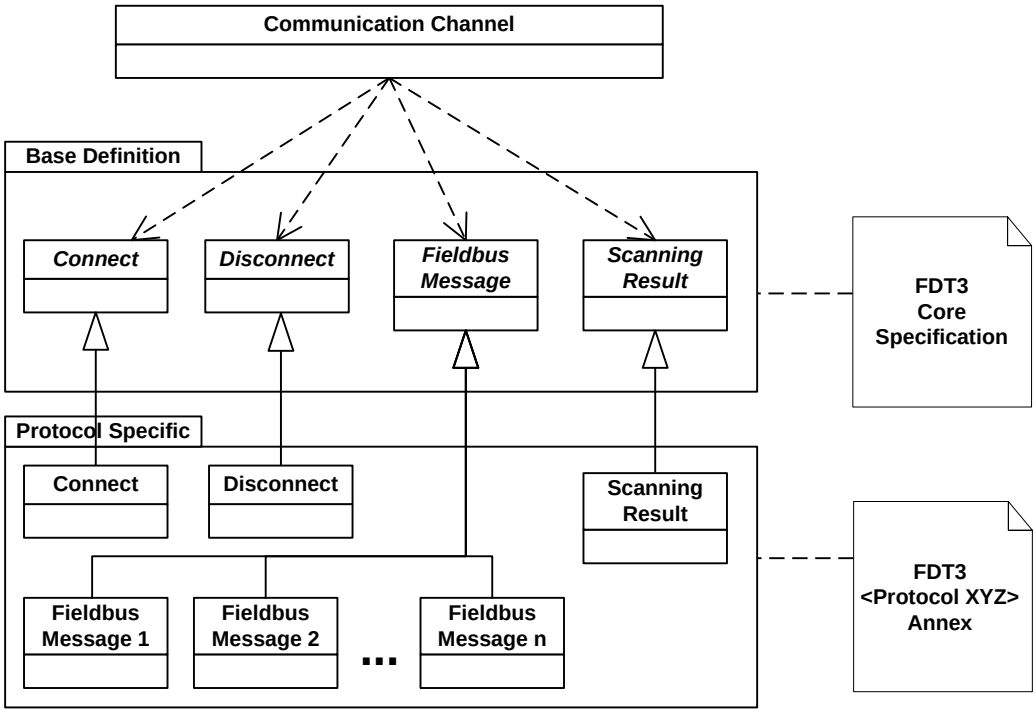


Figure 8 — Communication Channel

FDT also allows manufacturers to define support for proprietary bus protocols, which are not part of the standard; for example for a manufacturer-specific fieldbus or point-to-point communication

The communication services provided by the Communication Channel interface may be used by the Frame Application or a DTM to exchange data with a connected device or to initiate a function (e.g. identification request, device reset, broadcast, etc.). The general communication concept is further described in IEC 62453-2.

A Communication Channel shall be able to support multiple connections at the same time if the communication protocol supports this. This means supporting several DTMs creating connections to different devices as well as several DTMs creating connections to the same device. Depending on the supported protocol it may also be possible that one DTM creates several connections to one device.

4.7 DTM categories

4.7.1 General

DTMs represent measurement and control devices, modules, gateways or blocks as well as communication interfaces. The DTMs representing these devices only differ in the availability of provided data, functions, and user interfaces. The different DTM categories reflect the physical entities, which are represented by a DTM. The defined DTM categories cannot describe rules for all existing physical devices, but are intended to provide a general guideline for common categories of physical devices.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

4.7.2 Device DTM

A Device DTM contains the application software for a standard field device, e.g. a pressure transmitter, a valve, or a drive. Such a DTM for example supports functions to configure and parameterize the device (see Figure 9).

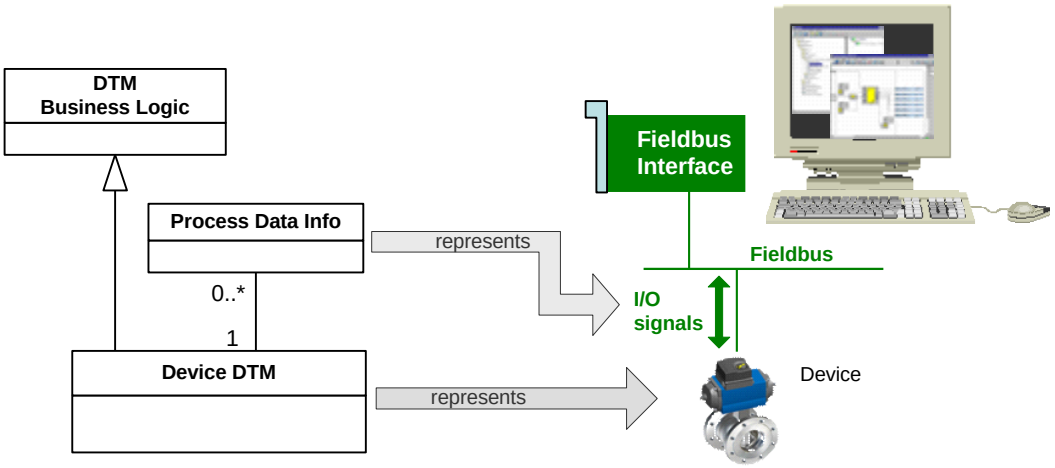


Figure 9 — Device DTM

The Device DTM shall provide ProcessDataInfo objects (see 7.11.1), if the device provides process data.

4.7.3 Communication DTM (CommDTM)

A Communication DTM is a special category of DTM containing the application software for specific communication hardware. For example, it may support configuration settings like baud rate, start and stop bits, etc.

A Communication DTM shall provide Communication Channels (see 4.6) which provide the services to access the fieldbus (see Figure 10). The number of provided channels may depend on the configuration of the Communication DTM (i.e. it might occur that a DTM temporary provides no channel, for instance during reconfiguration).

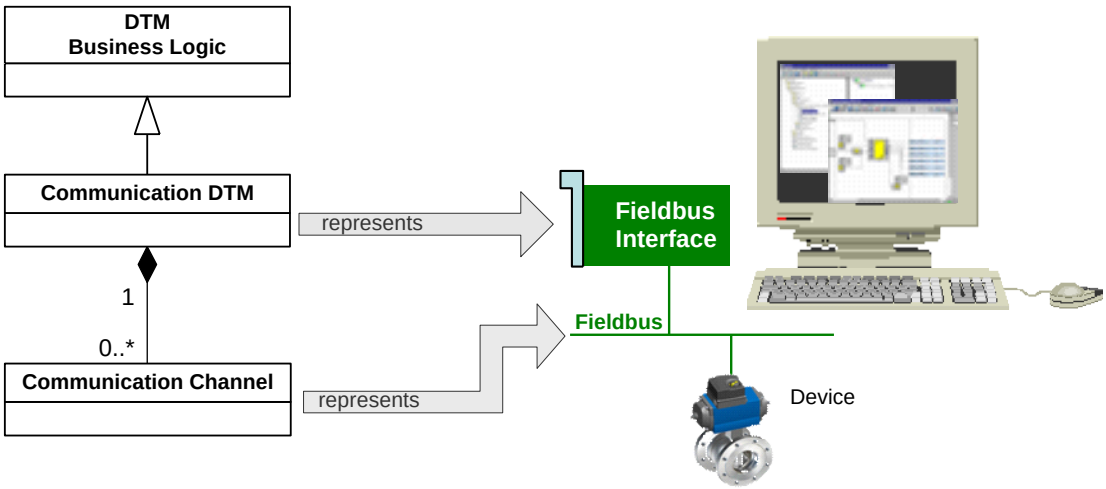


Figure 10 — Communication DTM

The advantage of Communication Channels provided by a Communication DTM compared to the Communication Channels provided by a Frame Application is, that they may be utilized by different Frame Applications. Each Frame Application thus may extend the number of protocols

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

that the system is able to access. Both ways of providing Communication Channels may coexist in a Frame Application.

4.7.4 Gateway DTM

A Gateway DTM is a special category of DTM representing a device that connects different fieldbus segments, for example PROFIBUS DP with HART (see Figure 11).

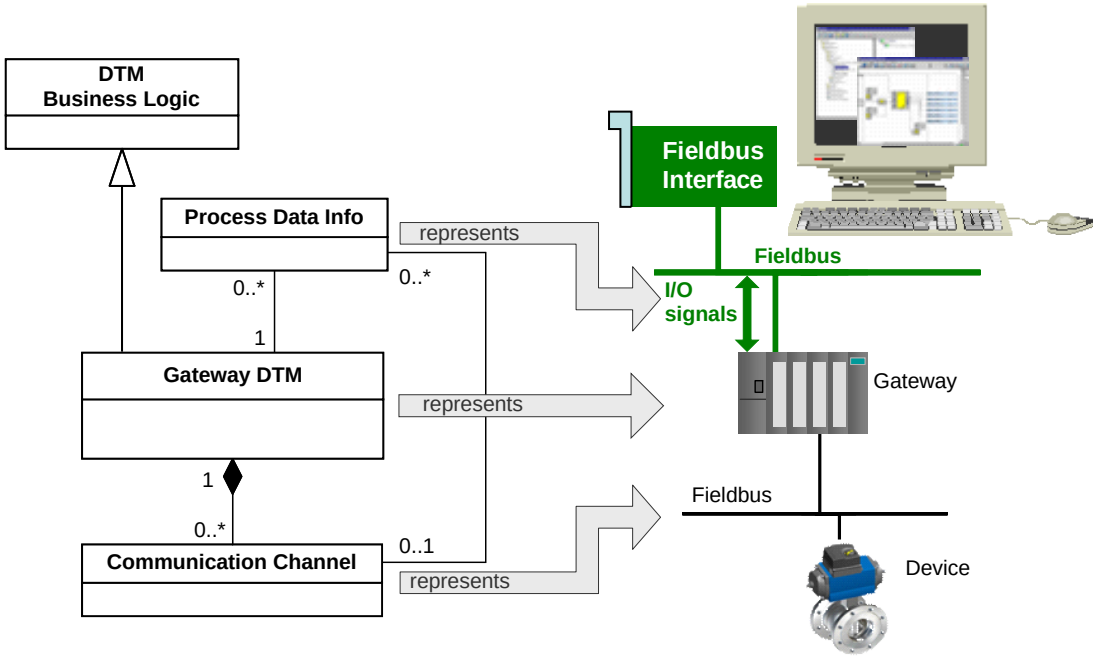


Figure 11 — Gateway DTM

Such a DTM can be seen as a combination of a Communication DTM and a Device DTM. It shall provide Communication Channels (see 4.6) to provide access to the linked fieldbus and ProcessDataInfo objects (see 7.11.1) for the fieldbus where it's connected to, if it is required to make the gateway I/O signals or process values accessible to the host system.

There may be a relation between the IO signals provided by a gateway device and the communication interface of that device (e.g. the gateway provides IO signals that are received from a device connected to the communication interface). In such a case the relation is represented as relation between Process Data Info and Communication Channel.

A Child DTM is connected to the Gateway DTM via the Communication Channel of the Gateway DTM. The Frame Application is responsible to create and manage the link between the DTMs. The Device DTM is able to communicate with its device via the services provided by the Communication Channel. This concept is called 'Nested Communication' and is further described in chapter 4.11.3.

4.7.5 Composite Device DTM

Some devices are composed from modules and sub modules. Hardware modules are plugged into physical slots of the composite device (modular device). For example, an I/O module can be plugged into a Remote I/O device. The communication between the composite device and the modules is provided usually by a backplane bus. The function of the backplane bus is defined by the manufacturer and often is based on private protocol definitions. Such a composite device may be represented by a hierarchy of DTMs as shown in Figure 12.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

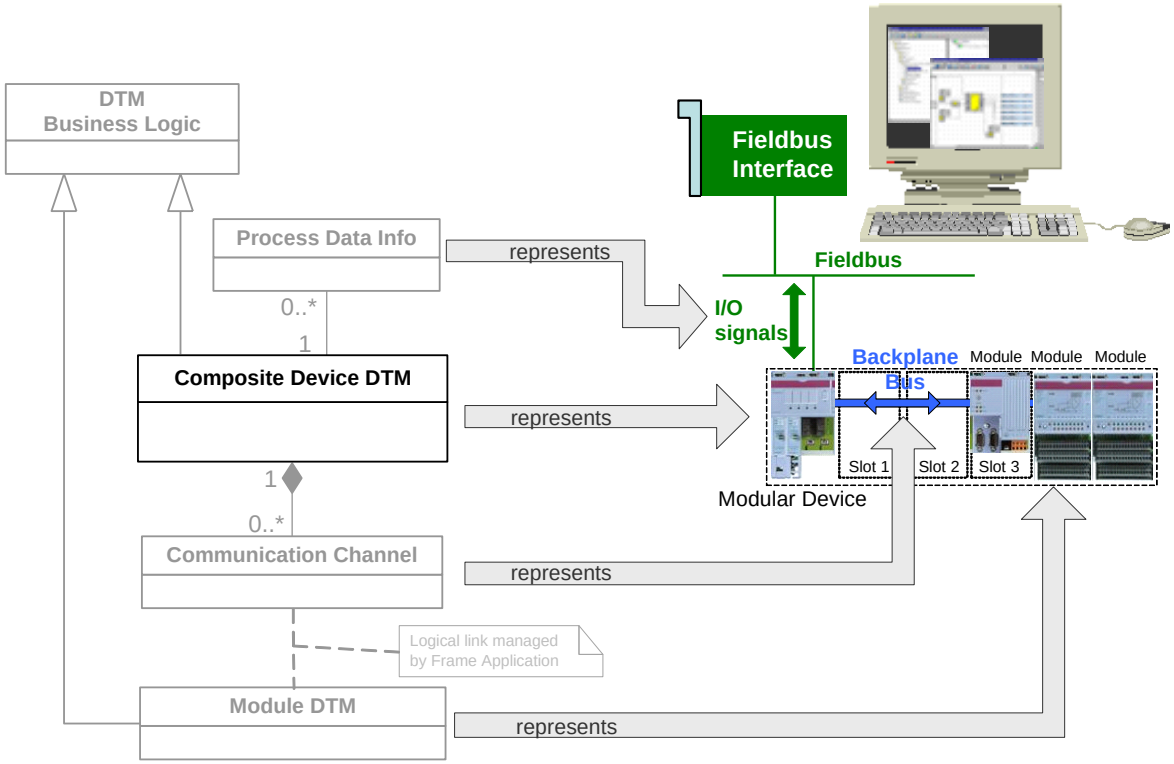


Figure 12 — Composite Device DTM

A Composite Device DTM is the root element for the complete device representation. This DTM shall provide Communication Channels (see 4.6) which represent the backplane bus. Depending on the software design of the Composite Device DTM one channel may be provided for representation of the whole backplane bus or multiple channels may be provided for representation of the slots where the modules are plugged in.

If the composite device provides process data, then the Composite Device DTM shall provide ProcessDataInfo objects (see 7.11.1). If the composite device forwards process data that is generated in the device modules, then the ProcessDataInfo provided by a Composite Device DTM may be composed from ProcessDataInfo that is provided by the Module DTMs.

4.7.6 Module DTM

A Module DTM is a special category of DTM containing the application software for a device hardware module (see Figure 13).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

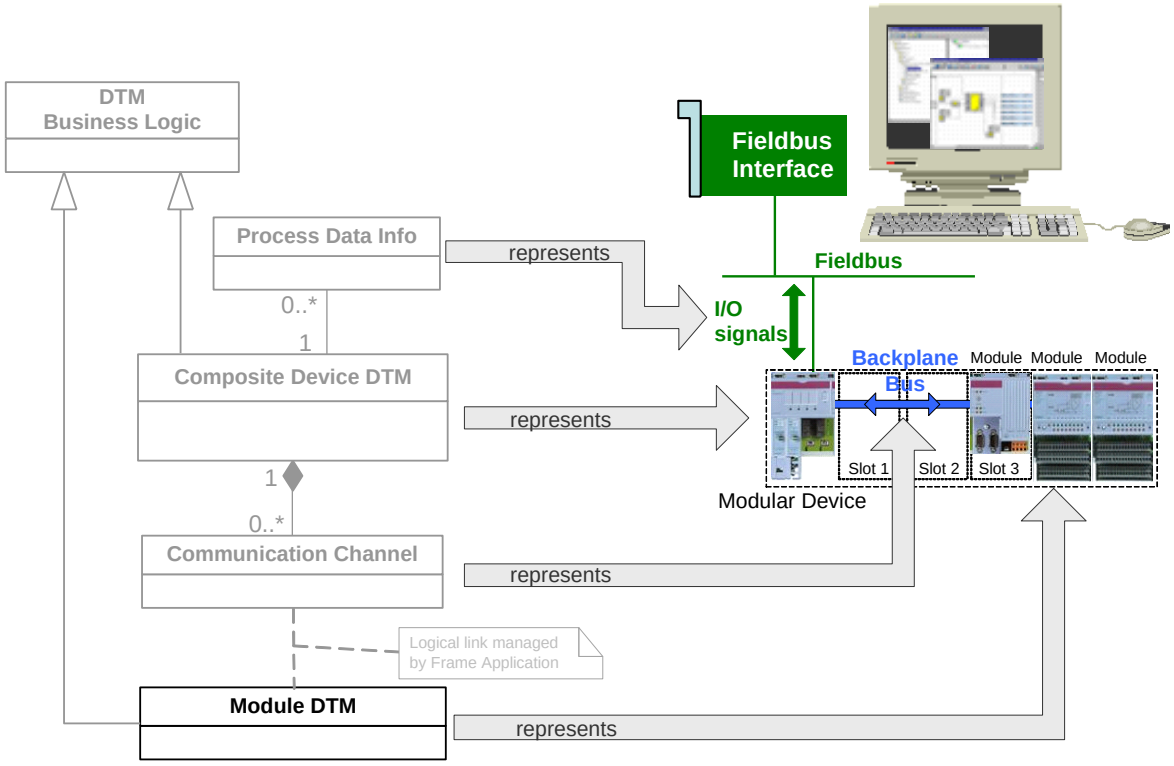


Figure 13 — Module DTM

The device modules are represented by Module DTMs. The Composite Device DTM and Module DTMs are generally shipped together and supplied by the vendor of the modular device. The relation between Composite Device DTM and related Module DTMs (and their respective DtmTypes) is defined by a specific ProtocolId for the backplane bus protocol.

A Module DTM is connected to the Composite Device DTM via the Communication Channel of the Composite Device DTM. The Frame Application is responsible to create and manage the link between the DTMs. The Module DTM is able to communicate with its module via the services provided by the Communication Channel. This means that the concept 'Nested Communication' (see chapter 4.11.3) is also applied for Composite Device DTMs and the related Module DTMs.

If the device modules provide process data, then the respective Module DTMs should provide ProcessDataInfo objects.

If the device modules provide access to subordinate communication (e.g. in a modular gateway), then the respective Module DTMs should provide Communication Channels.

4.7.7 Block Type Manager

A Block Type Manager (BTM) is a special category of DTM containing the application software for accessible software objects inside a device. Often these software objects are defined as blocks, such as resource blocks, transducer blocks, and function blocks (see Figure 14).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

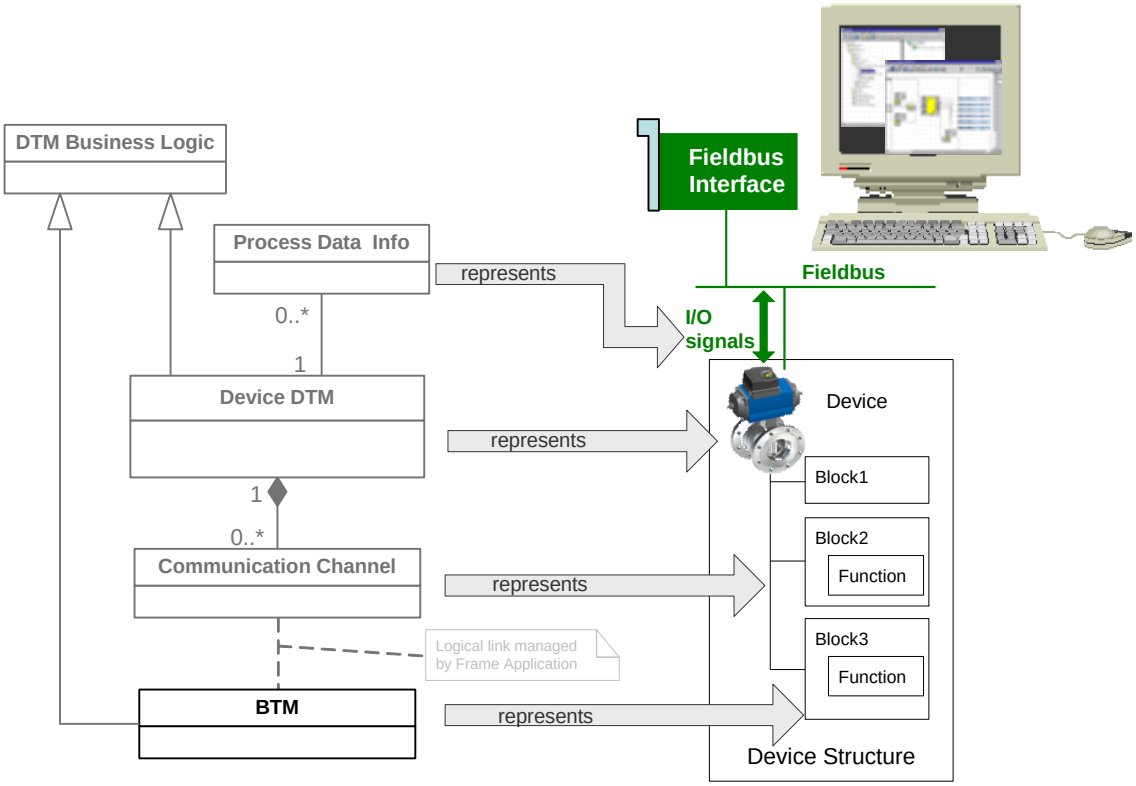


Figure 14 — Block Type Manager

A Device DTM is the root element for complete device representation. This Device DTM shall provide Communication Channels (see 4.6) which provide services to access the block information within the device.

A device may host standard blocks, vendor-specific blocks, and device-specific blocks. All types of blocks are represented by BTMs. The blocks are more flexible than device modules (see 4.7.6), for instance blocks may be moved from one device to another device (standard blocks may be even be moved between devices of different type). The Device DTM and the BTMs for device-specific blocks in general are supplied by the device vendor. BTMs for standard blocks may be supplied separately and may be used with all DTMs for devices that are able to host the respective blocks.

A BTM is connected to the hosting Device DTM via its Communication Channel, indicating that the respective blocks are hosted in the physical device. The BTM is able to interact with its block in the device via the services provided by the Communication Channel of the Device DTM. The Communication Channel represents an internal communication on the device and is used mainly to manage the link between BTM and block. The protocols supported by the CommunicationChannel are used to manage which BTMs may connect to a DTM (standard protocols for standard blocks, specific protocols for specific blocks). Additional validation rules may be applied by the Communication Channel. The Frame Application is responsible to create and manage the link between the Device DTM and the BTMs.

The BTM concept is independent from fieldbus protocol. However, a protocol may require modeling of device structures as blocks. Thus, whether and how the BTM concept is used to model device structures is defined by the FDT Protocol Annex.

If the blocks provide process data, the respective BTMs may provide ProcessDataInfo objects.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

4.8 User management

4.8.1 General

FDT does not define a standard system for user management. The user management is part of product-specific definitions and may be implemented differently for different Frame Applications. Still it is necessary to define a common handling for access permissions, access rules and how components from different vendors communicate information regarding access permissions.

4.8.2 Multi-user access

Some Frame Applications provide multi-user capability. Such a system provides access for multiple users at the same time and may be distributed over several computers. This specification considers the distributed environment as one Frame Application. The Frame Application and the DTM are equally responsible to provide the multi-user access and to ensure consistency of data.

If, within one Frame Application, multiple users access the same device or the same device dataset, the Frame Application shall start a separate DTM instance for each user. All these DTM instances shall have same DTM type, shall be instantiated for the same DtmDeviceType and for the same physical device. Each DTM instance manages a separate instance dataset. These instance datasets are synchronized by means of the persistent dataset (see 4.14.5).

Note 1: An example architecture for multi-user scenarios may be found in chapter 11.4.

Note 2: These rules also apply to Communication DTMs. If a Communication DTM provides access to a resource, which supports only limited access (e.g. a COM port), the Communication DTM might have to provide special measures in order to support multi-user access.

4.8.3 User levels

4.8.3.1 Introduction

DTMs may be integrated in different Frame Applications, which may have varying requirements to restrict visibility and accessibility of device and persistent data, for example for plant safety reasons or to present a customized view to the user. The grade of restriction varies with the types of users supported by a system. Examples for users requiring data access restrictions are:

- a user assigned to observe a plant shall not have access to calibration-specific device parameters and consequently shall not see related DTM functionality,
- a device commissioning specialist needs to have access to calibration data and functions,
- a user assigned to operate a plant shall be able to change (write) set point values and be offered appropriate functionality while a user assigned to observe a plant is not allowed to execute such changes.

4.8.3.2 Access Control Concept

FDT uses a role based access control concept. A Frame Application initializes a DTM Business Logic with the FDT-specific user level (see definition of IDtm3.Init()in Annex B). The user level is immutable over the lifetime of the Business Logic/User Interface instances.

In terms of access control, every actor in an FDT system may have one of the following three user levels at the interface of a DTM (Table 1):

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 1 — FDT User levels

User Level Name	Description
Observer	This user level stands for an actor that observes the current process only.
Expert	This user level stands for an actor who has to execute specific use cases, e.g. operation use cases (operation expert) or device maintenance use cases (maintenance expert). This user level allows the Frame Application to configure access and privileges.
Engineer	This user level stands for an actor that has to do the plant planning, device configuration/parameterization and plant maintenance.

The user levels allow a stepwise extension of permissions. The Observer typically has a minimum permission set, the Expert has an intermediate permission set (which is configured by the Frame Application) and the Engineer has a full permission set.

NOTE: For an explanation of the fundamental user levels and use cases that were considered for the design of the FDT specification see Annex A. A Frame Application may support only a subset of these use cases or additional use cases not defined in the annex.

According to the role set by the Frame Application, the DTM Business Logic and User Interface shall control access to device and persistent data (see definition of interfaces IInstanceData and IDeviceData in Annex B) as well as adapt its user interface appearance. This includes to hide some data or display it as read-only, but also to partially disable DTM-specific functionality (see definition of IFunction.FunctionInfo in Annex B), if it requires data access rights that are not associated with the specified user level.

It is mandatory for a DTM to implement a safe and read-only usage for the “Observer” user level. It is also mandatory for a DTM to implement unlimited usage for the “Engineer” user level. It is optional for a DTM to implement configurable custom usage for “Expert” user level. If “Expert” level is not implemented by a DTM but is set by the Frame Application, the DTM shall use the behavior for “Observer” user level.

Table 2 gives an overview about the user interfaces and functions that are expected to be available for the individual user levels. The data access rights should be defined to allow for the execution of these use cases.

The assignment of roles to individual users is Frame Application specific. The Frame Application may implement an own user management sub-system or use fixed user levels.

Table 2 — Role-dependent Access Rights and User Interfaces for DTMs

Use Case	Sub Cases	User level		
		Engineer(M)	Expert(O)	Observer(M)
System Planning	Network Management	If a DTM implements these use case, it shall expose all related commands and user interfaces.	If the DTM implements these use cases and supports the user role “Expert”, it shall allow Frame Application to configure access to the exposed data, commands and user interfaces	-*)
	Busmaster Configuration			-
	Channel Assignment			-
System Generation	Network Management			O {r}
	DTM matching			O {r}
Device Configuration	-			-
Simulation (Force)	-			-
Offline Operation	Offline Parameterization			-
	Persistent Data Comparison			-

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Use Case	Sub Cases	User level		
		Engineer(M)	Expert(O)	Observer(M)
Online Operation	Online Functions (reset + other functionality that requires online device connection)	If a DTM implements these use cases, it shall expose all related commands and user interfaces.	If the DTM implements these use cases and supports the user role “Expert”, it shall allow Frame Application to configure access to the exposed data, commands and user interfaces	-
	Online Parameterization			-
	Calibration			-
	Device/Persistent Data Comparison			-
	Adjust SetValues			O {r}
	Upload			-
	Download			-
Bulk Operation	Upload			-
	Download			-
Online View	Network Scan			O {r}
	Online Status			M {r}
	Online Trend			M {r}
	Device Identification			M {r}
	Online View Parameter Set			O {r}
Report Generation	-			M {r}
Device-specific Operations	Device-vendor-specific (or extended) DTM functions after DTM/Device-specific OEM Service login			-

M Mandatory (if a DTM implements this use case, it shall expose all related commands and user interfaces in the specified user level)
O Optional (a DTM may expose the related commands and user interfaces in the specified user level)
r User level shall have read access to all data related to the use case
- Use case not supported in this user level (DTM shall not expose any related commands/user interfaces)
*) A DTM shall allow all user levels to set the device address in the DTM with the method SetAddressInfo().

4.8.3.3 Frame Application configured access control

It is very difficult and even impossible for the DTM vendor to provide correct access control settings for all occasions. The data, which can be accessed, and the functions, which can be used, are changed by the user; depending on where the DTM is used or what is the operational phase of the plant. Here are some examples:

1. The same user may have different permissions for the same device controlled by the same DTM when the device is connected in the plant or in the instrument shop. In the instrument shop, the user may have all equipment to calibrate the instrument and the corresponding privileges should be granted. Little or no changes may be allowed when the instrument is connected later to the actual running control system.
2. The same user may have full control when the plant is being engineered, but the changes to the device may be significantly restricted, when the plant is in running state.
3. The same group of users may have different permissions for different instruments – some of the maintenance personnel may be trained to work with transmitters, other may be specialized in valve maintenance.
4. In a small application one person may be responsible for the entire application and he may have unlimited access to all device maintenance procedures, but in a big application, often the access is controlled according to the individual experience of the technical staff.

To address the different cases, an “Expert” user level is provided. When the DTM supports the “Expert” user level, the data which can be accessed and the function which can be invoked are

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

restricted by the Frame Application depending on the rules in the plant, on the operational phase, the individual user or team experience and other factors.

The Frame Application can use the Expert user level to define additional levels of access for individual users or for groups of users. For example, if access control is configured for an Operation Expert, the Frame Application may enable access to Set Point Values, to Tuning parameters and to Diagnostic functions. In another example, if the Device DTM is invoked in the instrument shop environment, the Frame Application may enable access to Calibration parameters, to calibration functions and to Online Parameter View.

Note: The user level "Operator" as defined in FDT1.x specification is not supported in FDT3. The term Operator in this document is used to describe an expert for plant operations.

4.9 FDT and system topology

4.9.1 General

FDT differentiates two topology views: logical topology and physical topology.

A logical topology is created by a hierarchy of DTMs. Child DTMs are connected to Parent DTMs via the Communication Channel of the Parent DTM. A Parent DTM may have multiple children. This relation is managed by the Parent DTM. This means that a Parent DTM knows all its Child DTMs.

A Child DTM may be assigned to multiple parents (e.g. if different network paths may be used to access a device). A Child DTM is not notified if it is assigned to a Parent DTM, but it may request a list of parents from the Frame Application by using the method ITopology.GetParentNodes().

A Child DTM can use only one communication path at a time to access the respective device. The Parent DTM providing this communication path will be marked by the Frame Application as 'primary parent'.

This means, that the logical topology describes the logical relations between the devices on an abstraction level that supports managing the communication between DTMs and devices.

A physical topology is created by defining physical connections between DTMs. Connections are defined between Ports of the DTMs. This means the physical topology describes the actual hardware installation. The connections are managed by the Frame Application. It is possible to use these connections for representation of all kind of network structures.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

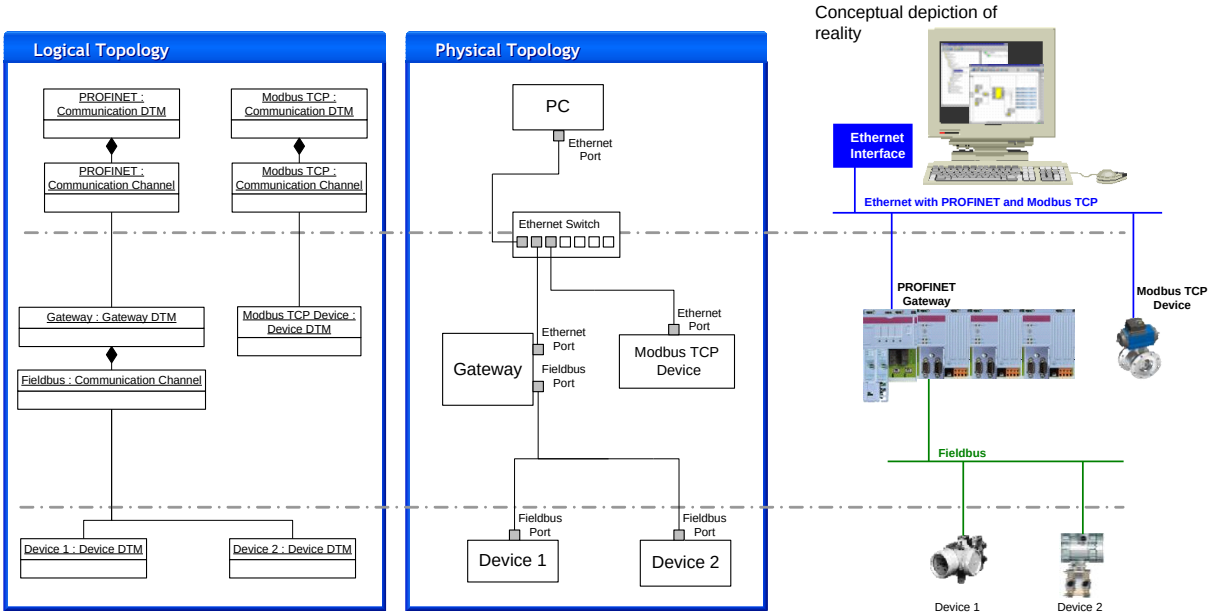


Figure 15 — Logical topology and physical topology

4.9.2 Topology management

4.9.2.1 Logical topology

The Frame Application is responsible for managing the logical topology – it is mandatory to support the logical topology. That means the Frame Application shall organize the routing of data for accessing a device in the plant. Some Frame Applications may require user interactions; others may support automatic operations such as topology import or fieldbus scanning. The sum of all links between DTMs according the logical topology is called FDT topology and further described in IEC 62453-2.

A DTM exposes all required information (see 4.4.2) which enables the Frame Application (and the user) to choose the appropriate DTM for a device, for example name, vendor, version of supported device types and corresponding identification properties.

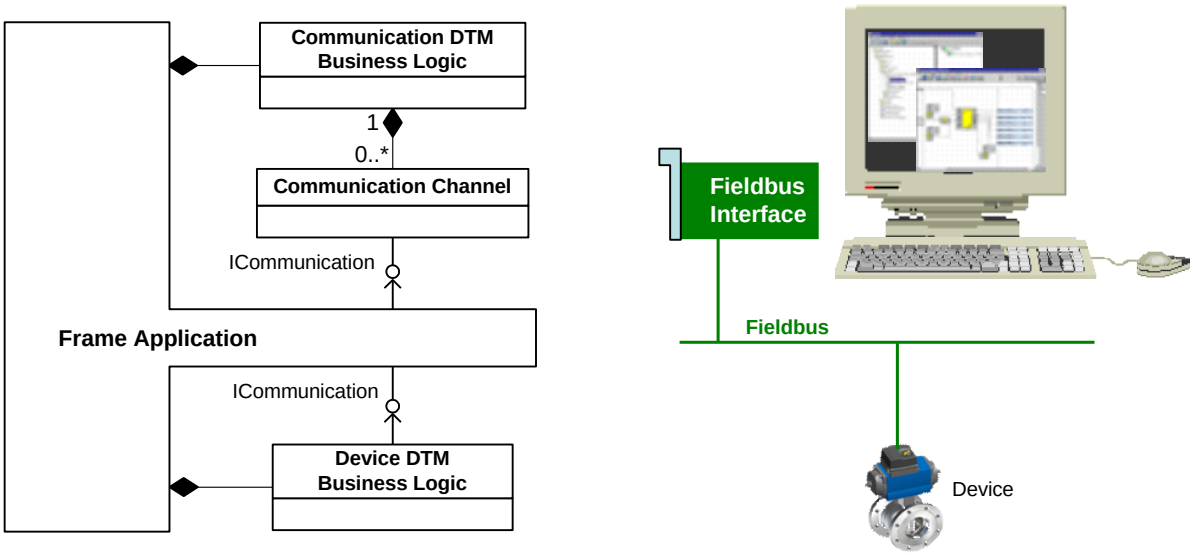


Figure 16 — FDT and logical topology

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

As shown in Figure 16 a Communication Channel is used as the linking element between Communication DTM and Device DTM. The Communication Channel provides access to the fieldbus.

The link between a Communication Channel and a DTM is created by the Frame Application. However, final decision whether a DTM shall be linked or not has to be made by the Communication Channel. The Frame Application has to call the method ISubTopology.<ValidateAddChild()> (see definition in Annex B) before link is created. The Communication Channel shall at least check whether the required network protocol of the DTM to be linked fits to its own supported protocol. If this is not the case, then the linking shall be rejected. In addition, the Communication Channel may perform further checks, for example whether the number of linked DTMs exceeds a limit.

Neither the Communication Channel (or corresponding DTM) nor the linked DTM shall need to manage topology information in order to access the respective physical device. The Frame Application supports to request topology information by the methods ITopology.GetParentNodes(), ITopology.GetSiblingNodes() (and ITopology.GetChildNodes() (for all see definition in Annex B).

The rules for identification of DTMs and devices are described in section 4.12.

For some communication protocols the order of the devices linked to the network affects the configuration of the network itself. This order is defined when inserting the corresponding DTM into the logical topology (ITopology.BeginAddChild()) and can be modified later via ITopology.BeginRepositionChild(). The Frame Application always shall maintain this order when returning collections of DTMs in ITopology.GetChildNodes() and ITopology.GetSiblingNodes().

If the Frame Application provides a view on the channel, it shall show all Child DTMs in their respective order. In this view, the Frame Application shall allow the user to insert a new DTM at a specific position between the existing Sibling DTMs or to change the position of an existing DTM in regard to its Sibling DTMs.

4.9.2.2 Physical Topology

Advanced topology management requires the additional planning of cable bound or wireless connections between devices. This capability is provided by the Physical Topology.

The management of the Physical Topology is the responsibility of the Frame Application. It is optional for a Frame Application to support the Physical Topology. It is mandatory for a DTM to expose all information which is required to determine whether a physical connection is possible or not. The Physical Topology may not have dependencies to the Logical Topology and shall be handled separately as shown in Figure 17.

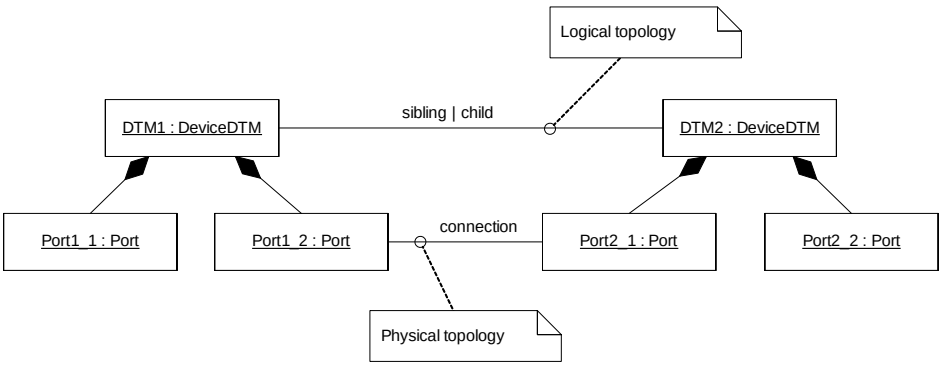


Figure 17 — DTMs and physical topology

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

The connections are managed by the Frame Application. Information regarding connections may be accessed with the interface IPhysicalTopology. See Annex B for a detailed description.

4.9.2.3 Communicating and non-communicating devices

An automation system integrates communicating devices, as well as devices which do not communicate and therefore are not configurable via communication (e.g. power supplies and other network infrastructure elements). Information about such non-communicating devices may be essential during the planning phase of the communication system and can be used to verify integrity of the network, for instance in regard to bus power overload, communication distance limitations, validity of the design (e.g. correct termination). In order to integrate such devices in an FDT-based system, a DTM may be provided to represent such a 'passive device'. The DTM provides information about the device/equipment to the Communication DTM, which is capable to use this information.

Different protocols require specific information to be provided and may have different limitations to be enforced. Information provided from DTMs for passive devices has to be provided in a standard way and format (in Network Management Info), so that different Communication DTMs can use it in a standard way. DTMs for communicating devices may need to provide similar information. Protocol-specific extensions have to define the information provided by the communicating devices and by the non-communicating devices and also how this information is used. For example, Communication DTMs for a bus powered protocol can use information from non-communicating devices (defining the power source) and communicating devices (defining the power consumption) in order to balance the power on the network.

4.9.3 Address management

Fieldbus systems generally use an addressing concept to distinguish between the different connected devices. Therefore a DTM generally needs the information about the address of its associated device in order to establish an online connection to it (see method ICommunication.<Connect()>, definition in Annex B).

A DTM for a device type which can be connected to a system which defines an addressing concept has to provide the interface INetworkData (see definition in Annex B). These services enable to read and write the device addressing information to the DTM. The addressing schema is fieldbus specific, therefore concrete information which shall be passed to the service is defined by the FDT Protocol Annex documents describing the protocol profile integration in FDT.

Always the component providing the Communication Channel (DTM or Frame Application) is responsible for address setting in the linked DTMs. A DTM providing Communication Channels shall support a DTM WebUI that enables the user to set the addresses (ApplicationID = NetworkManagement, see definition in Annex B) like shown in Figure 18.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

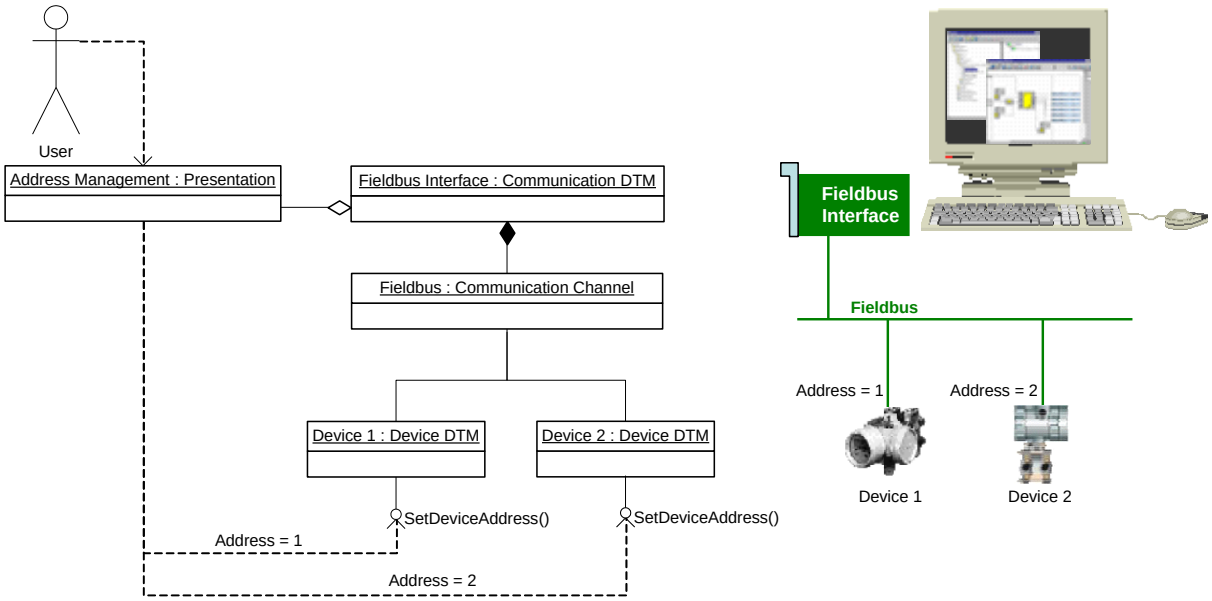


Figure 18 — Address setting via DTM WebUI

Furthermore the Communication Channel itself shall support the method ISubTopology.<SetChildrenAddresses()> (see definition in Annex B) to enable the Frame Application to initiate address setting. The sequence diagrams in 8.13 describe different addressing scenarios more detailed.

How address management is realized if Communication Channel is provided by the Frame Application is not in scope of this specification.

4.9.4 Configuration of fieldbus master or communication scheduler

Many fieldbuses use dedicated devices to control the communication between the different participants, for example the Master-Slave or the Token-Passing concept.

The devices controlling the communication over the fieldbus are called Fieldbus Master or Communication Scheduler. These devices usually need to be configured with the bus parameter information of the devices connected to the fieldbus. In general, the configuration is done with a device-specific configuration tool. The FDT component representing the Fieldbus Master or Communication Scheduler shall provide this configuration tool. That means the configuration tool is either part of the Frame Application or like in Figure 19 part of a DTM.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

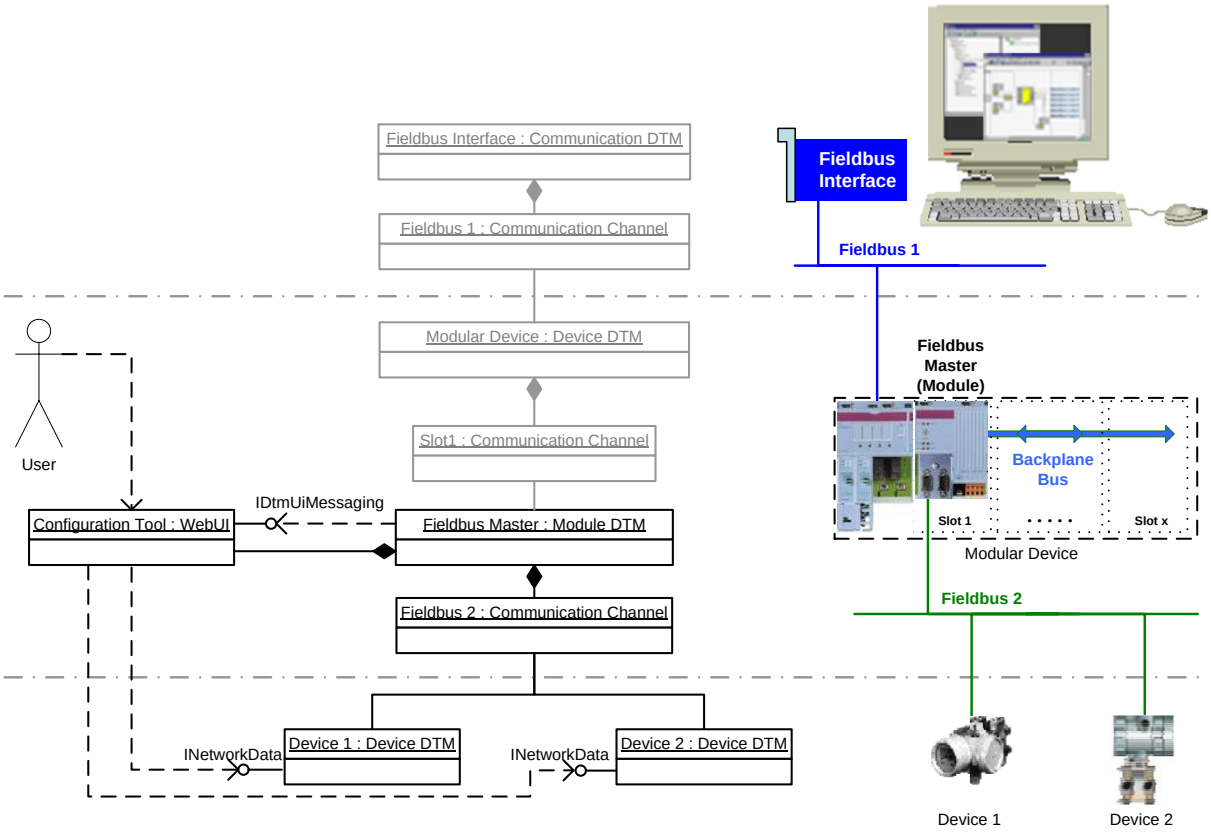


Figure 19 — Fieldbus master configuration tool as part of a DTM

The configuration tool is able to read and write the bus parameter information of the fieldbus participants via the interface INetworkData (see definition in Annex B) of corresponding DTMs.

The specific bus parameter information which can be read or written via NetworkManagementInfo services are defined by the protocol-specific documents describing the protocol profile integration in FDT. Furthermore, the general bus configuration is also more detailed described in the protocol-specific documents.

4.9.5 Data exchange between Frame Applications

The interaction between different Frame Applications is not in scope of FDT, but the FDT specification defines datatypes (see TopologyImportExport definition in Annex B) which can be used for this purpose. These datatype classes may be used by one Frame Application to export the FDT topology information to an XML file which then can be imported in another Frame Application.

4.10 Modularity

Different fieldbus protocols are using different device models. FDT supports following different approaches to describing the structure of device:

- monolithic DTM with topology description in NetworkDataInfo, ProcessDataInfo, ProcessImageInfo, DataInfo and CommunicationChannelInfo
- Module DTM, and
- BTM.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

4.11 FDT communication

4.11.1 General

In FDT the communication is based on passing communication requests from the consumer of communication to the communication provider, which is represented by a Communication Channel. Passing and receiving communication requests is based on asynchronous methods (see 5.8.7).

In order to submit communication requests the consumer of communication first has to establish one or more connection to the respective physical device. This allows the management of active connections by the Communication Channel.

4.11.1.1 Handling of communication requests

In order to optimize the communication, the interface of a Communication Channel allows passing multiple transaction requests in one call to <CommunicationRequest> (as a list). The <CommunicationRequest> is executed for a specific established connection, which is identified by its communicationReference.

The Communication Channel is expected to process the transaction requests in the order they are provided in the list. The results of the transaction requests may be passed back to the client of Communication Channel sequentially as part of the Progress callback (partial results) and the complete result shall be passed back at the end of the <CommunicationRequest> according to the extended AsyncResult pattern (see 5.8.7.2).

The relation between communication requests and communication responses can be managed by the IAsyncResult handle that is passed to a client in the call to <CommunicationRequest>. The transaction responses for these specific transaction requests will be received by that specific IAsyncResult handle. Each transaction can be identified by an ID, the same ID is provided in the transaction response.

The cancel of <CommunicationRequest> stops execution of the transaction requests. The results of already executed transactions shall be provided back to the client. For each transaction request, that has been not executed a CommunicationError “Cancelled” shall be provided to the client.

4.11.1.2 Handling of communication errors

Since it is possible to pass multiple transaction requests within one call to <CommunicationRequest>, multiple transaction responses will be provided in the result of <CommunicationRequest>. This set of transaction responses may contain a mix of positive communication results (e.g. communication data) and negative communication results (CommunicationError).

The communication client shall evaluate each of the received transaction results in order to recognize possible errors.

If a communication client receives errors for requests, that originated at that client, then the client shall inform the user. In a scenario with nested communication this means, the Gateway DTMs shall not inform the user about communication errors for requests that originally came from a Child DTM. The Child DTM has the responsibility for informing the user.

4.11.1.3 Handling of loss of connection

If the connection to a device is lost irreparably, the Communication Channel shall abort the respective connection(s) and send an Abort notification to the respective communication client for each aborted connection. The AbortMessage provides the CommunicationReference for the aborted connection. If a DTM creates multiple connections to the device, each connection is

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

managed separately, the abort of each connection may depend on specific condition depending on the used communication protocol.

The specific triggers for sending the Abort notification (e.g. device does not respond) will be defined specifically for each communication protocol.

If a Gateway DTM receives an Abort notification, then it shall send Abort notifications to all connected communication clients.

The origin of the Abort notification shall not provide a user message to inform the user about loss of connection. The communication client receiving the Abort notification shall inform the user about the loss of connection. E.g. if the communication client has open user interfaces, then the user shall be informed about the loss of connection by updating the connection status in the user interface. If the communication client is composed of several DTMs which may have open connections at the same time (e.g. a Composite Device DTM with several Module DTMs), then the communication client should avoid providing several user messages, but should inform the user about loss of connection with one message.

After sending an Abort notification the Communication Channel shall not send any further CommunicationResponses to the communication client. For all pending requests, the exception FDTConnectionAbortedException shall be thrown. The communication client should ignore any CommunicationResponse received after receiving an Abort notification.

4.11.2 Point–to–point communication

The Frame Application passes a communication interface (see interface ICommunication in Annex B) to the DTM, which provides in each case a point-to-point connection between a DTM Business Logic and a device.

It's under the control of the Frame Application to enable a DTM to communicate with its device. The Frame Application has to provide the communication interface to be used to the DTM by calling the method IDtm3.EnableCommunication2()(see definition in Annex B) and thus allowing communication access.

In order to access the device, the DTM uses this interface as shown in Figure 20.

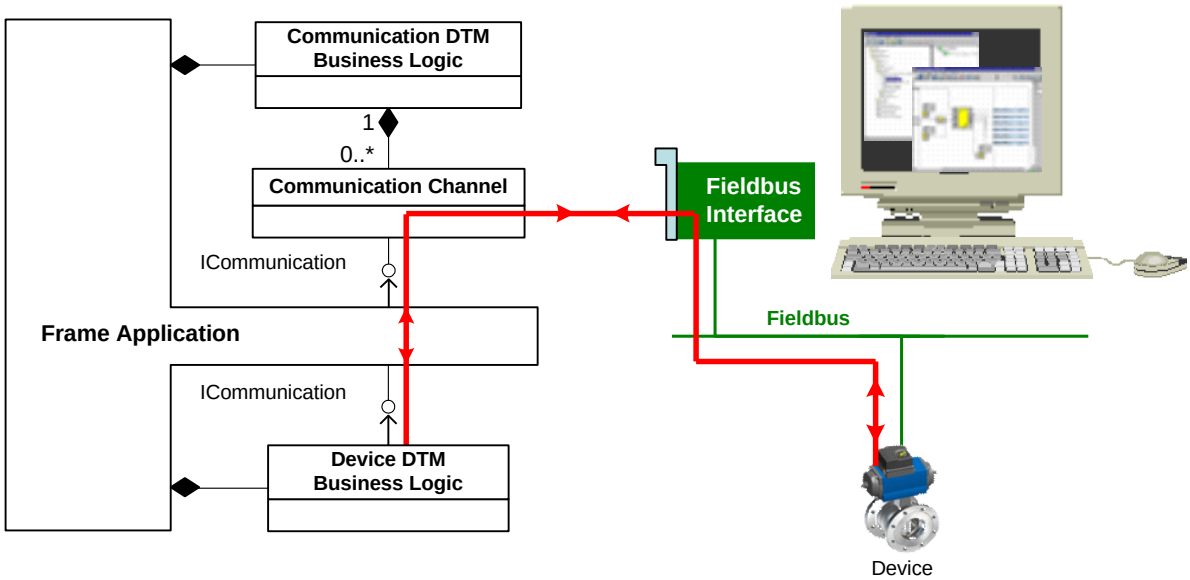


Figure 20 — Point–to–point communication

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

The Frame Application starts the corresponding Communication DTM and forwards the communication requests to the Communication Channel which then communicates with the hardware.

A DTM has to call the Communication Channel method ICommunication.<Connect()> (see definition in Annex B) in order to establish a communication connection to the device. After the connection has been established the DTM is able to communicate to the device by calling the ICommunication.<CommunicationRequest()> (see definition in Annex B). It is general expectation, that a DTM tests if it is connected to the intended device. See also 8.6.2.

DTMs shall consider the limited availability of fieldbus resources when connecting to the device. This means a DTM should create only as many connections as needed and connections shall be terminated if they are not used (e.g. the online function is finished).

4.11.3 Nested communication

A DTM shall not need to know anything about the system topology in order to access the corresponding physical device. Thus, the same DTM is able to communicate with devices attached to different fieldbus-interfaces of the same protocol type. Nested communication is used if the devices are connected to a sub system, for example if a device is connected to a channel of a Remote I/O (see Figure 21).

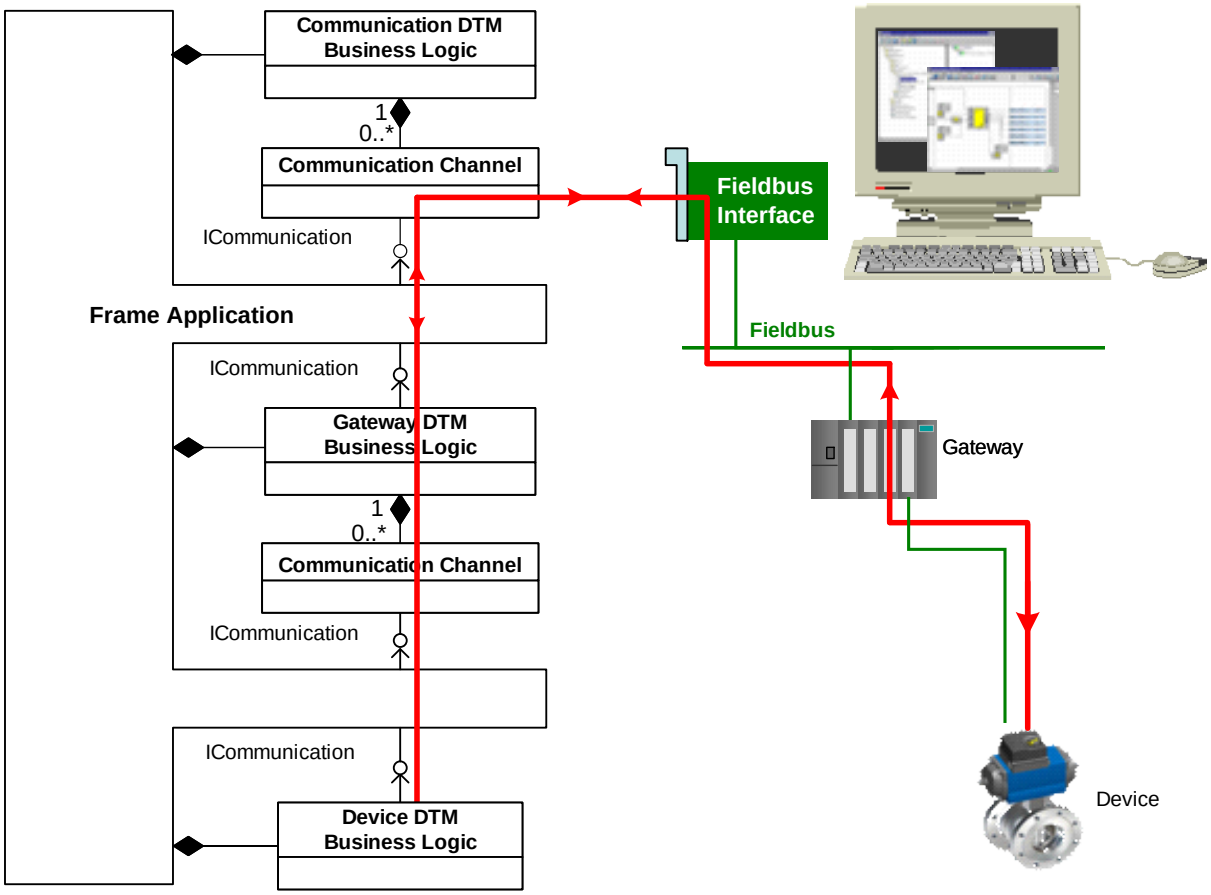


Figure 21 — Nested communication

Like in the point–to–point communication all DTMs simply use the communication interface with their devices without the awareness of the nested communication

The communication in FDT3 follows the concept of nested communication:

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

- Each Communication Channel wraps the communication message from the linked DTM below, without knowing the content
- The linked DTM below the Communication Channel does not have any knowledge about the system topology
- The DTM shall only support communication protocols that are supported by the respective device
- Communication / routing through any system topology is possible

See 8.7 for sequences related to nested communication.

4.11.4 Dynamic changes in network

Many fieldbuses provide a mechanism for temporarily disconnecting devices or switching between distinct groups of devices during operation (e.g. tool change for robots, docking/undocking of transportation vehicles). Such mechanisms lead to changes in the communication network (called “dynamic configurations”) – devices may be disconnected. The Frame Application (and the network configuration tool as part of the Frame Application) are able to manage the current device states at the DTM (see NetworkDataInfo.DeviceMayBeDisconnected and NetworkDataInfo.DeviceIsDisconnected in Annex B).

4.12 Identification

4.12.1 DTM instance identification

4.12.1.1 System Tag

An FDT Frame Application shall assign a unique identifier for each DTM instance. This unique identifier is referred to as “System Tag”. The System Tag is used by DTMs:

- for navigation in the FDT topology,
- for the management of Child DTMs in the FDT topology (e.g. address setting), and
- to identify a DTM instance at the event interface of the Frame Application

The System Tag is defined as GUID.

4.12.1.2 Assignment of System Tag

Following rules apply to assignment and use of System Tag:

- A Frame Application shall not change the value of the System Tag of a DTM instance during the complete lifecycle of a DTM instance. This means the same system_tag_value is used to identify the DTM instance in all interfaces (e.g. IChildDtmEvents, ITopology and ISubTopology).
- When a project is persisted the Frame Application shall save the System Tag of the DTMs such that they will be the same system_tag_value when loading as before.
- A Frame Application shall use the same system_tag_value only for DTM instances associated to the same Device Node (see Figure 25). This means it is not allowed to reuse system_tag_values for other DTM instances.
- A DTM shall not persist the value of its own System Tag.

Since the System Tag uniquely identifies a DTM instance, it is possible that DTMs store System Tags as references to other DTMs. For example, if a Parent DTM needs to keep track of its children and the data they expose (e.g. for Address Setting or Busmaster Configuration), then the Parent DTM may cache information published by its children. The Parent DTM can store the cached information using the System Tag as a key.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

NOTE: If multiple users work on the same Device Node, each user has an own instance of the DTM, but all DTM instances use the same System Tag.

4.12.2 System GUI label

The DtmSystemGuiLabel is a human-readable identifier, used to identify a DTM instance in the project and the related DTM GUI in the display to the user. The Frame Application and the DTM should use the DtmSystemGuiLabel in order to identify dialogs and windows belonging to the DTM instance.

The DtmSystemGuiLabel may reflect an identifier of the device (e.g. the device tag) that is provided in NetworkDataInfo.Label. The DTM may use NetworkDataInfo.Label to provide an initial value for DtmSystemGuiLabel. It is recommended for a FA to use the NetworkDataInfo.Label value, when constructing the DtmSystemGuiLabel.

4.12.3 Hardware identification

A DTM supports the method IHardwareInformation.<HardwareScan()> (see definition in Annex B) that enables to read device information online from the connected device (see Figure 22).

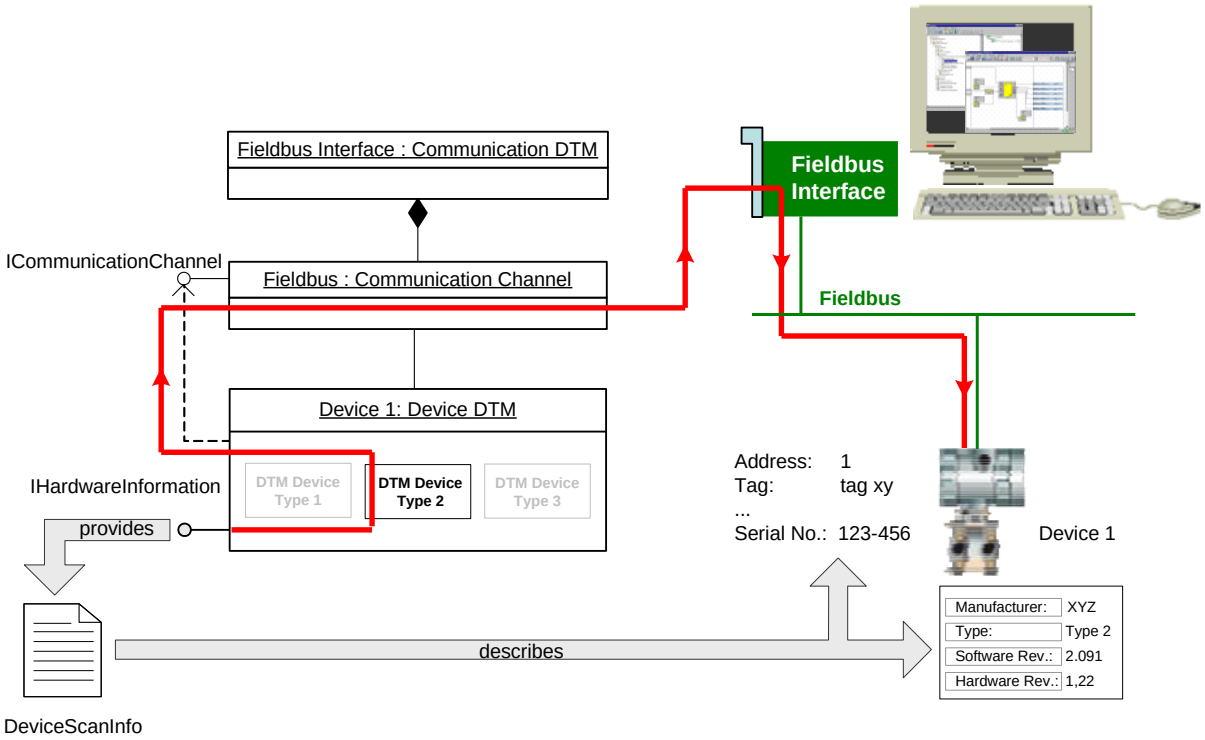


Figure 22 — Identification of connected devices

The service returns device type related information like manufacturer, type, hardware and embedded software version as well as device instance related information like fieldbus address, tag and serial number. This information is also fieldbus specific like the information returned by IDtmInformation.GetDeviceIdentInfo() (see definition in Annex B). Therefore, the concrete protocol-specific format is defined by the protocol-specific documents. The transformation to protocol-independent format is implemented by the protocol-specific datatypes. See examples in 7.5.

4.13 Scanning and DTM assignment

4.13.1 Scanning introduction

Many fieldbus protocols (or point-to-point communication) define scanning services supporting to request a live list that contains information about connected devices. The information and

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

services described in subsequent clauses enable the Frame Application to generate or validate corresponding FDT topology without the knowledge of the protocol-specific definitions.

4.13.2 Scanning

Scanning is supported by the Communication Channel method IScanning.<ScanRequest()> (see definition in Annex B).

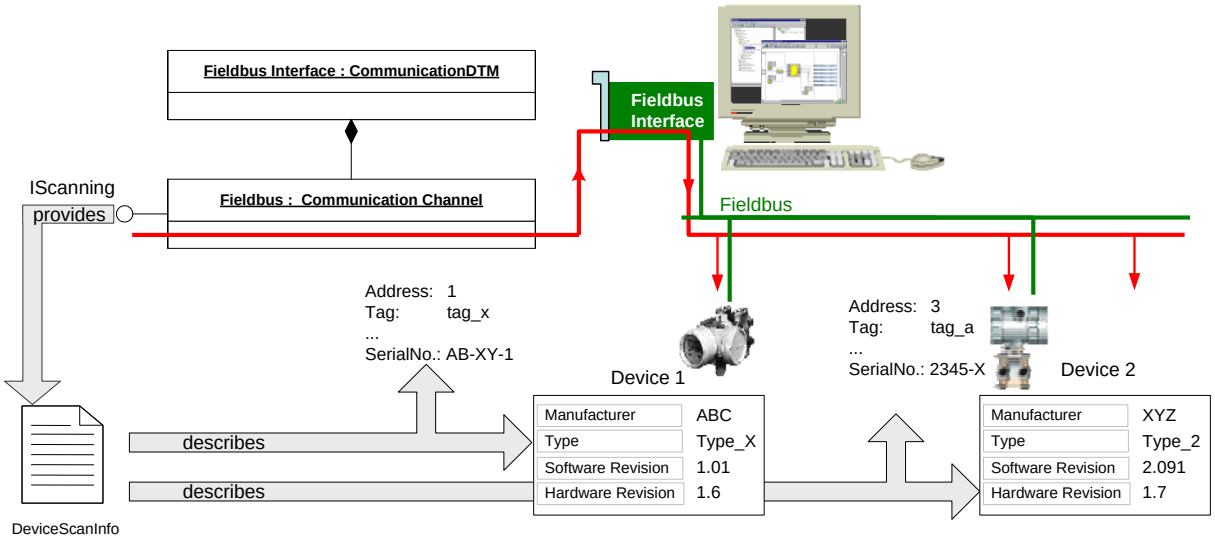


Figure 23 — Fieldbus scanning

The service returns device type and instance related information for all connected devices which can be determined by the means defined by the fieldbus protocol (see Figure 23). For example, device manufacturer, type, hardware and embedded software version or fieldbus address, tag and serial number. This information is fieldbus specific. The concrete format and transformation to the protocol-independent format which can be used by Frame Application without fieldbus-specific knowledge is defined by the FDT Protocol Annex document describing the protocol profile integration in FDT.

4.13.3 DTM assignment

A Frame Application may use the information returned by an IScanning.<ScanRequest()> (see definition in Annex B) to find proper DTMs that can be added to the FDT topology by comparing it with the device type identification information returned by service IDtmInformation.GetDeviceIdentInfo() (see definition in Annex B) of known DTMs.

Furthermore, a Frame Application may also use the scan result information to validate whether DTMs in an existing FDT topology are the proper ones. This validation is executed by comparing the scan result information with the device type identification information returned by those DTMs. Each element from the scan information can be compared with the DTM device identification information. In this comparison a Frame Application may also apply specific rules.

4.13.4 Manufacturer-specific device identification

Sometimes devices cannot be uniquely identified by the means defined by the fieldbus protocol. For example, because same type ID is used for several different device types of one manufacturer. In this case the Frame Application may find several DTM Device Types or even DTMs that appear to be the proper ones for devices found during a scan. However, such device may be identifiable by means defined by the device manufacturer.

FDT enables the manufacturer to implement this specific device identification within a DTM and to allow a Frame Application to use it in a standard way, independent of manufacturer and protocol.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

The DTM for such a device shall add manufacturer-specific identification properties to the device identification information that is returned by IDtmInformation.GetDeviceIdentInfo() (see definition in Annex B). If the Frame Application tries to find a proper DTM, then it may find only device identifications which match partially but have additional elements. The additional elements are not included in the scan result, because they are known only by the device manufacturer. In such a situation the Frame Application shall look for a DTM Device Type for which the DeviceIdentInfo.SupportLevel property is set to 'IdentSupport' (see definition of DeviceIdentInfo in Annex B). The 'IdentSupport'-DTM is added temporarily to the FDT topology in order to request additional information from the device. This additional information can be requested by the method IHardwareInformation.<HardwareScan> (see definition in Annex B). The DTM then shall execute the manufacturer-specific device identification and return the additional device identification elements as result of the method IHardwareInformation.<HardwareScan()>. This enables the Frame Application to execute a comparison, which covers all device identification information, have a full match and to replace the 'IdentSupport'-DTM with the correct DTM and the matching DTM Device Type.

4.14 DTM data persistence and synchronization

4.14.1 Persistence overview

The Frame Application is responsible for the persistent storage of data (data persistence). This includes topology information as well as data managed by the DTM itself (e.g. device parameters). FDT defines only the interfaces, which shall be used by the DTM for data persistence (see Figure 24). While the implementation of the persistent storage system is specific for a Frame Application, the format of stored data is specific for each DTM. Both are not in scope of the FDT specification.

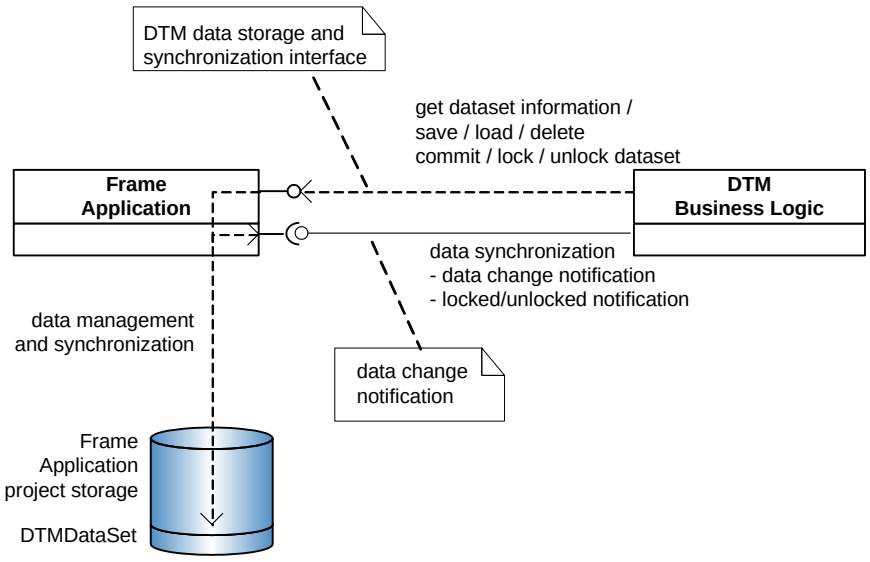


Figure 24 — FDT storage and synchronization mechanism

The Frame Application storage interface provides the DTM Business Logic methods to access its dataset (called DTMDataSet) in the Frame Application storage implementation, e.g. in a database or file persistence.

The Frame Application has to guarantee the data consistency for multi-user and multi-client data access and provides corresponding methods and events to the DTM Business Logic.

4.14.2 Relations of DTMDataSet

The Frame Application manages for each physical device one DTMDataSet and the related DTM instance as shown in Figure 25.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

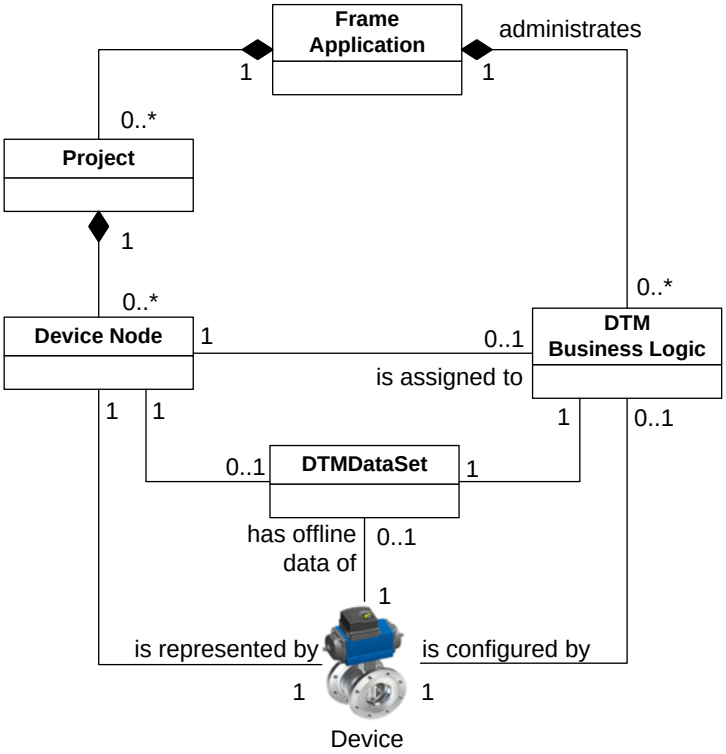


Figure 25 — Relation between DTMDataSet, DTM instance, and device

Note: For multi-user scenarios, the multiplicities on the DTM Business Logic all have an upper limit of 'many' (see 11.4).

The Project is part of internal model of the Frame Application. It is an abstract, logical object used here to describe the management of device-instances. FDT does not define any interfaces for the Project object, since it's a pure Frame Application internal object and may have different specific implementations.

The Device Node also is part of internal model of the Frame Application. It is an abstract, logical object used here to represent a physical device in the Frame Application. It controls the lifetime and data of device-instances within a Frame Application. FDT does not define any interfaces for the Device Node object, since it's a pure Frame Application internal object and may have different specific implementations.

A Frame Application typically (vendor-specific) saves DTMInfo and TypeInfo information of the corresponding DTM (see 7.4) together with the DTMDataSet to be able to start the DTM Business Logic, which originally saved the data.

4.14.3 DTMDataSet structure

Figure 26 shows the structure and content of a DTMDataSet.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

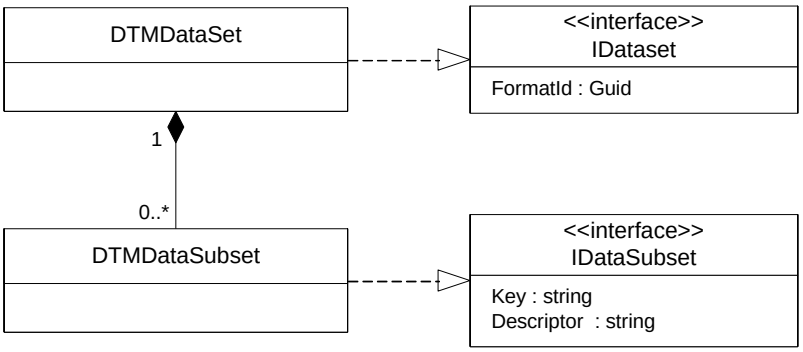


Figure 26 — DTMDataSet structure

The DTMDataSet has a property FormatId, which is a unique identifier for the format of the data. This ID is created by the device (DTM) vendor. The DTM Business Logic can use this information to decide how to load the data, e.g. to migrate the data from an older version.

A DTM always writes the DTMDataSet in one specific format but may be able to read also other data formats. In such a case a DTM can declare to support more than one FormatId. If different DTMs declare to support the same FormatId the following scenario can be supported:

A DTM vendor can provide a scenario to migrate the data from an old version of a DTM to a newer DTM version. The new DTM version declares to support the old FormatId as well as the new FormatId. The Frame Application detects the old FormatId and creates the new DTM. The new DTM loads the DTMDataSet, migrates the data and saves the data with the new format (identified by a new FormatId).

Note: This scenario may work for DTMs of one vendor or for DTMs from different vendors. However, the definitions necessary to support such a scenario are out of scope of this specification.

A DTMDataSet can have one or more DTMDataSubsets. The DTMDataSubsets contain the persistent data of a DTM. The DTM Business Logic can explore the DTMDataSet and add or remove DTMDataSubsets to/from the DTMDataSet. The DTMDataSubsets are identified by an ID which is created by the DTM. The DTM can use the IDs to read or write the data.

Which data is stored in one DTMDataSubset is DTM-specific. The DTM should group data in one DTMDataSubset if it belongs to one functional unit and needs to be loaded together. In order to improve the system performance a DTM should avoid loading of unnecessary data whenever possible, especially at start-up of the DTM. Following grouping should be considered:

- Basic data which is needed during the complete lifetime of a DTM instance (e.g. Network Management Info)
- Device parameter group information which is needed if corresponding DTM WebUI is opened (e.g. a page in a dialog) or if the Frame Application requests data (e.g. DeviceDataInfo objects (see 7.9))
- Process data information which is needed if Frame Application requests ProcessDataInfo objects (see 7.11.1)
- etc.

The DTMDataSubset data format is DTM specific. Any serializable datatype can be used. The Frame Application is not allowed to modify the data.

4.14.4 Types of persistent DTM data

Two types of DTM related data are considered:

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

- Instance-related data (called “instance data”). Instance-related data belongs to the DTM itself. It is specific for a DTM which data it stores but the DTM has to guarantee that it is able to represent the stored device instance by loading these data;
- Bulk data. DTM-specific data, for example historical data. A DTM can save bulk data as separate DTMDataSubsets in the DTMDataSet in the same way as instance related data (each in a separate collection). Configuration data shall not be stored in the bulk data. DTM shall be prepared to be loaded without previously stored bulk data.

Instance-related data and Bulk data may be stored in separate storages in order to allow a Frame Application to distinguish instance related DTMDataSubsets and bulk DTMDataSubsets for management purposes.

4.14.5 Data synchronization

If multiple users access the same device, systems shall start several DTM instances of the same DTM type and for the same physical device (see 4.8.2). The different DTM instances access the same DTMDataSet. This is for example the case in a distributed system where multiple users access the same Frame Application (see 11.4).

Note: In any case the Frame Application should never start multiple DTM WebUIs for different users accessing the same DTM BL instance.

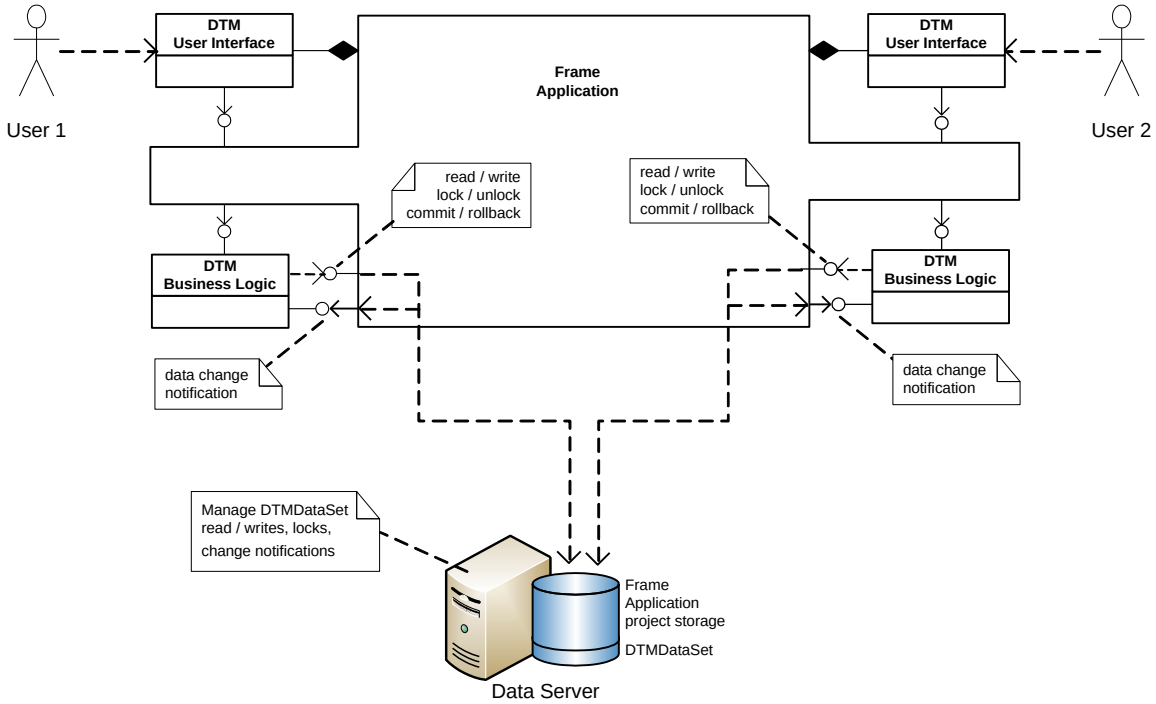


Figure 27 — Data Synchronization

To support such a scenario the Frame Application shall support interfaces, which allow realizing a dataset locking and changing notifications concept. (see Figure 27)

FDT3 uses a pessimistic locking concept on DTMDataSet (device) level. The concept works as following:

- A DTM shall try to lock its DTMDataSet before execution of an operation that may lead to data changes (e.g. opening of a parameterization user interface).
    - If locking was successful, then data changes are allowed
    - If locking failed, e.g. because another DTM instance has already locked the data, then no data changes are allowed (e.g. opened user interface shall disable input fields)

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

- A DTM that has no lock can only read the last committed data from the DTMDataSet
- A DTM that has the lock can read and write the data in the DTMDataSubsets.
- Changes in the DTMDataSet are only visible to the DTM that holds the lock until DTM commits the changes and until the Frame Application sends TransactionCommitted to other DTMs.
- Uncommitted changes are automatically discarded if the DTM unlock the DTMDataSet
- The Frame Application notifies all other DTM instances working with the same DTMDataSet if
    - Data in a DTMDataSubset has changed and changes are committed (DTM should re-read and display the data)
    - DTMDataSet is locked or unlocked (DTM should change the state of UIs, e.g. input fields are enabled / disabled)

4.15 Device data and IO information

4.15.1 Exposing device data and IO information

In addition to device-specific functions and user interfaces a DTM provides access to device data and to instance data via the programming interface(see definition of IInstanceData, IDeviceData, DataInfo, Read-Write Request and Read-Write Response in Annex B).

It is recommended, that a DTM exposes all parameters, which are accessible in the user interfaces of the DTM, also by IInstanceData and IDeviceData. A DTM shall expose at least all parameters defined in applicable profiles of FDT Protocol Annexes and FDT Application Profile Annexes. If a DTM is supporting a device with a device description (e.g. EDD or EDS), parameters should be exposed with the same name and label as in the corresponding DD (for example EDD: parameter name should be the identifier of the corresponding EDD-Variable).

Device data and instance data may expose different sets of parameters. Device data may expose dynamic data like process value, device status and operating hours, whereas instance data should not expose such dynamic data.

The DTM shall adapt the list of exposed parameters according to the user level (e.g. restrict access to parameters). DTMs shall update the list of exposed parameters during runtime, for instance when parameters become inaccessible due to a changed configuration (e.g. changed measurement principle). Parameters, which are only available if the DTM is in an OEM service mode, shall not be exposed.

The DTM should expose the data in DataGroups in the same organization as presented in DTM WebUIs. Parameters shall be exposed in a way (format and semantic information) which allows the processing of the data without fieldbus knowledge. For example, instead of raw data in hex format, a parameter shall be exposed as readable value with a numeric datatype and provide additional information like unit and range.

NOTE: The intention of this recommendation is to unify the support based on IData interfaces and the support provided by the DTM GUI.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

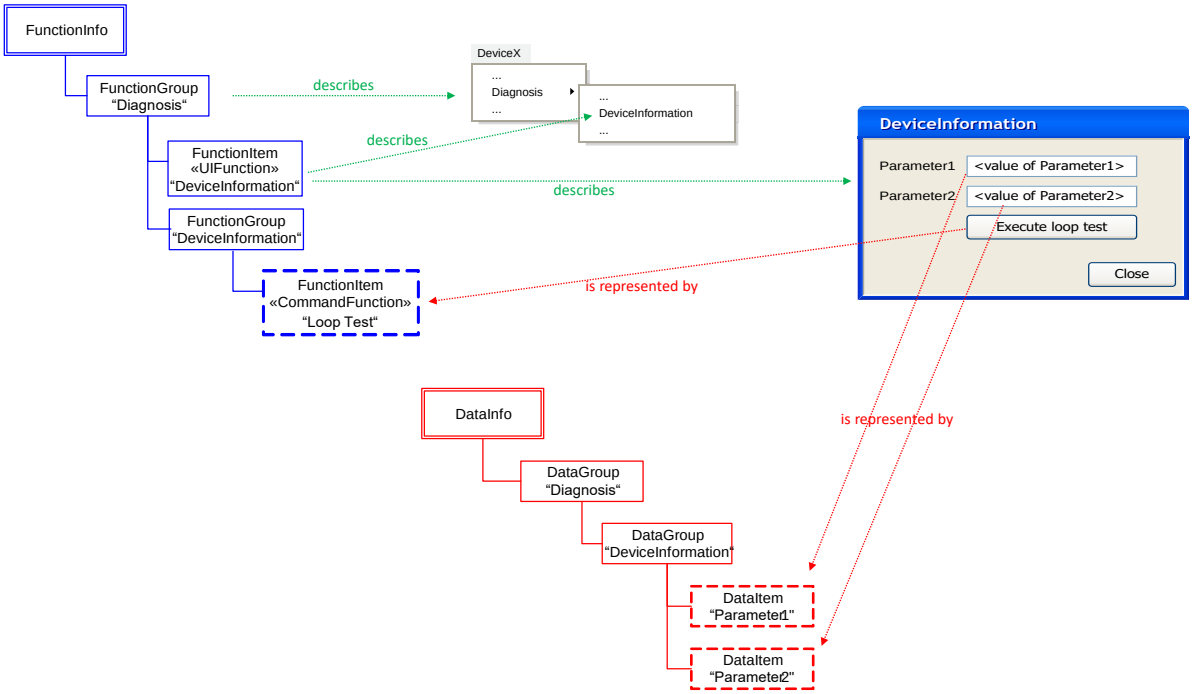


Figure 28 — Example for same organization of FunctionInfo and DataInfo

FDT Protocol Annex specifications may define additional requirements regarding the exposed device data.

A Frame Application may use the exposed data for various use cases, for instance for comparison. For examples see the following sections.

4.15.2 Data access control

A DTM may support configuration of access control by the Frame Application with the interfaces IDeviceCustomConfiguration/IInstanceCustomConfiguration. If a DTM supports this interface, it shall provide a non-changing list of all functions at the IFunction interface. Based on this complete list of available functions, the Frame Application may restrict access to command and DTM WebUI.

The Frame Application can use IDeviceCustomConfiguration/IInstanceCustomConfiguration interfaces to read the description for all data and all data groups exposed by the DTM independent of the current settings of the device and independent of the mode of operations. The Frame Application can use the information to present the list of exposed data, the name, label, descriptor, read/write status and semantic information to the user and let the Administrator create custom access permissions for each user or group of users. The Frame Application can enable access to individual data using the method <EnableParameters> with a list of IDs for all parameters that shall be changeable.

Each data item is represented by an object of class AccessibleData. This class defines properties related to data access control (Table 3).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 3 — Description of properties related to data access control

AccessibleData Properties	Description
IsReadable	Specifies whether the value can be read from the device or DTM instance. The value may change depending on the internal business logic of the device / DTM.
IsWritable	Specifies whether the value can be written to the device or to the DTM instance. The value may change depending on the internal business logic in the device / DTM.
IsChangeEnabled	This attribute is applicable only when the DTM has been initialized with Expert user level. It specifies whether the FA has enabled changes to the data for the current user. This property controls what can be changed directly by the user through the DTM-UI or through the methods called by the Frame Application. TRUE: Allows the parameter to be changed by the FA using IDATA or by the user through the DTM WebUI FALSE: Parameter access is restricted, and the value cannot be changed by the Frame Application using IData interface or by the DTM WebUI The Frame Application shall verify that the values of IsChangeEnabled and IsWritable attributes are both set to TRUE for the parameter to be writable in the DTM.

The “IsChangeEnabled” property value is provided by the DTM and can be set only by the Frame Application. It cannot be changed by the DTM. The IsChangeEnabled property shall be set to FALSE by default by the DTM for the user with Expert user level. The value of IsChangeEnabled property shall be ignored by the DTM and by the Frame Application when the user level is Observer or Engineer.

The Frame Application can enable the change of a data item by setting the “IsChangeEnabled” flag to TRUE. Setting the IsChangeEnabled flag to TRUE is required to allow the change of the data item in the DTM/ device. The device/DTM may have additional restrictions, e.g. the data or data group may remain read only, the value of a data item may be restricted by the value of other data items, the data item may be read only in the device, etc.

When “IsChangeEnabled” attribute for a data item is FALSE, the data item cannot be modified by the Frame Application or by the user through the UI of the DTM. It is not expected that “IsChangeEnabled” attribute will change the visibility of a parameter in the user interface of the DTM, but the DTM shall present the value as read only if “IsChangeEnabled” is set to FALSE.

When a parameter value is set in the DTM BL, it may apply additional internal logic and modify the values of the related parameters even if the “IsChangeEnabled” flags for those related parameters are set to FALSE. The user will be able to see the modified values for the related parameters but will not be able to modify their values since the “IsChangeEnabled” flag is set to FALSE. In a similar way, if the parameter value is set in the device, the device may change multiple parameter values of dependent parameters even if these parameters cannot be modified directly. This means that the “IsChangeEnabled” flag is only used to control the modification of data items by the user or by the FA, not by the DTM or the device itself.

When the FA wants to set the “IsChangeEnabled” flag to TRUE for a data group, it has to set the “IsChangeEnabled” flag to TRUE for each of the data items in the respective group. If a DTM has a user interface that shows a group of parameters, it is recommended to create the user interface in a way, which allows to control which parameter in the group is changeable and which is non-changeable. If the DTM cannot control the access to parameters of a group individually, then the entire user interface may be enabled for change for all parameters of the group if one of the parameters in the group is changeable.

IData exposes the list of parameters according to the actual status or device mode. However, the DTM shall expose all parameters independent of state or device mode or role through <GetAllDataInfo()> method. The Frame Application will provide the list of changeable parameters to the DTM by calling the <EnableParameters()> method. The DTM shall apply the

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

“IsChangeEnabled” values set by this method to the individual data groups and data items. The settings shall be applied to all parameters, independent of the device mode.

The <EnableParameters()> methods (for instance data items and for device data items) shall be called only once in running state before any function or any other method is invoked in the DTM. Once set, the DTM shall preserve the settings for "IsChangeEnabled" flag during the lifetime of the DTM instance and shall reject any other request to <EnableParameters()>.

The DTM shall not save the value of the IsChangeEnabled flag in its instance data set. It shall initiate the flag to the default state ("IsChangeEnabled" = FALSE) any time a new DTM instance is created and initialized with Expert user level. The Frame Application shall invoke <EnableParameters()> each time a new instance of the DTM is initialized with Expert user level.

There might be device data or instance data that cannot be exposed as parameter and thus ICustomConfiguration interface cannot be used to modify the "IsChangeEnabled" property in the Expert user level. By default, the DTM is expected to create this data as non-changeable and the Frame Application will not be able to make it changeable.

4.15.3 Routed IO information

If a device (for instance a gateway device) delivers IO signals that originated from a connected device, then the IO Signal Info items of the ProcessDataInfo (see 7.11.1) returned by corresponding Gateway DTM Business Logic shall describe this relation (see Figure 29).

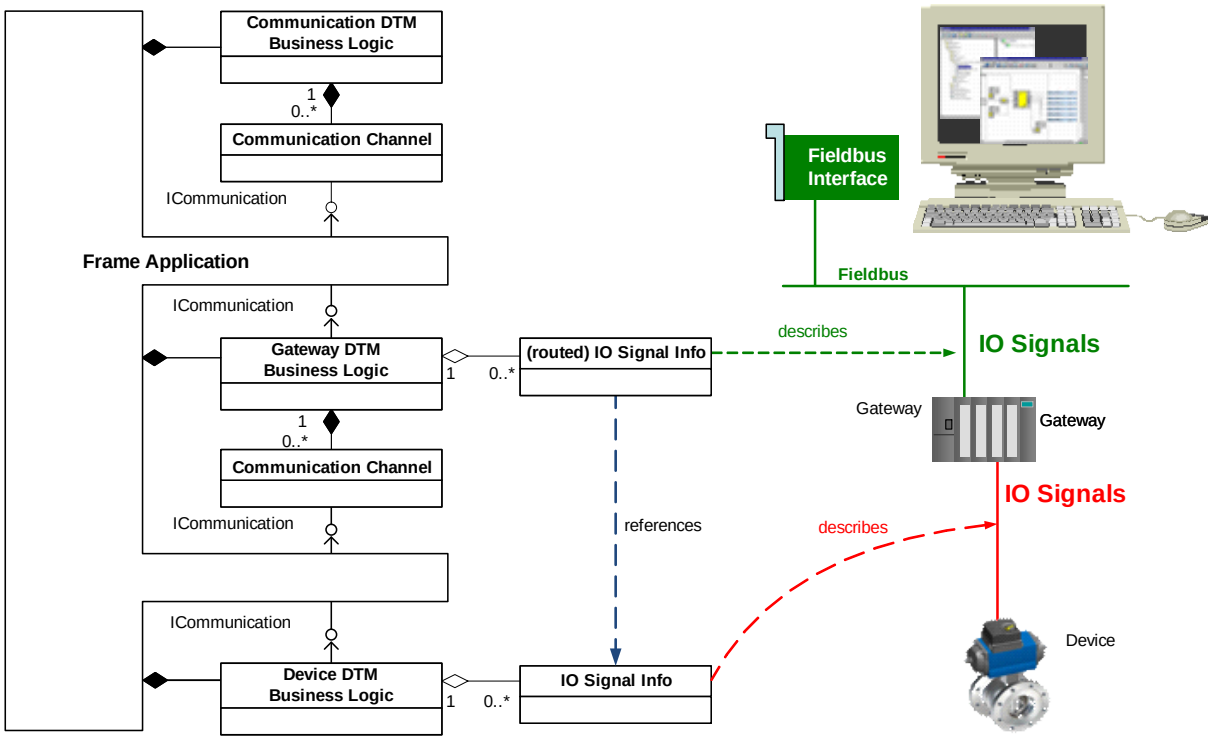


Figure 29 — Routed IO information

The IO Signal Info items of the Gateway DTM shall reference the IO Signal Info items of the Device DTM by the SystemTag of the Device DTM and the Id of the IO Signal Info.

4.15.4 Comparison of DTM and device data

FDT supports comparison of DTM and device data, for example:

- Comparison of persisted data with data in the device
- Comparison of historical data with current data

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

- Comparison of data from different devices

In order to support these scenarios, FDT defines two alternative comparison concepts:

5. DTM publishes all data in the corresponding interfaces. In this case the Frame Application is responsible to perform the comparison (see 5.15.1).
6. DTM provides the comparison interface. In this case the Frame Application shall call this interface for the comparison. (see 5.15.2)

4.15.5 PLC tool support

4.15.5.1 General

In a typical PLC system, a fieldbus master hardware periodically communicates the IO signals with the connected devices and provides them to the PLC in a shared memory area. This memory area is named process image (see Figure 30, simplified by showing only input data). Individual IO signals are addressed by the PLC application with an offset into this image, while the overall layout of the image is controlled by the fieldbus master configuration software which can be a DTM.

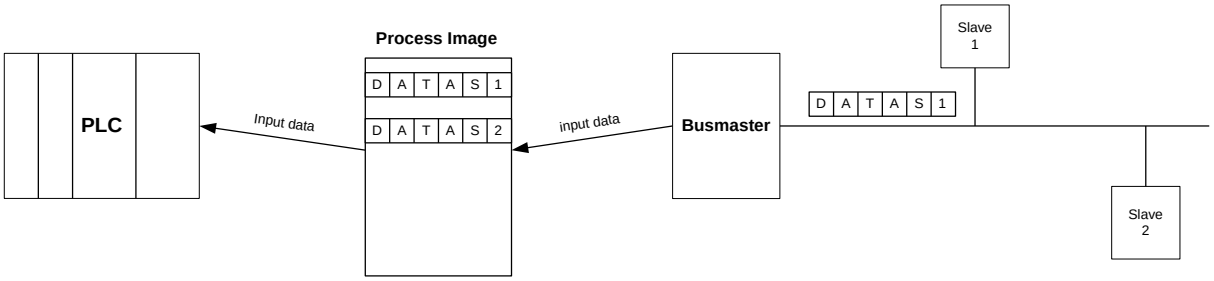


Figure 30 — Process Image

A DTM which represents the fieldbus master shall provide the IProcessImage interface. This interface enables a PLC Tool Frame Application to request the process image information from the Fieldbus Master DTM (see Figure 31). This information is used for the mapping of variables within a PLC program to the IO signals in the process image.

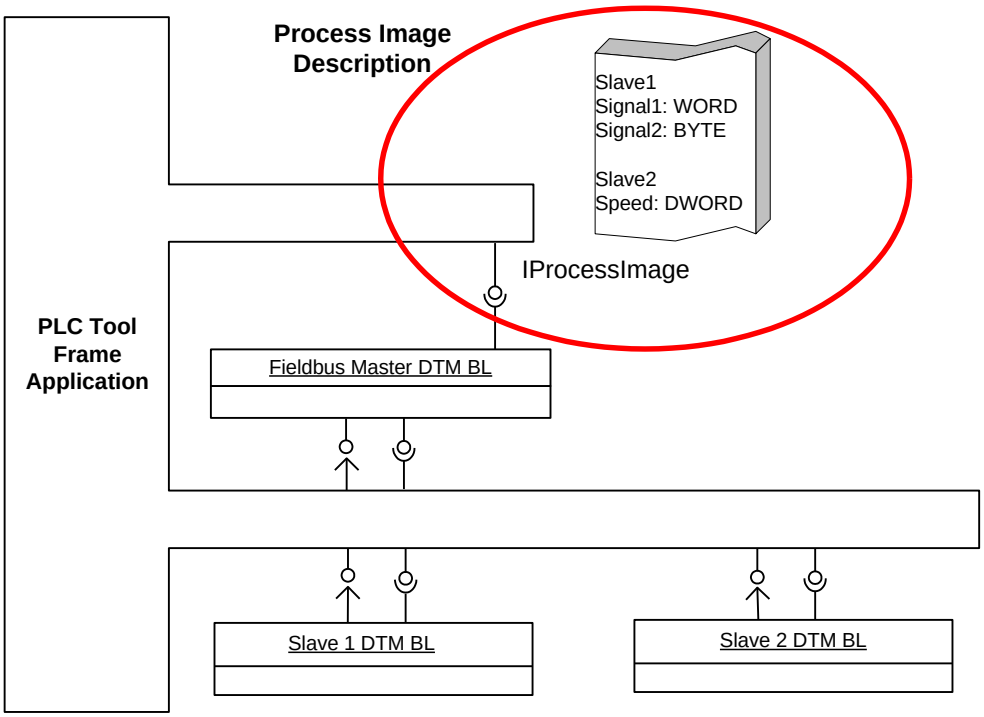


Figure 31 — Transfer of layout information using IProcessImage

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

The basic process image information representation is fieldbus-neutral (see 7.11.2). This allows PLC Tool Frame Applications to do the variable assignment independent of the underlying fieldbus system. In addition, fieldbus-specific IO signal information (see 7.11.1) is also included in the process image information. This information is gathered by the fieldbus master DTM from the Child DTMs representing the connected devices using their IProcessData interface (see 8.12).

4.15.5.2 Process Image Modifications while PLC is running

In some automation systems it is a requirement to apply changes to the layout of the process image while the PLC is running (executing control), because the plant process cannot be stopped.

PLC Tool Frame Applications which support such modifications at runtime need to assess if a change can be applied without stopping the PLC. That means, if a user makes a configuration change which would lead to a change in the Child DTM Network Information (see 7.13) then this change should be verified before it is applied to the respective field device. This enables the PLC Tool Frame Application to reject changes which would require to stop the PLC if this is not possible at this moment.

The validation if the PLC needs to be stopped can only be done in the fieldbus master DTM and the PLC Tool Frame Application itself. Thus, the Child DTM should support validation of changes in the Network Information by calling the INetworkInfoValidation interface of its parent. The Fieldbus-Master DTM can then redetermine the resulting process image and validate if it can be applied using the IProcessImageValidation interface implemented by a PLC Tool Frame Application (see 8.12.3).

The validation needs to be done before the changes are applied. Only if validation is successful the Child DTM is allowed to apply the changes.

4.15.6 Support for multirole devices

4.15.6.1 General

Next to Master/Slave Gateways, which are described by Gateway DTMs, Slave/Slave Gateways do exist. As these kinds of devices do not open a new type of communication, they are modeled as Device DTMs or Module DTMs which may be part of more than one logical topology. Both slave roles may support the same or different bus protocols.

Furthermore, some bus protocols allow sharing of devices and/or modules between multiple masters. These shared devices are part of multiple topologies, too.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

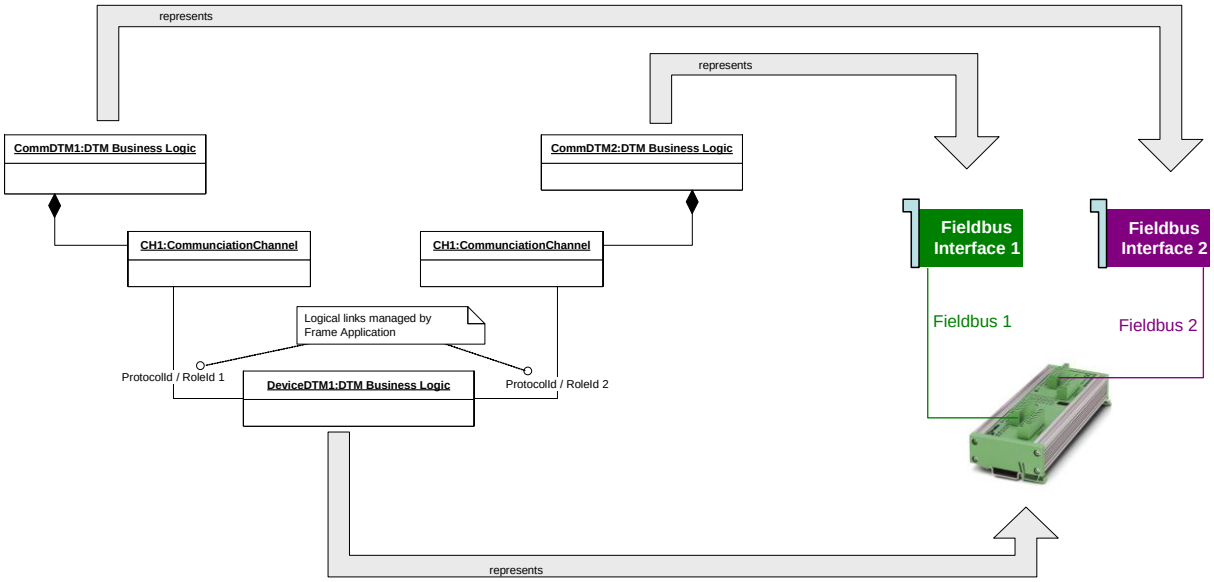


Figure 32 — Multirole Device

Different roles are assumed in different topologies, e.g. the same device may act in one topology as slave and may act as master in another topology. In one topology only data relevant for a certain protocol or for the respective role is of interest to the Frame Application.

It is the responsibility of the Frame Application to handle the instantiation and release of one DTM in multiple topologies.

4.15.6.2 Accessing multirole related data

The support of multirole devices is optional for Frame Application and DTMs. If both support multirole devices, instead of direct access to interfaces at the DTM, role related data can be accessed by way of the IDtmRoleAccess and IDtmProxyRoleAccess interfaces (see definition in Annex B). It is in the responsibility of the DTM to provide role related data only when accessed by the IDtmRoleAccess or IDtmProxyRoleAccess interface.

If a DTM accesses Sibling DTMs, which provide role related data, the accessing DTMs should support access via role access interfaces.

4.16 Clone of DTM instances

4.16.1 General

A Frame Application may offer the functionality to copy a part of the FDT topology (i.e. multiple DTMs) e.g. for “copy and paste”.

If a part of the FDT topology is copied, then the System Tag for all cloned DTMs of the copy shall be changed by the Frame Application. Otherwise the System Tag would not be unique any more.

To create a cloned DTM instance the Frame Application shall perform following steps:

- Copy the DTMDataset to the new device node
- Create a new DTM instance using the same DTM (unique class identifier).

Depending on the use case the Frame Application should ask the user to set the correct fieldbus address in the DTM, to set a correct TAG and to adjust DTM offline parameters before the dataset is downloaded, for example:

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

- device-position-specific parameters like settings related to mounting related settings
- device-instance-specific parameters like device calibration, linearization

A DTM shall reset cached online parameters (e.g. device serial number, operating hours etc.) and consider removing bulk data subsets when LoadData() is called with argument isCloned set true. The Frame Application applies the argument isCloned to all DTMs involved in the cloning operation.

If a Parent DTM is storing the System Tags of its children then these are invalid after the Parent DTM was cloned.

If a Parent DTM instance is cloned and has cached the System Tags of its children, then it shall rebuild its internal data structure based on the list of changed topology nodes passed to IDtm3.LoadData().

4.16.2 Replicating a part of topology with Parent DTM and a subset of its Child DTMs

Cloning of a DTM with only some of its children is not supported. A Frame Application should not offer this function to the user. This restriction is to avoid inconsistencies. If a Frame Application would offer this functionality, then the rules which are implemented in ISubTopology.<ValidateRemoveChild()> could not be applied.

4.16.3 Cloning of a DTM without its children

If a DTM which has children is cloned without its children, then the internal data structure used to manage children most likely is invalid. If IDtm3.LoadData() has an empty list of changed topology nodes, then a Parent DTM needs to release the complete set of data associated to its children (See section 8.18.1 for the workflow).

4.16.4 Delayed cloning

If a Frame Application allows delayed cloning (“copy” the DTM, then make changes to the topology, then paste the DTM) then a Parent DTM is responsible to ensure the consistency of its internal data structure used to manage children. This is done by keeping track of the topology via ISubTopology.<ValidateAddChild()> and ISubTopology.<ChildAdded()> (see section 8.8.2 for the workflow).

4.17 Lifecycle concepts

Automation systems in process industry typically have a life time of 10-15 years or more. Over time hardware and software components in a system will be exchanged, which may require updates or upgrades of FDT related components.

The FDT3 life cycle concepts rely on identification and versioning of components which may change during the plant lifetime.

The concept defines rules to identify software and hardware components and rules to ensure backward compatibility of a component from one version to another.

The general lifecycle guidelines and the implementation of lifecycle concepts with FDT3 is described in section 10 of this document.

4.18 Audit trail

4.18.1 General

Audit trail is about recording who has accessed an automation system and what operations were performed during a given period of time. FDT defines Frame Application services which shall be used by the DTM to record operations performed on the associated device.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Frame Applications can use this information for:

- recording the information, date and time of operator entries and actions
- generating the records, e.g. for inspection and reviews
- evaluating the system

These features are for example needed for a Frame Application to comply with FDA [18] guidelines.

4.18.2 Audit trail events

A DTM shall send an Audit Trail notification to the Frame Application to record any changes in the device. DTMs shall only send notifications for changes and not for internal state transitions (e.g. the instantiation of a user interface shall not trigger Audit Trail events).

The following notifications are defined:

- Function Notification: Notifies the Frame Application that a function was called (e.g. self test functionality of the device or download was executed).

  A function notification shall indicate the start of a function and the end of a function. A notification about the end of a function shall contain the information if the function was executed successfully, cancelled or executed with a failure. A DTM shall also fire notifications for operations which are triggered by the Frame, e.g. Download parameters.

  The notifications related to functions are:

  - 'Function_name' started
  - 'Function_name' finished successfully
  - 'Function_name' finished with error 'error_reason'.
  - 'Function_name' cancelled by user
- Parameter Change Notification: Notifies the Frame Application that a parameter was changed. Contains the old value of the parameter as well as the new value.

  A DTM shall group notifications which belong to one logical operation (changes set) into one single notification. This means that there shall be e.g. one single notification for the complete set of parameters which are part of a download.

  The notification related to parameter change is:

  - Parameter ‘Parameter_name’ changed from ‘old_value’ to ‘new_value’.

If the DTM supports different cultures and languages (see 5.10), then the Audit Trail notifications also shall be localized.

It is up to Frame Application to request an additional comment from the user e.g. to document the reason of a performed action. The Frame Application may request this comment when an operation is started on a DTM, for example

- Upload/Download,
- DTM functions are started, or
- DTM WebUI is started.

The Frame Application shall not disturb the user interaction with the DTM by requesting a comment during execution of the DTM action. If a comment is needed the user should be asked for the comment before the DTM action is started or after the action is finished.

The DTM does not need to provide any Audit Trail Information for the changes in the list of changeable parameters. The Frame Application may provide the notifications for the modifications in the list of changeable parameters without invoking the DTM.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

5 Technical concepts

5.1 General

FDT Objects shall be build upon the .NET platform [7] and executed in a runtime which is included in a .NET implementation (see 5.4). This statement excludes DTM WebUIs and Frame WebUIs which are described in (see 5.12).

The services, specified in the IEC 62453-2 specification [4][5], are modeled as .NET interfaces passing .NET datatype arguments (see chapter 7). These interfaces and datatypes are used for FDT Object interaction and data exchange. In addition, .NET exception classes (see 5.8.9.4) are defined for returning error information in an interface method call.

The FDT .NET interfaces, argument datatypes, and exception classes are defined in a .NET standard library (FDT core library). Figure 33 shows the assemblies and their dependencies to specific annex libraries.

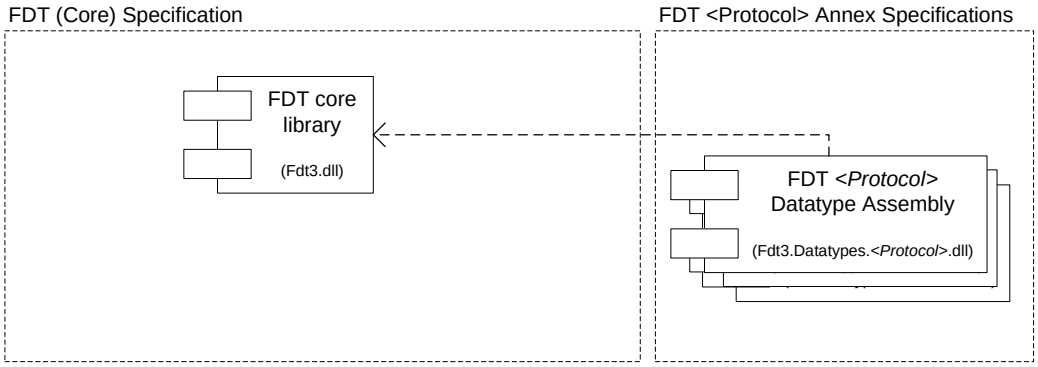


Figure 33 — FDT .NET Assemblies

The FDT core library assembly is provided together with this specification and shall be used for the development of Frame Applications and DTMs.

Some of the interface methods have to exchange protocol-specific information. These methods work with abstract base classes defined in the FDT Datatype assembly (e.g.: communication interface, see ICommunication interface definition in Annex B). Protocol-specific classes defining the protocol-specific data to be exchanged are derived from these base classes. These classes are defined in separated .NET standard libraries, which are provided together with corresponding FDT Protocol Annex specifications or by the DTM vendor in case of a vendor-specific protocol.

All FDT assemblies (FDT core assemblies and FDT protocol assemblies) are strong named (see 5.18.2) and additionally signed with an Authenticode key (see 5.18.3) owned by FDT Group, installed into the FDT-specific NuGet repository (see 9.3 and 9.4), and shared between the different FDT Objects.

The DTM Business Logic and Communication Channels shall be implemented by .NET classes. The DTM WebUIs are implemented as HTML pages and related components. The .NET classes are provided in .NET Standard libraries and the HTML pages are provided in WebUI packages (see Figure 34), which are installed and registered during the DTM installation process (see 9.5).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

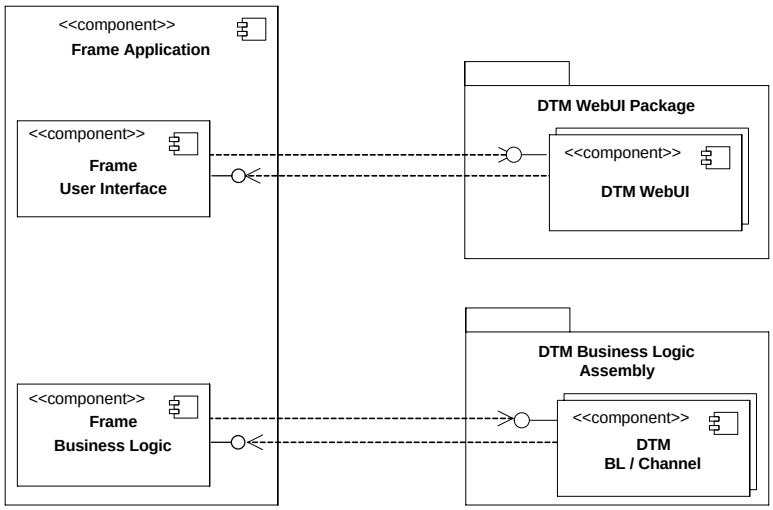


Figure 34 — FDT Object implementation

The DTM-specific assemblies shall be signed with a vendor-specific key.

The DTM Business Logic and Communication Channels shall be .NET classes implementing the interfaces defined in the FDT3 assembly (Fdt3.dll).

The DTM Business Logic shall be “creatable”:

- marked as public (non abstract)
- provide a public default constructor with no arguments

The implementation of the Frame Applications is not in scope of FDT. FDT only defines the interfaces which shall be provided to the DTM Business Logic and User Interface for callbacks.

The DTM WebUI are implemented as HTML pages with JavaScript. Accordingly, the data exchange between the DTM WebUIs and the Frame UI is done via JavaScript interfaces. The JavaScript interfaces are specified using TypeScript.

5.2 Support of HTML versions

This version of the FDT standard supports the HTML version as shown in Table 4.

Table 4 — Supported HTML versions

Supported HTML version
5

5.3 Support of JavaScript versions

This version of the FDT standard supports JavaScript conforming to the ECMAScript version as shown in Table 5.

Table 5 — Supported ECMAScript versions

Supported ECMAScript version
7

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

5.4 Support of .NET Common Language Runtime versions

5.4.1 General

DTMs are developed as .NET Standard Library objects. These objects may be used in .NET Framework CLRs and in .NET Core CLRs.

FDT Group defines the .NET Standard Library versions which shall be supported by FDT Software.

This version of the FDT standard supports the .NET Standard Library version as shown in Table 6.

Table 6 — Supported .NET Standard Library versions

Supported .NET Standard version
2.0

In future, FDT Group may define support for additional .NET Standard library versions.

Frame applications shall be based on a .NET implementation which supports the defined .NET Standard Library versions or higher. The documentation [8] includes a table which shows the minimum versions that support the defined .NET Standard library version for various CLI implementations (such as .NET Core or .NET Framework).

5.4.2 DTM rules

A DTM shall support at least one of the .NET Standard versions listed in Table 6. The supported version(s) shall be exposed in the DTM manifest (see 7.6.2). Support in this context means that the DTM vendor guarantees the correct function of the DTM (e.g. verified by tests).

Additionally a DTM shall provide the required minimum version of the .NET Standard in the package manifest (see 9.6.3).

DTMs shall be deployed as libraries. At runtime, they will depend on the .NET implementation provided by the Frame Application.

Communication DTMs may require an internal dependency to a platform-dependent component. An example of this is a native driver component for local connections such as USB ports. Such a native driver component may have to be provided specifically for each supported target platform, e.g. Windows 32 bit, Windows 64 bit or a specific Linux distribution. In this case, the DTM shall provide the different components in its deployment package. The Frame Application is then responsible to install the correct component or to deny the installation on unsupported platforms (see also 9.5). Device and Gateway DTMs shall not use such dependencies, they shall be built upon the .NET standard library only.

5.4.3 Frame Application rules

A Frame Application shall support all .NET Standard Library versions listed in Table 6. This also means that the Frame Application is responsible for providing the .NET implementation on the target. The deployment model such as Framework-dependent deployment or Self-contained deployment (see [9]) is not in the scope of the FDT specification.

The Frame Application shall check the .NET Library versions supported by the DTM before a DTM Business Logic is started. If the same .NET Library version is supported by the Frame Application, then the DTM Business Logic may be loaded and executed.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

A Frame Application may support multiple versions of the FDT specification. In this case it is responsible to start every DTM in a matching runtime. This may mean that the Frame Application needs to start DTMs in multiple processes, where each process provides the correct runtime for the started DTMs. However, the actual implementation of such a compatibility is product specific and is not in the scope of this specification. The general rules for backwards compatibility are provided in Annex D.

5.5 Support for 32-bit and 64-bit target platforms

Device and Gateway DTMs shall be compiled using the “any”- platform target. Therefore, the Frame Application can choose the platform and runtime (including the bitness) without any restrictions imposed by the DTMs.

Communication DTMs may have platform dependencies (see 5.4.2). This dependency is resolved by the Frame Application during the installation of the DTM (see 9.5).

5.6 Object activation and deactivation

5.6.1 General

A Frame Application needs to find and load the DTM-specific .NET assemblies dynamically into the memory and execute the contained DTM Business Logic classes by calling corresponding FDT interfaces. Finally, the created objects need to be destroyed and unloaded from the memory.

This chapter describes the means which shall be utilized for object activation and deactivation and the corresponding rules that shall be followed by Frame Applications and by DTMs.

5.6.2 Assembly loading and object creation

The DTM-specific .NET Standard assemblies are installed by the DTM installation process. The installation process also registers the DTM by installing “DTM manifest” file(s) in an FDT-defined directory (see 9.5). A manifest file contains the information where to find the .NET Standard assemblies and which DTM classes are contained (see 7.6.1 and 7.6.2).

The Frame Application shall use this information for loading and execution of the DTM classes.

Because of this behavior FDT defines the following rules:

1. Rules regarding assembly dependencies (see 5.6.3)
2. A standard loading method shall be used in the context of FDT (see 5.6.3.2). The use of other .NET assembly loading / object creation means is not allowed.
3. Rules regarding shared assemblies (see 5.6.4).
4. DTM assemblies shall be installed to a path which is browsable and readable.

The security aspects regarding loading and execution of assemblies are described in chapter 5.18.

The next steps after creation for the DTM Business Logic are described in 6.3.2.

5.6.3 Assembly dependencies

5.6.3.1 Introduction

DTM-specific .NET Standard assemblies may depend on other .NET Standard assemblies, for example on a device vendor-specific library or on a 3rd party library as outlined in Figure 35.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

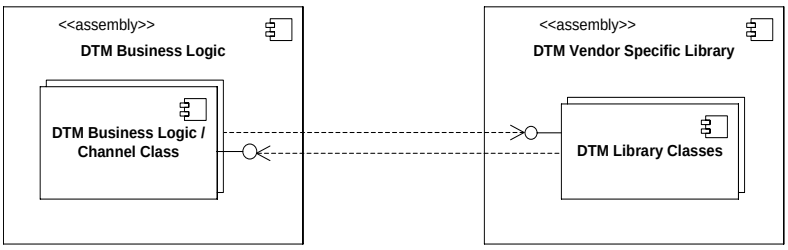


Figure 35 — Example: Assembly dependencies

These dependencies and the interaction between the classes / controls contained in the assemblies are DTM-specific, but the DTMs have to follow some rules in order to function correctly and to avoid problems in conjunction with other DTMs executed in a Frame Application.

5.6.3.2 Loading of dependent assemblies

The Frame Application loads the .NET Standard assemblies from the DTM installation path.

The DTM shall be deployed as a complete package, including all dependencies. The Frame Application is responsible to resolve the dependencies for each DTM. However, DTMs shall not assume that the Frame Application is able to resolve dependencies from external sources such as a NuGet repository.

5.6.4 Shared assemblies

5.6.4.1 General

Special attention is necessary for assemblies which are shared. Shared in this context means, that an assembly with the same identity is used by multiple DTMs . This applies to assemblies containing DTM BL as well as all other used assemblies.

Note: The identity of strong named assemblies consists of the assembly simple name, version, culture and public key token.

Note: The behavior described here applies to all shared assemblies independent of the location of the assembly.

If a shared assembly is used, then following rules apply:

1. Any incompatible change to the shared assembly shall lead to a new identity (e.g. different version number).
2. Shared assemblies shall not presume to be loaded from a specific installation path (e.g. rely that some files are stored in the same directory or in a sub-directory).
3. Static variables in shared assemblies are also shared if the assembly is loaded into the same ApplicationDomain. Thus static variables shall not have side effects in such scenarios. It's strongly recommended not to use static variables in a shared assembly.

If the rules above cannot be ensured by a DTM vendor, then the assembly shall not be used as a shared assembly. That means either the assembly gets a DTM-specific identity or it shall not be used at all.

5.6.4.2 Specification Assemblies

The FDT Specification Assemblies (Core interfaces and types, Protocol Annex types) are also shared assemblies (see 5.6.4.1). However, they are special as they provide the interfaces for the interaction between the Frame Application and a DTM as well as between two DTMs. Therefore, it is necessary that all FDT Components inside a process use the same version of the FDT Specification Assemblies at runtime.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

The FDT Group may provide compatible updates for the FDT Specification Assemblies over the time. In order to ensure that all FDT Components can work correctly, the following rules apply:

-

- The Frame Application as well as any DTM shall reference the FDT Specification Assemblies in their deployment packages. The FDT Specification Assemblies shall be provided as separate NuGet packages.
- The Frame Application is responsible to install the NuGet packages into a local NuGet repository
- The Frame Application is responsible to always load the newest installed version of a FDT Specification assembly, and thereby override the actual references of the DTMs.

These rules only apply to compatible updates of the FDT Specification Assemblies. If two interacting FDT Components are based on incompatible versions of the FDT Specification, the Frame Application needs to provide adapters in order to resolve this situation.

5.6.5 Object deactivation and unloading

5.6.5.1 Introduction

Destroying of DTM Business Logic and unloading of corresponding .NET assemblies have to be considered separately.

5.6.5.2 Destroying of objects

Steps to destroy an object depend on the object type. The procedure for the DTM Business Logic is defined in 6.3.2.

The DTM shall free all allocated resources during the ‘releasing’ state.

5.6.5.3 Unloading of assemblies

A .NET assembly which is loaded into a process is never unloaded, until the process is destroyed. Frame Applications may load DTMs into separate processes in order to remove unused Assemblies from the memory.

The DTM assemblies shall be developed with this .NET behavior in mind. To reduce the memory consumption it’s recommended

- to minimize the use of static variable, because these increase the memory consumption of the assembly.
- to move DTM functionality which is not always (or rarely) needed to separate assemblies. These assemblies are loaded only if corresponding code is executed.

5.7 Datatypes

5.7.1 General

.NET data classes (datatypes) are used for the data exchange between the different FDT objects. Instances of these classes are passed as arguments in the FDT interface methods, properties, and events.

The datatypes are defined in the .NET assembly Fdt3.dll, which is distributed together with this specification document. This assembly shall be used for the development of FDT Objects.

The datatypes are designed as so called “Data Contract” classes. These are classes using the attributes defined in the .NET System.Runtime.Serialization namespace. The actual data is provided by properties with corresponding [DataMember] attribute set as shown in Figure 36.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
using System.Runtime.Serialization;
using Fdt;

/// <summary>
/// Description of SomeDatatype
/// </summary>
[DataContract]
public class SomeDatatype : FdtDatatype<SomeDatatype>
{
    /// <summary>
    /// Description of data provided by the property
    /// </summary>
    [DataMember(IsRequired = true)]
    public string DataProperty1 { get; set; }

    /// <summary>
    /// [Optional] Description of data provided by the property
    /// </summary>
    [DataMember(IsRequired = false)]
    public FdtList<SomeSubDatatype> DataProperty2 { get; set; }
}
```

Figure 36 — Example: Datatype definition

The attributes control the serialization / deserialization of the instances (see 5.7.2) and also defines which properties are mandatory and optional (see 5.7.3).

All data classes are directly or indirectly derived from the base class FdtDatatype (see 7.1), which provides methods to verify (see 5.7.4) or clone instances (see 5.7.5).

5.7.2 Serialization / deserialization

The data classes support serialization / deserialization of data in different formats over the DataContractSerializer class provided in the .NET System.Runtime.Serialization namespace (e.g. binary format and XML) [11]. This may for example be used by the Frame Application to transport the data classes in WCF interfaces (Windows Communication Foundation) or for remote interaction in a network, but such use cases are out of scope of this specification.

5.7.3 Optional elements

Properties with [DataMember(IsRequired = true)] attribute assigned are mandatory (see DataProperty1 in Figure 36). That means they shall not be set to null (or string.Empty) if the instance is passed over an FDT interface.

Properties with [DataMember(IsRequired = false)] attribute assigned are optional (see DataProperty2 in Figure 36). That means they may be set to null if the instance is passed over an FDT interface.

For better distinction optional properties are marked with an “[Optional]...” comment. Additionally, all data classes provide a constructor for mandatory properties, which may be used to initialize a class instance with required data. The use of this constructor is optional. The mandatory properties can also be set later, but before the instance is passed over an FDT interface.

5.7.4 Verify

All data classes provide a Verify() method, which checks the rules defined for the data. Dependent on the class this may only be the basic mandatory / optional rules or additional rules defined in the data class description.

The FDT object receiving data from another object may use this method to check whether data is valid. However, the use of this method is optional. The receiving object may use other means to handle invalid data (e.g. check used properties whether they are null manually) or provide a

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

specific mode which allows to switch verification on/off. This may be a good option to reach maximum performance during runtime, but to provide a fallback strategy for trouble-shooting.

5.7.5 Clone

All data classes provide a Clone() method, which creates a new object that is a deep-copy of the called instance. That means all objects are duplicated - the top-level objects in the properties provided by the data class itself, as well as all lower level objects in properties of the sub-classes.

The cloning of data class instances is mandatory if an FDT object class member variable is passed over an FDT interface as argument or return value. This rule applies to methods, events and properties of interfaces.

This is necessary because of two reasons:

1. The receiving object may change the property values of a received data instance. If only a reference to the data would be passed, changing a value would also affect internal data of the sending object.
2. The receiving object may keep a reference to the received data instance. Further changes to the original data instance after the call returned may lead to unexpected results and threading issues.

If references are passed (e.g. interface reference or AsyncResult objects), no cloning shall be used.

Figure 37 shows two examples where cloning is necessary.

```
public class MyDtm
{
    private SomeDatatype _someData = new SomeDatatype(/* init with data */);

    public SomeDatatype DoSomething1()
    {
        return _someData.Clone();
    }

    public void DoSomething2()
    {
        MyOtherObject anotherObj = new MyOtherObject();
        anotherObj.DoSomethingWithMyData(_someData.Clone());
    }
}
```

Figure 37 — Example: Data cloning

If data class instances are created each time a method is called and no internal instances are referenced, then passing of instance references is allowed as shown in Figure 38.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
public class MyDtm
{
    public SomeDatatype DoSomething1()
    {
        SomeDatatype someData = new SomeDatatype(/* init with data */);

        return someData;
    }

    public SomeDatatype DoSomething2()
    {
        SomeDatatype someData = new SomeDatatype(/* init with data */);

        AnotherObject anotherObj = new AnotherObject();
        anotherObj.DoSomethingWithMyData(someData);

        return someData;
    }
}
```

Figure 38 — Example: Methods without data cloning

5.7.6 Equals

The Equals() method compares the identity of objects, it cannot be used to compare the contents of different objects.

In order to compare the contents of objects, developers need to implement the comparison.

5.7.7 Lists

The generic class FdtList<> is used for lists of data class instances (see 7.1). Like FdtDatatype this class provides methods to verify or clone the content of the FdtList<> instances itself and all contained elements.

If an FdtList is passed over an FDT interface, then the instance shall never be empty. If the corresponding property is optional, then the property shall be set to null instead.

5.7.8 Nullable

Nullable represents an object whose underlying type is a value type to which also ‘null’ can be assigned. (like a reference type)

5.7.9 Enumeration

Enumeration is a distinct type consisting of a set of named constants.

5.7.10 Protocol-specific datatypes

5.7.10.1 General

Protocol-specific datatypes shall be defined in .NET Standard libraries which are provided either by FDT Group together with the corresponding FDT Protocol Annex specifications or by DTM vendors in case of vendor-specific protocols.

The protocol-specific assemblies shall contain datatypes derived from the corresponding base classes in the FDT Datatype assembly (see Figure 39).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

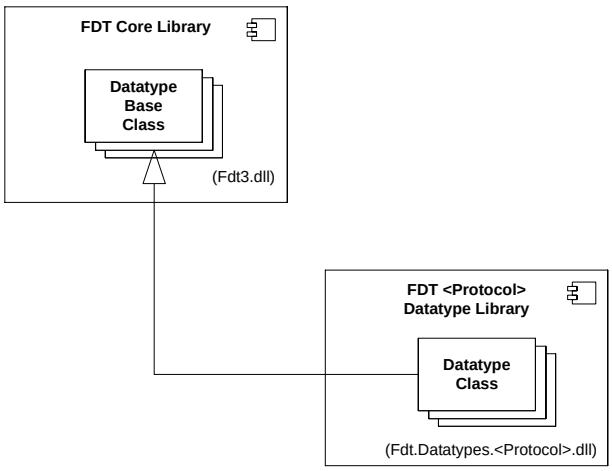


Figure 39 — Protocol-specific datatypes

Some of the FDT interface methods exchange protocol-specific information. These methods are defined with the protocol neutral base classes.

Protocol-specific assemblies shall support 32-bit platforms as well as 64-bit platforms. This means they shall be built using the “any”- platform target.

5.7.10.2 Interaction DTM - Frame Application

Typically, the DTMs create instances from the protocol-specific classes and pass them to the Frame Application over an FDT interface. The Frame Application then works with the properties and methods in the base classes. Thus, the Frame Application is able to handle any DTM independent of the protocol. Chapter 7.5 provides examples for using the DeviceIdentInfo classes with protocol neutral data and protocol-specific data.

5.7.10.3 Interaction DTM - DTM

If protocol-specific datatypes are used for a DTM to DTM interaction, then one DTM typically creates an instance of the protocol-specific classes and passes it over the corresponding FDT interface. The DTM which receives the data then casts the reference back from the base class to the protocol-specific datatype. Chapter 7.7 contains examples for using the protocol-specific Communication classes.

5.7.10.4 Installation and registration

The protocol-specific .NET assemblies shall be installed and registered by the DTMs using the protocol (see 9.4).

The protocol-specific .NET assemblies are installed in the FDT-specific NuGet repository. The DTM-specific assemblies can use static references in order to load the protocol assembly automatically together with itself (see 5.6).

In some cases the Frame Application also needs to load the protocol-specific assemblies and create instances from the contained classes, e.g. for deserialization of a protocol-specific datatype. In order to support such scenarios the protocol-specific assemblies shall be registered with a corresponding manifest file (see 9.4.3). The Frame Application can evaluate the provided information and then load corresponding assembly specifically (see Figure 40).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

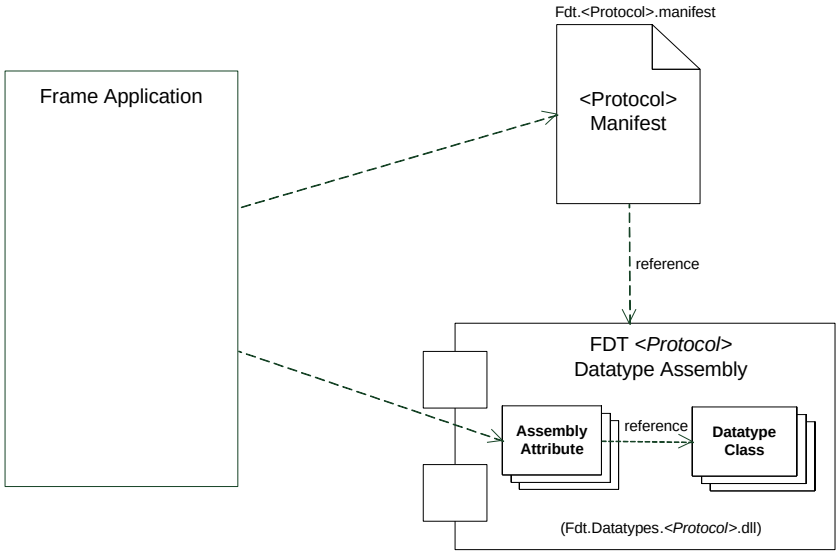


Figure 40 — Protocol manifest and type info attributes

In addition, the assembly shall expose type information as attributes assigned to the assembly itself. The Frame Application can use this information to create instances from the protocol-specific classes. The attribute classes are defined in the FDT Datatype assembly.

Following attributes (and corresponding datatypes) shall be supported by a protocol-specific assembly:

- ProtocolInfo attribute(see 7.3)
- DeviceIdentInfoType attribute(see 7.5)
- CommunicationType attribute (see 7.7)
- IOSignalInfoType attribute (see 7.11.1)
- DeviceAddressType (see 7.12)
- NetworkDataType attribute (see 7.13)

The example in Figure 41 shows the attributes assigned to the HART-specific datatype assembly (Fdt.Datatypes.Hart.dll).

```
[assembly: ProtocolInfoAttribute(ProtocolId = Hart.ProtocolId, ProtocolName =
Hart.ProtocolName)]

[assembly: DeviceIdentInfoType(
                      DeviceIdentInfoType = typeof(DeviceIdentInfo<HartDeviceIdentInfo>),
                      ProtocolDeviceIdentInfoType = typeof(HartDeviceIdentInfo),
                      DeviceScanInfoType = typeof (DeviceScanInfo<HartDeviceScanInfo>),
                      ProtocolDeviceScanInfoType = typeof(HartDeviceScanInfo))
]

[assembly: CommunicationType(AbortMessageType = typeof(HartAbortMessage),
                             ConnectRequestType = typeof(HartConnectRequest),
                             ConnectResponseType = typeof(HartConnectResponse),
                             DisconnectRequestType = typeof(HartDisconnectRequest),
                             DisconnectResponseType = typeof(HartDisconnectResponse),
                             SubscribeRequestType = typeof(HartSubscribeRequest),
                             SubscribeResponseType = typeof(HartSubscribeResponse),
                             UnsubscribeRequestType = typeof(HartUnsubscribeRequest),
                             UnsubscribeResponseType = typeof(HartUnsubscribeResponse))
]

[assembly: IOSignalInfoType(IOSignalInfoType = typeof(IOSignalInfo<HartIOSignalInfo>),
                            ProtocolIOSignalInfoType = typeof(HartIOSignalInfo))
]
```

Figure 41 — Example: Protocol assembly attributes

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Protocol-specific datatypes shall support the serialization/deserialization mechanisms as defined in section 5.7.2. The example in Figure 42 shows how the Frame Application can load a protocol-specific assembly and create an instance of a datatype class by using the DataContractSerializer.

```
public DeviceIdentInfo DeserializeDeviceIdentInfo(ProtocolManifest manifest, Stream stream)
{
    string path = Path.Combine(manifest.AssemblyInfo.Path, manifest.AssemblyInfo.Name);
    Assembly assembly = Assembly.LoadFile(path);

    Type attributeType = typeof(DeviceIdentInfoTypeAttribute);
    DeviceIdentInfoTypeAttribute deviceIdentAttrib =
        (DeviceIdentInfoTypeAttribute)assembly.GetCustomAttributes(attributeType, false)[0];

    DataContractSerializer serializer =
        new DataContractSerializer(deviceIdentAttrib.DeviceIdentInfoType);
    return  (DeviceIdentInfo)serializer.ReadObject(stream);
}
```

Figure 42 — Example: Handling of protocol-specific assemblies in Frame Application

5.7.11 Custom datatypes

The FDT datatypes are not intended for customization, because they are used in cooperation of software from different parties. That is why most FDT-datatypes are sealed (protected against changes/inheritance).

The only datatypes that can be extended are the base classes for protocol-specific datatypes and for UI-messaging datatypes. If extending such datatypes, following rules shall be applied:

- Use the [DataMember] attribute for all newly declared class members.
- All class members shall have a type serializable with DataContractSerializer.
- Custom data types shall be serializable with the DataContractSerializer.

Protocol-specific datatypes (as described in 5.7.10) also shall be sealed.

In order to support serialization (see 5.7.2), the custom datatypes shall expose information about the known types.

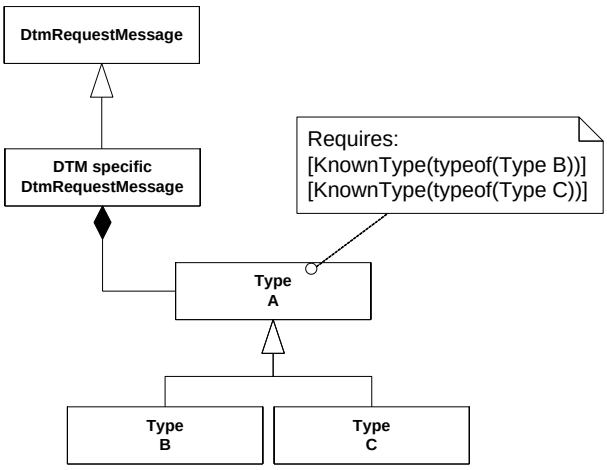


Figure 43 — Example: Handling of KnownType for custom data types

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

If there are derived custom datatypes (as shown in the example in Figure 43), then the KnownTypeAttribute shall be used to declare all derived datatypes.

5.8 General object interaction

5.8.1 General

All FDT Objects interact with each other exclusively via the interfaces defined by this specification. These interfaces are defined according to the services specified in IEC 62453-2 specification [5].

The interfaces define properties and methods of the server object as well as events, that may be received by the client object. In order for a client object to receive those events, the client object has to register delegates for these events. If not explicitly defined otherwise for an interface it is optional for the client object to register for the events of an interface.

5.8.2 Decoupling of FDT Objects

FDT3 decouples the FDT Objects from each other. The Frame Application is the one and only component that directly interacts with the DTM Business Logic, User Interfaces and Communication Channels via the FDT3 interfaces corresponding to the services defined for the objects in the IEC 62453-2 specification [5].

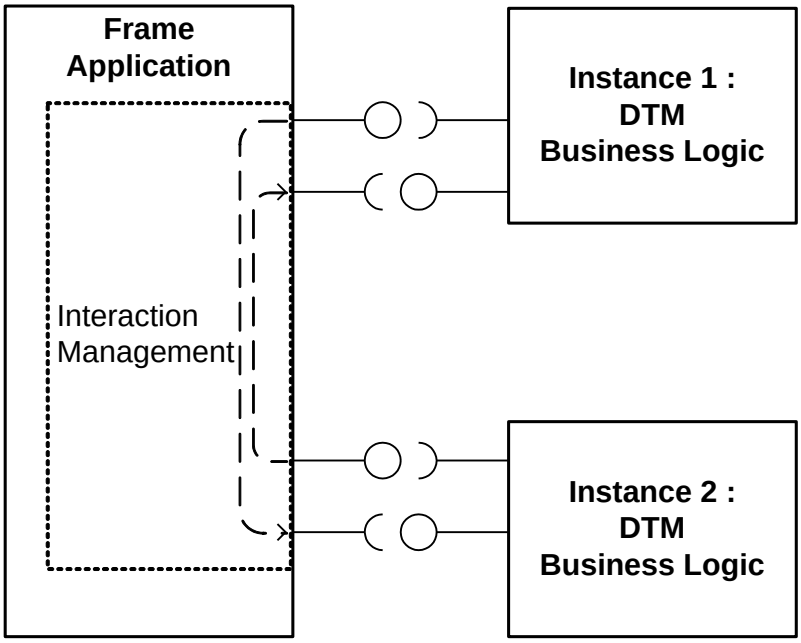


Figure 44 — Decoupled FDT Objects in FDT3

All component interactions are passed through the Frame Application or proxy components (see Figure 44). The Frame Application shall not change interactions or inject interaction requests.

This addresses the following objectives:

1. Interoperability
   The decoupling of the FDT objects by the Frame Application shall improve interoperability.
2. Tracing
   The Frame Application is able to observe the complete interaction between FDT objects. Thus it can implement a system wide tracing which is useful for diagnosis and trouble-shooting.
3. Testing
   Each component shall also be testable in a component test environment. The test support

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

can be achieved by tracing, replaying recorded sequences or by error injection. In the testing use case the Frame Application is allowed to change interactions.

4. Threading / Synchronization
   The Frame Application is responsible for the assignment of FDT Objects to processes. An FDT Object shall not expect to be executed in the same thread, process or host like other related FDT Objects. The Frame Application can enforce rules in regard to method calls, whereas the rules may differ between the different FDT Objects (see 5.9).
5. Remoting
   The Frame Application can pass the messages to a different process or a remote computer (see 11.3)

An example for components, which are used for decoupling of FDT Objects, are proxy objects (e.g. channel proxy) that are used to interact with the respective FDT Object.

Note: The Frame Application part handling the interaction between the different FDT objects – called Interaction Management in Figure 44 – could be separated from the actual Frame Application implementation. It could be realized as shared component, which is then used by different vendors. This would reduce the implementation effort for the different Frame Application vendors and increase the interoperability with DTM.

5.8.3 Parameter interchange with .NET datatypes

The arguments of interface methods are defined as .NET datatypes. The definition of these .NET datatypes includes:

- Type definition (e.g. definition as .NET class/structure)
- Definition of standard methods for
  - Serialization to/from XML
  - Serialization to/from binary stream
  - Verification

The XML format and the format for the binary stream are well defined formats, specified in FDT3.

In order to ensure interoperability for FDT components, the .NET interfaces and datatypes specified by FDT3 are implemented in a primary assembly, which are provided by FDT Group. It is mandatory for all FDT components to use this primary assembly.

5.8.4 Interaction patterns

In FDT3 following interaction patterns are used:

- Properties
- Synchronous methods
- Asynchronous methods
- Events

These patterns and their usage is explained in the following sections.

5.8.5 Properties

Properties are used for simple get or set operations on simple data objects that are performed synchronously. Other interfaces of an FDT object are also provided by properties.

5.8.6 Synchronous methods

Synchronous methods are used for simple operations that can be performed synchronously within the calling thread. The called object shall not block the calling thread, e.g. by waiting on asynchronous operations to finish or waiting on events.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Examples for synchronous methods are:

- Information Requests (e.g. IDtmInformation.GetDeviceIdentInfo())
- Simple state machine operations (e.g. IDtm3.Init(), IDtm3.EnableCommunication2())
- Frame Application calls that do not require nested calls (e.g. ITopology.GetParentNodes())

5.8.7 Asynchronous methods

5.8.7.1 Introduction

Asynchronous methods are typically used to perform operations that may take a relatively long time to complete, such as I/O or database operations, communication requests. Such an asynchronous operation executes in a separate thread. When an application starts an asynchronous operation, the application can continue execution while the asynchronous operation is performed. Asynchronous methods are implemented using the IAsyncResult pattern.

5.8.7.2 IAsyncResult pattern

The IAsyncResult pattern as defined in [15] is used for asynchronous calls to services.

Using this pattern an asynchronous operation is implemented as a set of methods:

- The Begin*OperationName*() method starts the asynchronous operation OperationName. The BeginOperationName method shall return control to the calling thread immediately. If the BeginOperationName method throws exceptions, the exceptions are thrown before the asynchronous operation is started and the *OperationName*Completed() callback method is not invoked.
- The End*OperationName*() method ends the asynchronous operation OperationName and retrieves the results of the operation. If the operation has not completed when EndOperationName is called, End*OperationName* blocks until the operation is finished. Exceptions which occurred during the asynchronous operation are thrown from the EndOperationName method.
- The Callback delegate *OperationName*Completed() (implemented by the client) is provided only for a specific service call that triggers one specific event type that can be received.

For further information on how to implement the IAsyncResult pattern see F.1.

One of the advantages of the IAsyncResult-Pattern is, that the client may choose to use the service in a blocking or in a non-blocking way.

If a service is used in a blocking way, the client calls the Begin() method and immediately the End() method (see Figure 45). The calling thread of the client will be blocked, until the service execution is finished.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

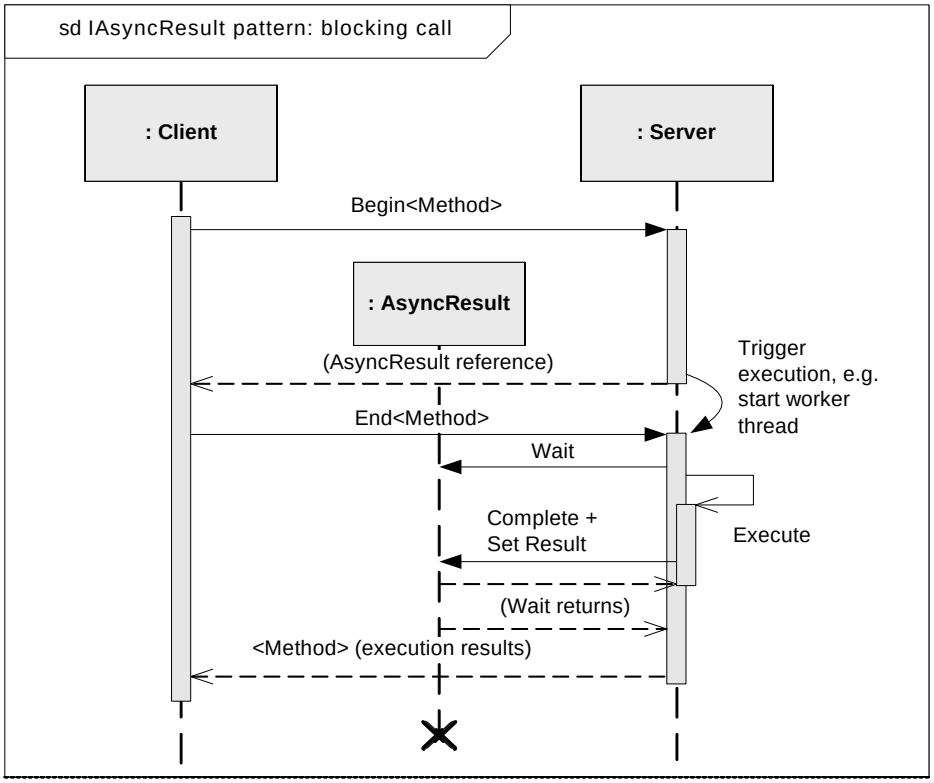


Figure 45 — IAsyncResult pattern: blocking call

Figure 46 show an example how blocking use of asynchronous operation may be implemented.

```
void SyncUpload(IDtm3 dtm, ICommunicationChannelProxy channelProxy)
{
    // go online and stay connected (synchronous)
    dtm.EnableCommunication2(channelProxy, ConnectMode.StayConnected,
                                            DeviceTypeCheckMode.PerformCheck);

    // perform upload from device (synchronous)
    try
    {
        IOnlineOperation onlineOperations = dtm.OnlineOperations;
        IAsyncResult result =
            onlineOperations.BeginReadDataFromDevice(null, null, null);
        onlineOperations.EndReadDataFromDevice(result);

        MessageBox.Show("Upload finished");
    }
    catch (Exception e)
    {
        MessageBox.Show("Upload failed! " + e.Message);
    }

    // go offline (synchronous)
    IAsyncResult offline_result = dtm.BeginStopCommunication(null, null, null);
    dtm.EndStopCommunication(offline_result);
    dtm.DisableCommunication();
}
```

Figure 46 — Example: Blocking use of asynchronous interface

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

In order to simplify the presentation of interactions based on the IAsyncResult pattern, a simplified presentation for blocking call is used throughout the document. Figure 47 shows the simplified depiction of IAsyncResult pattern with blocking call:

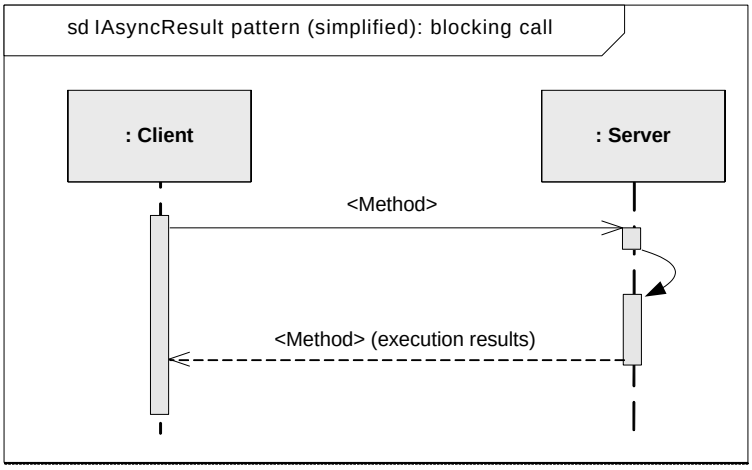


Figure 47 — IAsyncResult pattern (simplified): blocking call

Rule: If the client follows the pattern for blocking execution, it shall provide no callback.

If a service is used in a non-blocking way, the client calls the BeginOperationName() method and provides a callback delegate for OperationNameCompleted() (see Figure 48). The EndOperationName() method is called as part of handling the OperationNameCompleted() callback.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

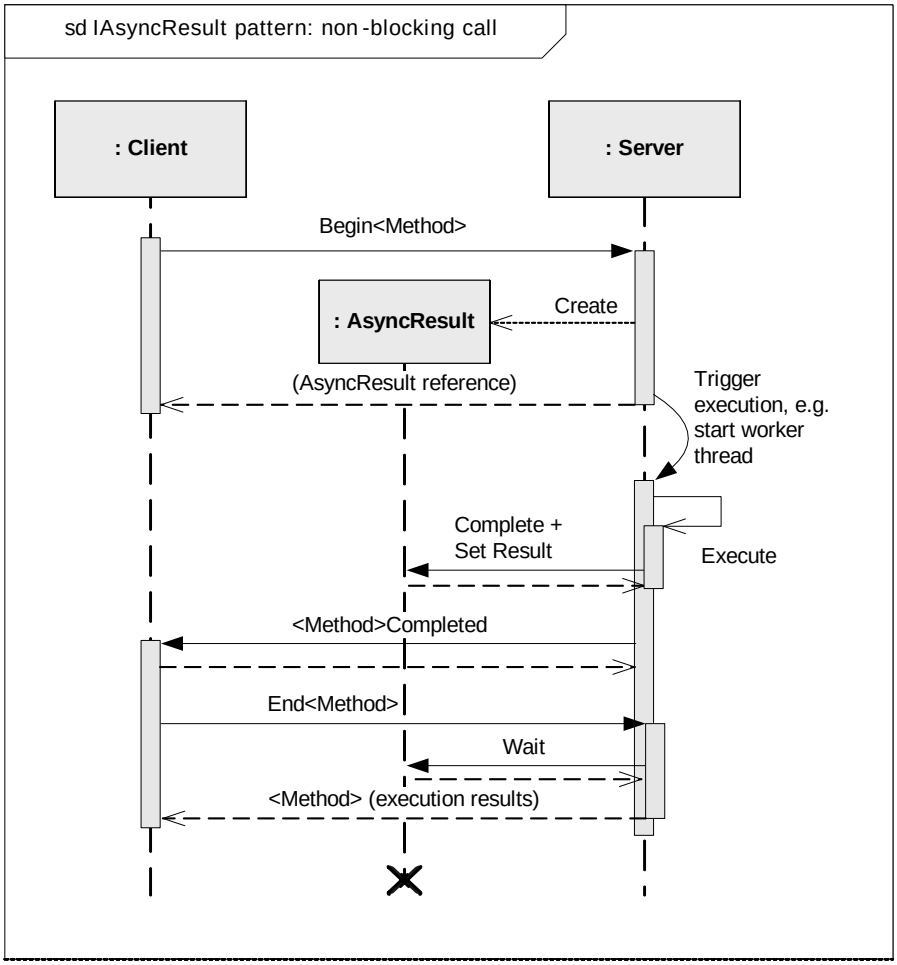


Figure 48 — IAsyncResult pattern: non-blocking call

Rule: If the callbacks are provided, the client shall follow the pattern for non-blocking execution.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
void AsyncUpload(IDtm3 dtm, ICommunicationChannelProxy channelProxy)
{
    // go online and connect only if necessary
    dtm.EnableCommunication2(channelProxy, ConnectMode.OnDemand,
                                            DeviceTypeCheckMode.PerformCheck);

    IOnlineOperation onlineOperations = dtm.OnlineOperations;
    IAsyncResult result = onlineOperations.BeginReadDataFromDevice(UploadProgress,
                                                                   UploadComplete, dtm);
}

void UploadProgress(ProgressInfo progressInfo)
{
    UpdateProgressBar(progressInfo.PercentComplete, progressInfo.Message);
}

void UploadComplete(IAsyncResult result)
{
    IOnlineOperation onlineParameter = result.AsyncState as IOnlineOperation;

    try
    {
        onlineParameter.EndReadDataFromDevice(result);
        SignalUploadFinishedToUI();
    }
    catch (Exception e)
    {
        SignalUploadErrorToUI();
        throw;
    }

    // go offline (not waiting for results..)
    _stop_result = dtm.BeginStopCommunication(StopCommunicationProgress,
                                              StopCommunicationComplete, dtm);
}
```

Figure 49 — Example: Non-blocking use of asynchronous interface

In the above example (Figure 49) the UploadProgress() delegate is decoupled in order to avoid blocking of the server.

In order to simplify the presentation of interactions based on the IAsyncResult pattern, a simplified presentation for non-blocking call is used throughout the document. Figure 50 shows the simplified depiction of IAsyncResult pattern with non-blocking call:

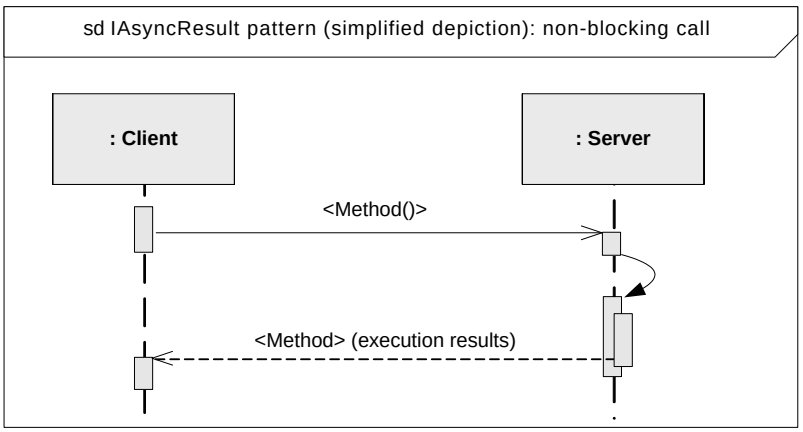


Figure 50 — IAsyncResult pattern (simplified depiction): non-blocking call

NOTE: Throughout the document the simplified depiction of IAsyncResult pattern is used to show how methods are using the IAsyncResult pattern. The patterns for blocking and non-blocking calls can be used equivalently. The use of one of the call pattern in a workflow does not prohibit the use of the other call pattern if not stated explicitly otherwise.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

5.8.7.3 Extended IAsyncResult pattern (Progress pattern)

In addition to the IAsyncResult pattern, the extended IAsyncResult pattern provides the possibility to cancel an asynchronous operation and to receive progress notifications and intermediate results during the processing of the operation. This pattern is used for operations that may have long execution times.

For each operation a set of methods is provided:

- The Begin*OperationName*() method starts the operation.
- The End*OperationName*() method retrieves result of the operation. If the operation is not finished, the method blocks until the operation is finished. If an error occurred during execution of the operation, this method will throw an exception with the error information.
- The Cancel*OperationName*() method stops the operation. If the operation was cancelled, then the End*OperationName*() method shall always throw the FdtOperationCancelledException.
- The Callback delegates (implemented by the client) are provided only for a specific operation. Possible delegates are: *OperationName*Progress(), *OperationName*Completed().

Figure 51 shows how the method Cancel*OperationName*() may be used.

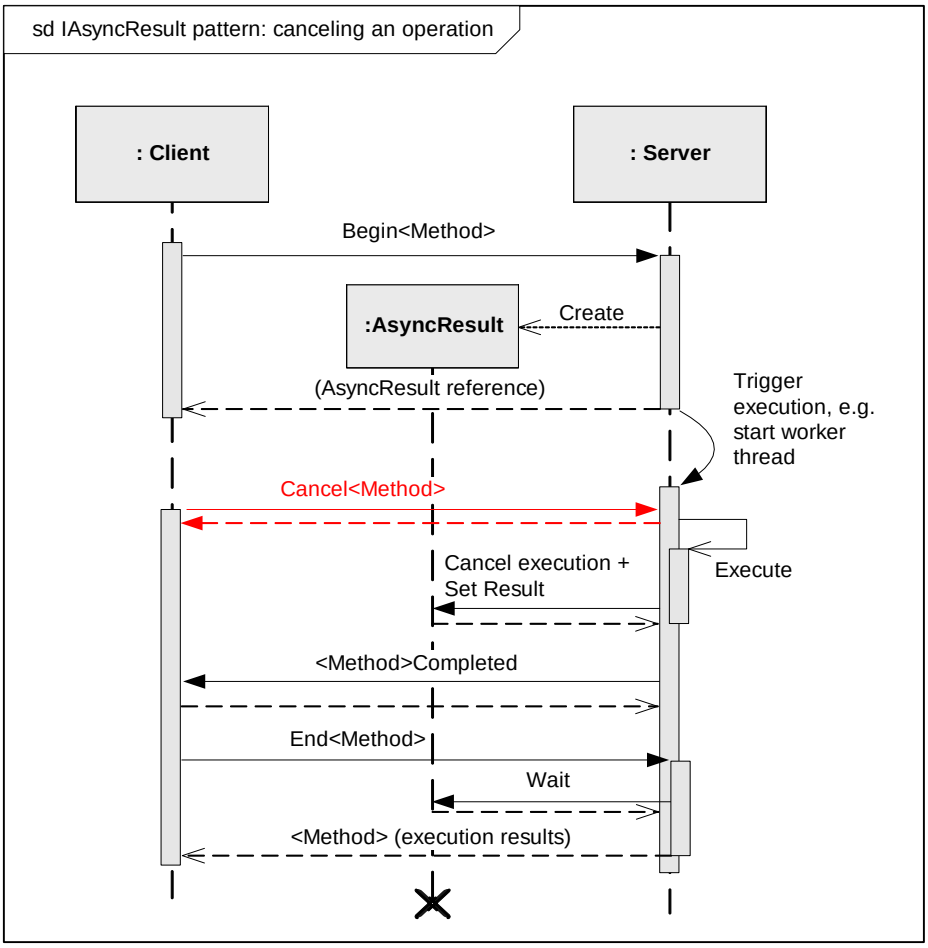


Figure 51 — IAsyncResult pattern: canceling an operation

If Cancel() cannot be executed, it may result in an FdtCancelFailedException. It may also occur that the operation finished at the same time as Cancel() was called. This may lead to the caller receiving a positive result.

The Completed() callback shall not be called within a call to Cancel() (avoid call-stacks).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

If Cancel() is called for an asynchronous operation, the End<Method> may throw a corresponding exception. See the documentation for each asynchronous operation.

After a call to Cancel() succeeded it may occur in an exceptional case, that the operation finishes successfully. Therefore the caller shall be prepared to receive a positive result instead of the corresponding exception.

Figure 52 shows how the progress callback may be used.

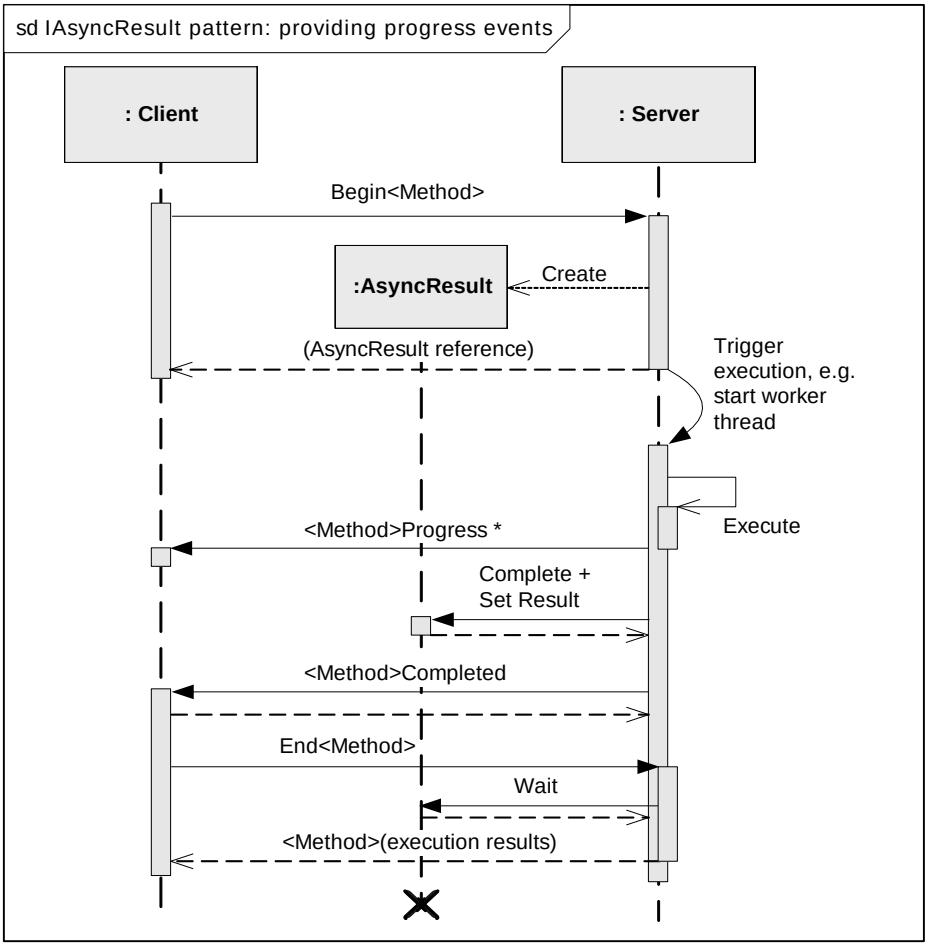


Figure 52 — IAsyncResult pattern: providing progress events

The Progress() delegate not only allows to pass progress information, but also can be used to transport partial results of the service execution. The transport of partial results is designed for each service specifically. Even if the progress delegate is used to transport partial results, the End*OperationName*() method will provide the complete result of the operation.

Rule: If the client follows the pattern for blocking execution, it shall provide no callback. For non-blocking execution both callbacks shall be provided.

If the callbacks are provided, the client shall follow the pattern for non-blocking execution.

5.8.8 Events pattern

An event is a message sent by an object to signal the occurrence of a condition.

This specification uses the Events pattern which is based on delegates (see [17]).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Clients provide delegates for receiving events (without trigger). A client registers with a server for receiving a specific event. Multiple clients may register with one server for receiving the same event.

One advantage of the events pattern is that events are defined in the same interface like the methods that may trigger those events. This allows to define the events methods in the same context in which they are used.

5.8.9 Exception handling

5.8.9.1 General

Exceptions are the primary means of reporting errors (refer to [12], Chapter 7]). They are used for both hard errors (e.g. passing of invalid arguments) as well as logical errors (e.g. connection aborted).

An exception provides two pieces of information:

- the exception message, explaining to the developer what went wrong (and how to fix it). Exception messages should be human readable text in English (not just an error number) that describes what went wrong
- the exception type that is used by exception handlers to decide what programmatic action to take

Note: In general error codes are not used as they can always be replaced by corresponding exception messages and exception types. However, there is one exception from the rule: communication errors. Communication errors that occur during communication requests with a device are reported within the communication response. However, if the server fails to perform the transaction itself, this will throw an appropriate exception.

The FDT specification defines the exceptions which shall be thrown if specific error situations occur when calling an FDT interface method or accessing a property. This shall be considered as part of the contract between the client and server of an interface.

5.8.9.2 Throwing exceptions

Exceptions shall be thrown in cases of execution failures. An execution failure occurs whenever an interface method or property cannot do what it was designed for. For example, if the <ReadDataFromDevice()> method cannot retrieve data from the device, it is considered an execution failure and an exception shall be thrown. Exceptions are the primary means of reporting errors. Error codes shall not be used.

If an FDT method invokes other FDT methods it shall handle all FDT exceptions that are defined for these methods. When the FDT method fails because of an FDT exception from an invoked method, then the FDT method shall throw the most appropriate FDT exception defined for the FDT method and include any caught exception as an inner exception. One example for this is if setting IO signal information via the interface IProcessImage to the DTM fails because the dataset cannot be locked. In this case an FdtOperationFailedException shall be thrown by the IProcessImage:SetIOSignalInfo() method. This shall include the inner exception, e.g. FdtNoWriteAccessException.

DTM-specific exceptions shall also be included in FDT exceptions as inner exceptions when they occur within an FDT method.

Event-handlers are not allowed to throw exceptions. If an event-handler calls other methods that may throw exceptions, the implementation of the event-handler shall catch those exceptions in order to protect the event-source from those exceptions.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

5.8.9.3 Handling exceptions

If an exception is handled, a rich and meaningful message should be provided to the end user. The message should explain the cause of the problem and describe what could be done to avoid the problem.

NOTE: Since the exception message is targeted to developers, the user message should be based on the exception type and the context of the caught exception.

NOTE: If an exception is just caught in order to re-throw the exception, no user message should be provided. The goal here is to avoid multiple user messages for a problem that occurred.

5.8.9.4 FDT exception types

For each FDT method a set of exceptions is defined that may occur on invocation of the method. All FDT exceptions are derived from the serializable class FdtException that is derived from System.Exception. Exceptions shall be serializable in order to work correctly across application domain and process boundaries.

Following is the list of FDT Exceptions:

FdtInvalidStateException

This exception shall be thrown when a property cannot be set or when a method cannot be executed, because the FDT object is not in an appropriate state (e.g. IDtm3.EnableCommunication2() is called in DTM state 'initialized'). Each implementation of an FDT interface member shall check whether the called object is in an appropriate state to execute the requested operation. If this is not the case, FdtInvalidStateException shall be thrown. For asynchronous operations this exception shall be thrown in the EndOperationName method.

Example: IDtm3.EnableCommunication2() is called in state "initialized"

FdtOperationFailedException

This exception shall be thrown when an operation cannot be performed or completed successfully. For all asynchronous operations this exception may be thrown by the EndOperationName method. If more specific exceptions are available, always the most specific exception shall be used.

Note: FdtOperationFailedException should not occur under normal operating conditions.

Example: IDtm3.LoadData() is called with a valid dataset but still fails. In this case an FdtOperationFailedException is thrown. However, if IDtm3.LoadData() is called with an invalid dataset, FdtInvalidDatasetException is thrown.

FdtOperationCancelledException

This exception shall be thrown if an asynchronous operation has been cancelled by CancelOperationName and the EndOperationName method is called. This happens under normal operating conditions. The client shall handle this exception and abort its own operation.

FdtCancelFailedException

This exception shall be thrown when a CancelOperationName method fails, e.g. because the operation has been finished already or cannot be cancelled for other reasons. This may happen under normal conditions because of the asynchronous execution of the operation. The client shall handle this exception and finish the calling operation. If the user has triggered the cancel operation, the user should be informed that the operation could not be cancelled.

FdtConfigurationErrorException

This exception shall be thrown when an operation cannot be performed because of a wrong configuration.

Example: A DTM performs a connect request. The Parent DTM cannot perform the request as the communication driver is not properly configured.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

FdtCommunicationErrorException

This exception shall be thrown when a communication error occurs. Communication errors that occur within a communication request are reported with the communication response.

Example: The Device DTM tries to establish a connection by calling ICommunication:BeginConnect() on the provided Communication Channel proxy. The Communication Channel (or the device) is not able to establish the connection with the device because of a communication error and throws FdtCommunicationErrorException in method ICommunication:EndConnect().

Note: Protocol-specific communication error exceptions are not defined. However, subclasses may be defined by FDT Protocol Annex documents if required.

FdtConnectRefusedException

This exception shall be thrown when an online operation cannot be performed because the connect request has been refused.

Example: IDtm3.EnableCommunication2() is called on a Device DTM. The Device DTM tries to establish a connection by calling ICommunication.BeginConnect() on its Communication Channel proxy. The Communication Channel (or the device) refuses the connect and throws FdtConnectRefusedException in method ICommunication.EndConnect().

FdtConnectionAbortedException

This exception shall be thrown when an online operation cannot be performed because the connection has been aborted.

Example: The connection is aborted by the Communication Channel during a download to the device. IOnlineOperation.EndWriteDataToDevice() throws an FdtConnectionAbortedException.

FdtDeviceTypeNotSupportedException

This exception shall be thrown when an online operation cannot be performed because the type of the connected device is not supported by the DTM.

Example: A download operation is started with IOnlineOperation.BeginWriteDataToDevice(). The DTM is in state notConnected and connects to the device. It checks the device type and detects an unsupported device type. The operation is aborted and IOnlineOperation.EndWriteDataToDevice() throws an FdtDeviceTypeNotSupportedException.

FdtInvalidUserPermissionsException

This exception shall be thrown when an operation cannot be performed because the operation is not allowed with the current user permissions.

Example: A function is started with ICommandFunction:BeginExecute(). The user is logged in as Observer and has no access rights to perform this function. The DTM aborts the operation. ICommandFunction:EndExecute() throws an FdtInvalidUserPermissionsException.

FdtInvalidValueException

This exception shall be thrown when an invalid value was given as an argument in the request.

Example: A client tries to write a value via IInstanceData/IDeviceData that is out of the valid range.

FdtInvalidTypeIdException

This exception shall be thrown when an invalid type id was given as an argument in the request.

Example: IDtm3.InitData() is called with a TypeId that is not supported by the DTM and throws an FdtInvalidTypeIdException.

FdtInvalidDataObjectException

This exception shall be thrown when an invalid data object was given as an argument in the request.

Example: An invalid DtmSystemTag is given in ITopology:BeginGetDtm().

FdtInvalidReferenceException

This exception shall be thrown when an invalid reference to another object was given as an argument in the request. FdtInvalidReferenceException should not occur under normal operating conditions.

Example: An invalid IAsyncResult object is given in an EndOperationName or in a CancelOperationName method

FdtInvalidCommunicationChannelException

This exception shall be thrown when an invalid Communication Channel is set.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Example: The argument parentCommunicationChannel is set to a channel that is not supported (e.g. protocol is not supported).

FdtInvalidDatasetException

This exception shall be thrown when an invalid dataset was given as an argument in the request. FdtInvalidDatasetException should not occur under normal operating conditions.

Example: IDtm3.LoadData() is called with a dataset that is not supported by the DTM type.

FdtNoReadAccessException

This exception shall be thrown when a read operation cannot be performed because the data object is not readable.

Example: A client tries to read a data object that is classified as write only. EndRead() throws an FdtNoReadAccessException.

FdtNoWriteAccessException

This exception shall be thrown when a write operation cannot be performed because the data object is not writable.

Note: Before writing any data, the DTM shall initiate a transaction on the dataset with IDataset:StartTransaction() if this fails, the operation shall be aborted and an FdtLockDatasetException shall be thrown in this case.

Example: A client tries to write a data object that is classified as read only. EndWrite() throws an FdtNoWriteAccessException.

FdtLockDatasetException

This exception shall be thrown when the dataset cannot be locked in order to perform transactions on the dataset or device.

Example: IOnlineOperation.BeginReadDataFromDevice() is called. The dataset cannot be locked as it is currently locked by another instance. IOnlineOperation.EndReadDataFromDevice() throws an FdtLockDatasetException.

FdtCommitTransactionFailedException

This exception shall be thrown when a commit transaction fails. This may happen e.g. when the database is located on a remote computer and the network connection is disrupted.

Note: FdtCommitTransactionFailedException should not occur under normal operating conditions.

FdtCannotCloseUiException

This exception shall be thrown if a user interface cannot be closed. The user interface may have changed data items that have not been committed yet or some active actions with the device that need to be finished.

In this case, the Frame Application shall inform the user that he needs to finish active actions with this user interface before it can be closed.

5.8.9.5 Standard Exception Types

In general FDT exceptions shall be used where applicable (please refer to the corresponding FDT interface definitions). Following .NET standard exception types should be used in situations where no FDT exceptions are applicable.

InvalidOperationException

InvalidOperationException shall be thrown if the object is in an inappropriate state. If the object is a defined FDT object use FdtInvalidStateException.

ArgumentException, ArgumentNullException, ArgumentOutOfRangeException

ArgumentException or one of its subtypes shall be thrown if bad arguments are passed to an interface member. The most derived exception type should be used where applicable. The ParamName property represents the name of the parameter that caused the exception to be thrown. Note that the property can be set by one of the constructor overloads. Use “value” for the implicit value parameter of property setters.

5.8.9.6 Other standard exceptions

Following exceptions shall not be thrown by FDT objects. Argument checking shall be performed to avoid throwing these exceptions.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

- NullReferenceException,
- IndexOutOfRangeException,
- AccessViolationException

Following exceptions shall not be thrown explicitly by FDT objects:

- StackOverflowException,
- OutOfMemoryException,
- InteropException,
- ComException,
- SEHException,
- ExecutionEngineException

5.9 Threading

5.9.1 Introduction

5.9.1.1 General

Multi-threading as supported by .NET runtime solves several problems with regard to throughput and responsiveness, but in doing so it introduces new problems such as races and deadlocks. In order to avoid such problems in FDT3 some rules are defined that shall be applied by all FDT components.

NOTE: The threading terms (e.g. apartment model) as used within the context of COM do not apply in .NET.

Additional information about Multi-threading and concurrency can be found in [25].

5.9.1.2 Races

A race is a failure which occurs because of improper synchronization between threads. Depending on which of two or more threads reaches a particular block of code first the result of a program (or a particular piece of code) cannot be predicted. When different threads access common memory concurrently, the computing result may be correct or not.

There are four conditions required for a race to be possible.

1. There are memory locations that are accessible from more than one thread. Typically, locations are global/static variables or are heap memory reachable from global/static variables.
2. There is a property (invariant) associated with these shared memory locations that is needed for the program to function correctly. Typically, the property needs to hold true before an update occurs for the update to be correct.
3. The property does not hold during some part of the actual update.
4. Another thread accesses the memory when the invariant is broken, thereby causing incorrect behavior.

5.9.1.3 Locks

The most common way of preventing races is to use locks to prevent other threads from accessing shared memory associated with an invariant while it is broken. This removes the fourth condition mentioned above, thus making a race impossible.

The most common kind of lock is called a monitor (sometimes the same basic functionality is named a critical section, a mutex, or a binary semaphore). A monitor provides Enter and Exit

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

methods, and once a thread calls Enter, all attempts by other threads to call Enter will cause the other threads to block (wait) until a call to Exit is made. The thread that called Enter is the owner of the lock, and it is considered a programming error if Exit is called by a thread that is not the owner of the lock. Locks provide a mechanism for ensuring that only one thread can execute a particular region of code at any given time.

5.9.1.4 Deadlocks

A deadlock is a situation wherein two or more concurrent operations are each waiting for the other to finish, and thus neither can make any further progress.

There are four conditions required for a deadlock to be possible:

1. Mutual exclusion. Only a limited number of threads may utilize a resource concurrently.
2. Hold and wait. A thread holding a resource may request access to other resources and wait until it gets them.
3. No preemption. Resources are released only voluntarily by the thread holding the resource.
4. Circular wait. There is a set of {T1, …, TN} threads, where T1 is waiting for a resource held by T2, T2 is waiting for a resource held by T3, and so forth, up through TN waiting for a resource held by T1.

Since multiple threads can access an FDT object concurrently, it is necessary to synchronize the access to internal data objects or to user interface objects by mutual exclusion (see condition #1). Condition #2 is hard to avoid since multiple resources are often required to perform an operation. Resources that are locked cannot be pre-empted from the current owner, so condition #3 cannot be avoided as well. The most common and actionable condition in FDT is condition #4, circular waits.

5.9.2 Threading rules

5.9.2.1 Implementation rules

Following rules shall be applied in order to allow multithreading and using locks to avoid races:

1. Software shall be prepared to receive calls in any thread. Each FDT object shall be able to receive calls in any thread. Operations that need to be performed in a dedicated thread (e.g. user interface thread) shall be synchronized to this thread internally.
2. Software shall protect internal data against parallel access. Make static data and instance data thread safe. Ensure that all thread-shared, read-write data is protected by locks.
3. Each lock shall be assigned to a specific region of memory (not a region of code!). This assignment shall be well documented in the developer documentation.
4. Each lock shall provide mutual exclusion for the region of memory that it is assigned to. No writes to that memory can occur without entering the same lock. Data structure invariants have to hold any time the lock protecting the data structure is not held.
5. If two data structures are related, locks for both structures shall be entered before using that relationship.

Implementers should also consider following recommendations:

- Memory regions (e.g. data structures) that are protected by locks should not overlap. Consider that mutual exclusion may not be guaranteed if you have overlapping regions with different locks. If it would always be required to enter two or more locks, a single lock would be more appropriate to protect the overlapping regions.
- Use as few locks as possible. The complexity grows quickly with the number of locks in the system, so it is best to have few locks that protect large regions of memory and only split them when lock contention is shown to be a bottleneck on performance. Generally, the finer the granularity of the locks, the more of them that can be held at once - and the longer they are held, the higher the risk of deadlock.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

- Check whether read locks are required. Entering locks is not only required when writing to memory but also when reading from shared memory. In general, when code needs a program invariant, all locks associated with any memory involved with the invariant shall be entered.

5.9.2.2 Avoiding deadlocks

For the reasons explained in 5.9.1.4, following rules are defined to avoid deadlocks (list is continued from previous section):

6. FDT objects are not allowed to call any FDT interface method or wait on incoming FDT calls, callbacks or events while holding any locks. This avoids a circular wait across multiple FDT objects.

Note: Consider to implement a state machine when you need to call other FDT objects in order to perform a complex operation. Parallel requests may be refused or queued until the running operation is finished.

Exceptions: It is allowed to call ITrace methods and asynchronous BeginXXX() methods within locked code areas. The interaction management will ensure that these calls are decoupled and processed in a safe way.

7. Each FDT object shall avoid that a circular wait can happen within a single FDT object. Multiple threads accessing the object at the same time may perform different tasks and require multiple resources that need to be locked. Avoiding the circular wait condition is usually done by acquiring locks always in a specific order or by lock leveling. This is a strategy where each lock is assigned a level. A thread can only acquire locks with the same or lower level that it already holds.

5.9.2.3 FDT Object interaction rules

All FDT objects shall apply following rules (list is continued from previous section):

8. Do not call FDT interfaces in the user interface thread.
   The user interface thread of a process shall be dedicated to receive user inputs and perform drawing tasks only. FDT objects shall not use the user interface thread to call FDT interface methods, perform callbacks or events.
9. Do not block the user interface thread.
   The user interface shall always stay responsive. The user interface thread is shared between the different FDT (user interface) objects for user input and drawing operations. If one object blocks this thread in order to perform some processing, this would affect the responsiveness of other objects.
10. Do not block a BeginMethodName method call.
    A BeginMethodName method shall only start an asynchronous operation. Therefore it shall not block the caller.
11. Do not block a synchronous method call.
    A thread calling a FDT synchronous method shall not be blocked. It is not allowed to call any EndMethodName within a synchronous method or to wait on events, because this will block the calling thread.
12. Process events and callbacks asynchronously.
    No FDT operations shall be performed within an event handler or callback method. A work item shall be created that is processed asynchronously. The calling thread shall not be blocked.

5.10 Localization support

5.10.1 General

There are two main processes for developing software that supports different languages and cultures.

- Globalization is the first process to design software that is capable of running with different cultures and languages. This process is realized by separating the executable code that is culture or language independent from those parts that are culture or language dependent. Language-dependent parts for example are such as user interfaces, calendars, numbers,

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

several string manipulation and comparison algorithms. The .NET Standard Library as technological basis for FDT3 supports this process through a number of classes that are packaged under the namespace System.Globalization.

- Localization is the second process to customize the software to a specific culture and language. This is primarily achieved by translating the user interface. This results in a primary assembly that contains only culture-neutral and language-neutral executable code and resources. Each additional culture, region or language is provided in a separate satellite assembly.

The .NET Standard provides infrastructure to access the hierarchically organized resources. First the framework classes attempt to access the resources that belong to the specified region or country. If this access fails, the framework classes attempt to access the resources that belong to the specified language. If this access also fails, the framework classes attempt to access culture-neutral or language-neutral resources.

It is recommended to utilize the infrastructure provided by the .NET implementations. There are several translation tools on the market and the translation agencies know how to deal with the XML based as well as the binary resource files.

For DTM WebUIs, based on HTML5 the language support is provided by HTML-specific means (see 9.5.5).

5.10.2 Access to localized resources and culture-dependent functions

.NET provides two different ways to access the localized resources and culture-dependent functions.

The preferred way is to use the automatic culture handling. Each thread within an application provides information about the currently used culture and language setting. It can be retrieved by reading the properties CurrentCulture and CurrentUICulture from class System.Threading.Thread. All user interface related functions rely on the property CurrentUICulture whereas other culture-dependent functions use the property CurrentCulture. The value of these two properties are initialized to the default values defined in the system settings “Control Panel - Regional Settings”. The values of the properties can be overridden by writing the property values.

The second way to access localized resources and culture-dependent functions is needed only in some rare cases. All culture-dependent functions provide an overloaded variant where the culture can be specified explicitly.

5.10.3 Handling of cultures

As described above,.NET handles the culture settings for each thread separately and derives the start value from the system settings. The Frame Application is responsible to synchronize the culture settings for all threads with outgoing function calls.

There are two properties defining the used language: Thread.CurrentUICulture and Thread.CurrentCulture.

The Frame Application indicates the currently used language by setting the property Thread.CurrentUICulture before the DTM is started. During initialization the DTM Business Logic shall read this property in order to display the DTM WebUI or any other output that is provided to the user (e.g. labels and descriptors). If the DTM implementation relies on the resource management classes that are contained in the .NET class library, no additional implementation will be necessary. Otherwise, the language-dependent resources need to be handled explicitly. Switching the UI culture by the Frame Application shall not trigger any notifications from the DTM (e.g. DataInfoChanged).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

If a DTM uses additional threads with outgoing function calls or if it raises events from additional threads, it is responsible to synchronize the language settings of newly created threads with the settings from the original thread.

The property Thread.CurrentCulture shall not be changed by any FDT component in order to reflect the culture settings of the operating system.

The described mechanism results in the following behavior: Texts, pictures and similar user interface elements will be displayed according to the language that was selected by the user at the Frame Application (in Thread.CurrentUICulture). Whereas input direction, sort orders, comparison, number formatting are determined by the culture settings of the operating system (in Thread.CurrentCulture).

5.10.4 Switching the User Interface language

A Frame Application may provide a mechanism that allows switching the language of the user interface. It is specific to the Frame Application, whether switching is realized during runtime or whether it needs the restart of the Frame Application. The Frame Application shall not switch the user interface language as long as any DTM or DTM component is instantiated. The Frame Application may need to shutdown and restart the DTM objects in order to switch the language. The DTMs will start with the new language setting.

At the Frame Application the user might select a language that is not available for certain DTMs. The DTM Business Logic and DTM WebUI shall operate as expected independent of the selected language. If a DTM does not support the selected language, it shall switch to a commonly used fallback language, which is English.

A DTM WebUI may provide a menu, where the user can override the language setting for this user interface. The DTM Business Logic and DTM WebUI shall not change the properties Thread.CurrentUICulture and Thread.CurrentCulture, because this would influence the behavior of other FDT components that share this thread. The language remains active until the language setting is changed again or until the user interface is closed.

In distributed systems, the Frame Application shall ensure that the DTM Business Logic and the DTM WebUI (displayed inside a Frame User Interface) are using the same language. This ensures that any text (or other language-specific information) which is generated inside the DTM BL but displayed on the DTM WebUI will be displayed in the correct language.

5.11 DTM User Interface implementation

5.11.1 General

A DTM Business Logic is operated by DTM User Interface. FDT3 supports following DTM User Interface type:

- DTM WebUI can be embedded into Frame Application WebUI, which can be executed in a web browser. The DTM WebUI shall be based on HTML5 and JavaScript.

5.11.2 Private dialogs

DTM WebUI are not allowed to show private dialogs (i.e. windows or dialogs outside of the browser). If a DTM WebUI needs to support additional user interaction, it shall show these user interfaces on top of the DTM WebUI (e.g. as modal box or overlay).

These UI elements

- shall stay within the display area of the DTM WebUI,

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

- shall not overlap with the Frame Application WebUI,
- shall not block access to the Frame Application WebUI and
- shall always be displayed on top of the DTM WebUI.

A DTM Business Logic is not allowed to open private dialogs or user interfaces. A DTM Business Logic shall always use the Frame Application user interface services (see IFrame.FrameUi property). This rule is necessary since operations on the DTM Business Logic may be executed unattended (e.g. batch processing). This means, a DTM should be able to execute even if IFrame.FrameUI is not available.

Note: For instance the use of the JavaScript method “alert()” is not allowed.

5.11.3 Modal DTM WebUI

Modal DTM WebUI are implemented by DTMs as WebUIs, which are opened with a specific request (BeginOpenDtmUiModal()) from the DTM BL to the Frame Application. The availability and the implementation of the request depends on the Frame Application. For instance, if the Frame Application is providing a WebUI, then modal DTM WebUIs will not be supported, because the interface IFrameUI will not be available.

Possible implementations (by the Frame Application) are:

- Frame Application modal (blocking the whole FA)
- DTM instance modal (blocking only one device)

The decision on how to implement this method is specific for a Frame Application. It could be even configurable by the user.

Note 1 An example for the need to implement Frame Application modal user interfaces, is, when a SAFETY related dialog needs to be confirmed until a hazardous operation is started (e.g. starting a heavy drive).

Note 2 A DTM may always implement DTM instance modal UI elements as described in 5.11.2.

5.12 DTM User Interface hosting

5.12.1 General

The Frame Application can dynamically load the DTM User Interface.

The sequence diagrams in 8.5 describe these operations in more detail.

FDT supports DTM user interfaces as platform independent DTM Web User Interfaces (DTM WebUIs). DTM WebUIs can be hosted by Web Applications or can be embedded into Windows Applications.

5.12.2 Hosting DTM WebUI

5.12.2.1 General

The DTM WebUI is hosted in the Frame User Interface (conventional user interface or Frame Application WebUI). The Frame Application provides a WebData Connector to the DTM WebUI. The DTM WebUI uses the WebData Connector in order to exchange information with the DTM Business Logic, i.e. the WebData Connector is executed within the DTM WebUI.

The WebData Connector is provided by the Frame Application in a JavaScript file and implements a specified interface (see Figure 68). Since the WebData Connector is provided at runtime by the Frame Application, it is not part of the DTM WebUI container file itself. The implementation of the WebData Connector is Frame Application specific and depends on the Frame Application’s architecture.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

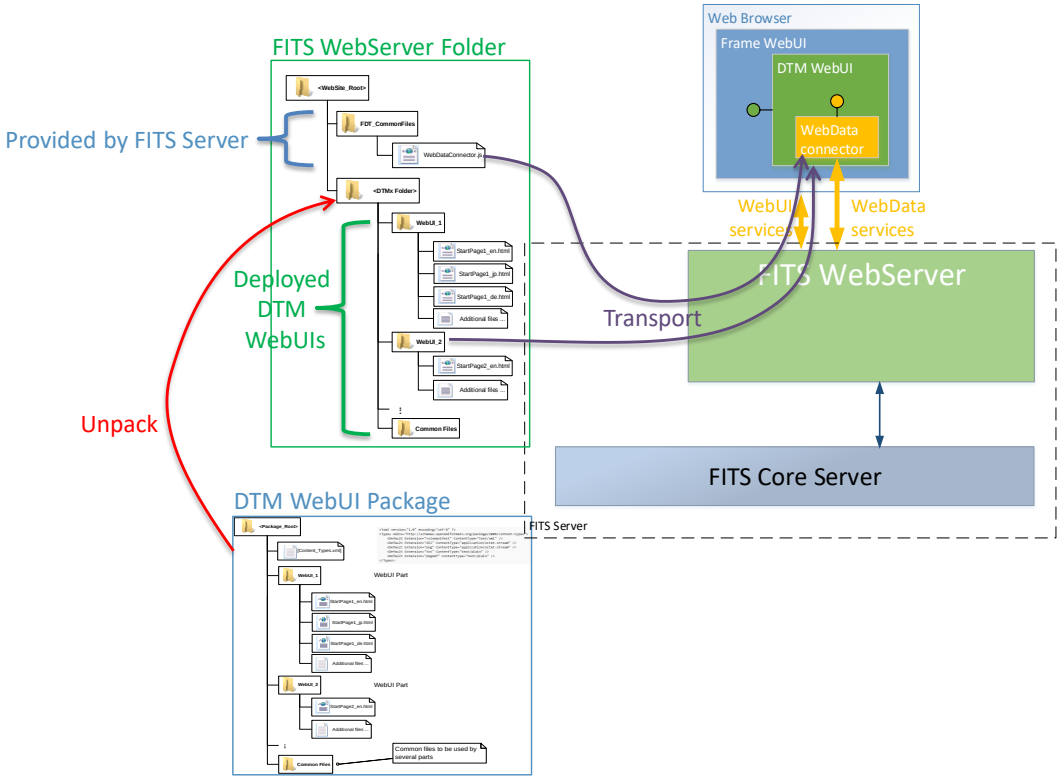


Figure 53 — General concept for hosting DTM WebUI

DTM WebUIs are provided within the DTM package files in container files according to the Open Packaging Conventions (see 9.5.5).

Depending on the architecture of the Frame Application, DTM Web User Interfaces can be embedded into a Web Application (provided via WebUI Services) or they can be embedded into a Windows Desktop Application (local hosting).

When a Frame Application User Interface hosts a DTM WebUI, the respective HTML page (with its components) is loaded into the FA user interface. Then the DTM WebUI is initialized and fitted into the layout of the FA user interface. This means that the DTM WebUI as a whole can be loaded and is able to function without access to the Internet.

5.12.2.2 Frame Application WebUI

If the Frame Application hosts the DTM Web User Interfaces in a Frame Application WebUI, it shall implement the following general sequence:

- Extract the container file into a folder that is published via the WebUI services.
- Make sure that the DTM WebUI can reference an implementation of the WebData Connector.
- Navigate to the DTM WebUI
- Invoke the initialization routine which is implemented in the DTM WebUI

The Frame Application shall consider that the DTM WebUI may use third party libraries such as Angular. In order to avoid versioning conflicts between the DTM WebUI and the Frame Application or other DTM WebUIs, it shall isolate the DTM WebUI in an appropriate way. This means that the Frame Application shall host the DTM WebUI in an HTML <iframe>.

Figure 54 shows code a snippet which illustrate how a Frame WebUI loads a DTM WebUI.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
// declare initialization message
const postMessage = {
    'websocketUrl': this.websocketUrl + "/" + this.sessionId + "/" +
    this.currentProjectId + "/" + this.currentDtmSystemTag,
    'dtmSystemTag': this.currentDtmSystemTag,
    'functionId': this.functionId,
    'invokeId': this.invokeId,
    'showIdentificationArea': this.showIdentificationArea,
};

// create an iframe element
const dtmFrame = document.createElement('iframe') as HTMLIFrameElement;
dtmFrame.id = 'iframe_' + this.invokeId;
dtmFrame.name = 'iframe_' + this.invokeId;
dtmFrame.frameBorder = '0';
dtmFrame.width = '100%';
dtmFrame.height = '400px';

// navigate to the DTM Web UI url
dtmFrame.src = this.uiUrl;

// post a message to the DTM Ui which will trigger the initialization
const uiUrlDomain = this.uiUrlHost;
dtmFrame.onload = function () {
    dtmFrame.contentWindow.postMessage(postMessage, uiUrlDomain);
};

// add the content to the visual tree
element.appendChild(dtmFrame);
tabContents.appendChild(element);
```

Figure 54 — Example : Integration of a DTM WebUI into a Frame WebUI

Since the DTM WebUI is hosted in an <iframe>, the initAsync() method shall be invoked via an event-listener which is part of the WebDataConnector (provided by the Frame Application).

If the Frame Application hosts the DTM Web User Interface in a Windows Desktop Application, it shall implement the following general sequence:

- Extract the container file into a temporary folder
- Create a web browser control and embed it into the Frame Application's user interface
- Make sure that the DTM WebUI can reference an implementation of the WebData Connector.
- Load the DTM Web User Interface into the browser control
- Invoke the initialization routine implemented by the DTM WebUI

See Figure 55 for an example.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
/// <summary>
/// Loads the DTM WebUI into the browser control.
/// </summary>
/// <param name="url">
/// The local url of the extracted DTM WebUI package ("file://...")
/// </param>
public void LoadPage(string url)
{
    // Browser is a web browser control, in this example
    // of the type CefSharp.Wpf.ChromiumWebBrowser.
    Browser.FrameLoadEnd += InitUiFunction;
    Browser.Address = url;
}

/// <summary>
/// Calls the initialization routine of the DTM WebUI after the page has been loaded.
/// </summary>
/// <param name="sender"></param>
/// <param name="args"></param>
private void InitUiFunction(object sender, FrameLoadEndEventArgs args)
{
    Browser.FrameLoadEnd -= InitUiFunction;
    var frame = args.Frame; // frame of the web browser control

    string systemTag = _dtmSystemTag.ToString("D");
    string invokeId = _invokeId.ToString("D");
    string functionId = _functionId.ToString();

    string script = $"Init(
                        'local', '{systemTag}',
                        '{functionId}', null,  '{invokeId}', true);";

    frame.ExecuteJavaScriptAsync(script);
}
```

Figure 55 — Example : Hosting a DTM WebUI in a WPF Frame Application

5.12.2.3 Integration of WebData Connector

The DTM WebUI is using a JavaScript API provided by the Frame Application for communicating to the DTM BL. This JavaScript API is defined in E.2.

Figure 56 shows how the JavaScript provided by the Frame Application is integrated into the DTM WebUI and how it can be used by the DTM WebUI in order to send information to the DTM BL. The example shows that a DTM-specific Message object is used, which provides the conversion to JSON and from JSON. The requirements for such conversion are defined in 7.15.

The JavaScript file for the WebData Connector shall be provided with the absolute path:

`/FDT_CommonFiles/WebDataConnector.js`

The path for the WebData Connector file shall be provided by the Frame Application and shall be used by the DTMs with the correct spelling and letter case as defined above.

NOTE The use of correct letter case is important for deployment to case-sensitive platforms (e.g. Linux).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>…</title>
    <link rel="stylesheet" href="…" type="text/css" />

    <!-- integration of the WebData Connector -->
    <script src="/FDT_CommonFiles/WebDataConnector.js"></script>
    <script>
        <!-- example for using the WebData Connector -->
        function onInputEventHandler() {
            if (event != null) {
                if (event.target) {
                    <!-- prepare the message data -->
                    var msgData;
                    msgData = new Message("…",
                                          event.srcElement.textContent);

                    webData.SendMessagesAsync(msgData.toJSON(),
                                              requestId.toString(), sendCallback)

                    …
                }
            }
    </script>
</head>
```

Figure 56 — Example : Integrating the WebData connector in a DTM WebUI

5.13 Static Function implementation

Static Functions are implemented as .NET functions. A DTM provides information about the available Static Functions (see StaticFunctionInfo in Annex B) with the method IStaticFunctionInformation.GetStaticFunctions. The information provided by the DTM describes the supported use case and the arguments of the function.

The Static Function may use communication data provided by the Frame Application to communicate with the device. The Static Function shall establish a new connection to the device for each execution, i.e. connection to the device cannot be shared.

The Frame Application shall take care that there is no parallel access to the device from DTM and from StaticFunction Provider.

For the execution of the static function the Frame Application has to choose exactly one protocol, because it has to provide the corresponding bus address and network data information to the StaticFunction Provider.

The DTM provides:

- the list of static functions, where each Static Function provides a list of supported protocols in its StaticFunctionInfo, and
- a list of active protocols, assigned by the Parent DTM during the creation of the network topology (see 8.8.2).

The intersection of the two lists might contain multiple protocols. The Frame Application is responsible for selecting the correct protocol.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

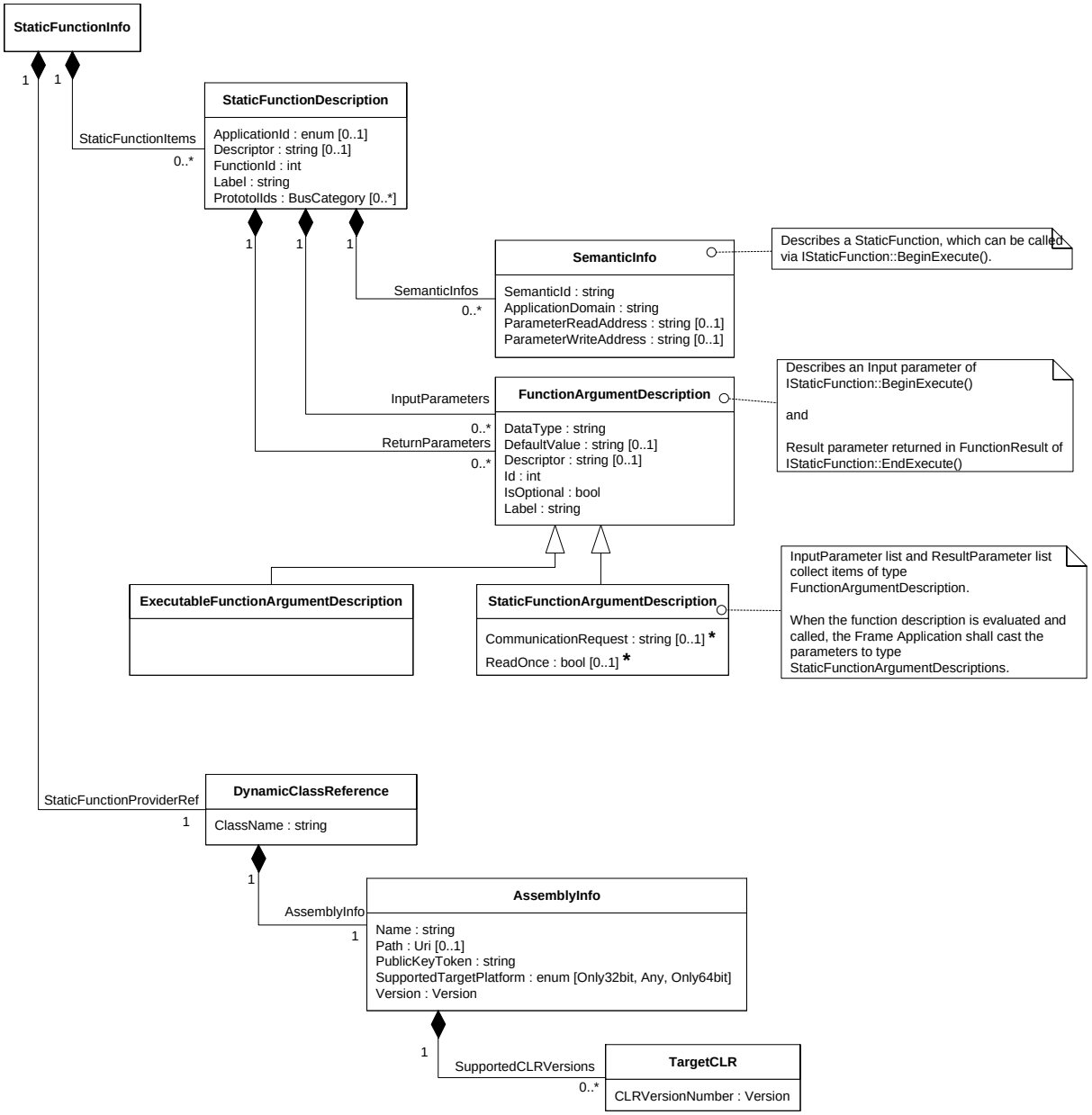


Figure 57 — Relation of StaticFunctionDescription to Static Function

Figure 57 shows the relation between description of a Static Function and the actual function that may be invoked. The StaticFunctionDescription describes a StaticFunction together with its input arguments and its result arguments. For each argument of the StaticFunction there is a corresponding description.

The input and result arguments of a Static Function may be of any datatype. The values of the arguments are provided as string. The string contains the serialized value of the datatype.

The StaticFunctionInfo provides a reference to the class that can be used in order to execute the StaticFunction (StaticFunctionProviderRef). Even though there is only one class reference for all StaticFunctions, an implementation could use this class as a dispatcher to individual classes that provide the actual implementation of the individual StaticFunctions.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

5.14 Persistence

5.14.1 Overview

In the call to InitData() or LoadData() the DTM receives a reference to the IDataset interface provided by the Frame Application.

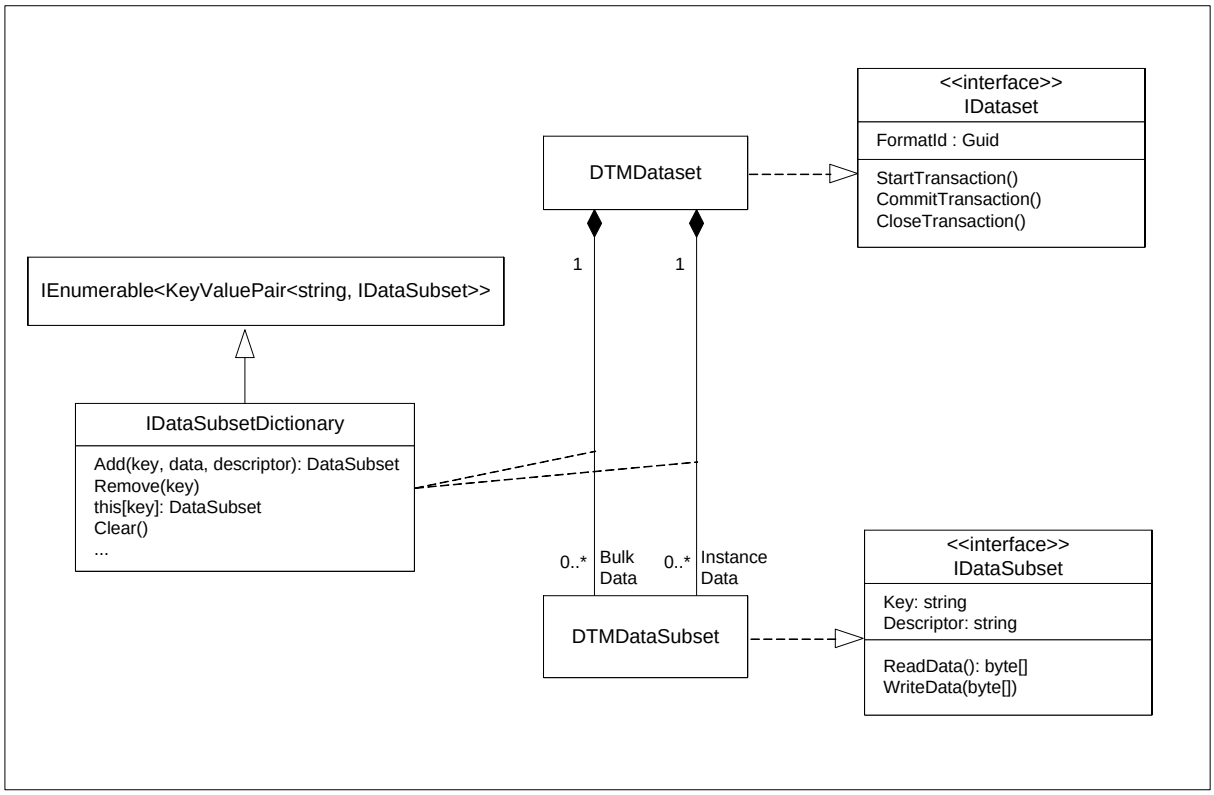


Figure 58 — DTMDataset structure

The DTMDataset contains two DTMDataSubset dictionaries for the actual persistence of data (see Figure 58): InstanceData dictionary and BulkData dictionary. Each dictionary contains DTMDataSubsets. The DTMDataSubsets shall be used for grouping of persistence data. The number and content of the DTMDataSubsets is DTM specific. In order to improve the system performance the DTM shall group data which need to be loaded and stored together in one DTMDataSubset. Furthermore, a DTM shall avoid unnecessary loading of data whenever possible, especially when starting the DTM Business Logic.

The InstanceData dictionary shall be used for data which is directly related to the represented device instance, for example the device parameters, network information, etc. The DTM has to guarantee that it is able to represent the device by loading this data. Following DTMDataSubsets should be considered:

- Basic data which is needed during the complete lifetime of a DTM instance(e.g. represented device type information, device tag and address and other identity information).
- Device parameter information that is needed if corresponding DTM WebUI is opened (e.g. a page in a dialog) or if the Frame Application requests data (e.g. DeviceDataInfo (see 4.4.3))
- IO signal information which is needed if Frame Application requests ProcessDataInfo (see 4.4.4)

The BulkData dictionary shall be used for further device instance-specific data, for example for bulky trend or historical data, which is only needed in special scenarios. The Frame Application may implement a special storage mechanism for bulk data, which might be optimized for

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

handling of big amount of data, but may be slower than the implementation for the instance data storage.

Beside these data separation and grouping rules the DTM shall also follow the rules defined for data searching (see 5.14.4) to support a maximum system performance.

5.14.2 Data format

The format of the data persisted in the DTMDataSubsets is DTM specific, but the DTM shall store information about the used format in the IDataset.FormatId property. The FormatId is a unique identifier created by the DTM vendor, it shall correspond to the used FormatId exposed in the IDtm3.ActiveType property.

The DTM shall use the FormatId information to decide how to load the data, e.g. to load data stored in a different format by an older DTM version.

A DTM may also expose further supported FormatIds in its TypeInfo. This information may be used by the Frame Application to migrate data stored by a different DTM (see 8.17).

5.14.3 Adding / reading / writing / deleting of data

By default the two DTMDataSubset dictionaries contained in the DTMDataset are empty. The DTM itself is responsible for adding the needed DTMDataSubsets to the dictionaries, for example at first start-up in the call to InitData() (see 8.2.1).

The DTMDataSubset dictionaries also provide methods to remove data from the persistence storage managed by the Frame Application. However, in case of deleting of the DTM instance the Frame Application itself is responsible to remove the data from the storage after the call to IDtm3.BeginRelease(deleteInstance=true) returned.

The IDataSubset interface provides methods to read and write binary data. The DTM itself is responsible to serialize / deserialize the data.

Figure 59 shows an example implementation on how a DTM can initialize a DTMDataSubset with binary data by using the BinaryFormatter class for serialization.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
public void InitData(Guid dtmDeviceTypeId, IDataset dataset)
{
    // initialize class members
    _dataset = dataset;
    _activeDtmDeviceType = _supportedTypes.Find((item) =>
        item.Id == dtmDeviceTypeId);
    _deviceAddress = new DeviceAddress<HartDeviceAddress>(1,
                        new HartDeviceAddress(0, "SHORTTAG", "Long Tag ....",
                            HartDeviceAddress.AddressingModeSelection.ShortAddress,
                            new HartLongAddress()));
    // start transaction (needed for adding of DataSubsets)
    _dataset.StartTransaction();

    // create binary formatter needed for serialization of data
    MemoryStream stream = new MemoryStream();
    BinaryFormatter binaryFormatter = new BinaryFormatter();

    // serialize ActiveType(Id) and DeviceAddress
    binaryFormatter.Serialize(stream, _activeDtmDeviceType.Id);
    binaryFormatter.Serialize(stream,
                              _deviceAddress.ProtocolSpecificDeviceAddress.ShortAddress);
    stream.Close();

    // create data subset for "basic" DTM data and initialize with default data
    _dataset.InstanceData.Add("basicData", stream.GetBuffer());

    // store used format and close transaction with auto commit = true
    _dataset.FormatId = _activeDtmDeviceType.DatasetFormats.Used.Id;
    _dataset.CloseTransaction(true);
}
```

Figure 59 — Example: Initialization of DTMDataSubset with DTM data

The DTM has to provide a unique key for the DTMDataSubset when adding it to the dictionary. The DTM can use the key to access DTMDataSubset, for example for reading and writing of the binary data.

Figure 60 shows an example on how a DTM can write binary data into a DTMDataSubset by using the BinaryFormatter class for serialization.

```
protected void SaveBasicData()
{
    // start transaction (needed for writing of DataSubset data)
    _dataset.StartTransaction();

    // create binary formatter needed for serialization of data
    MemoryStream stream = new MemoryStream();
    BinaryFormatter binaryFormatter = new BinaryFormatter();

    // serialize ActiveType(Id) and DeviceAddress
    binaryFormatter.Serialize(stream, _activeDtmDeviceType.Id);
    binaryFormatter.Serialize(stream,
                              deviceAddress.ProtocolSpecificDeviceAddress.ShortAddress);
    stream.Close();

    // create datasubset for "basic" DTM data and initalize with default data
    _dataset.InstanceData["basicData"].WriteData(stream.GetBuffer());

    // close transaction with auto commit = true
    _dataset.CloseTransaction(true);
}
```

Figure 60 — Example: Writing of DTM data in DTMDataSubset

Figure 61 shows an example on how a DTM can read data from a DTMDataSubset by using the BinaryFormatter class for deserialization.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
public void LoadData(IDataset dataset)
{
    _dataset = dataset;

    // read persisted "basic" data
    byte[] data = _dataset.InstanceData["basicData"].ReadData();

    // create binary formatter which is insensitive regarding assembly version
    // in which serialized classes have been defined
    MemoryStream stream = new MemoryStream(data);
    BinaryFormatter binaryFormatter = new BinaryFormatter();
    binaryFormatter.AssemblyFormat = FormatterAssemblyStyle.Simple;

    // deserialize ActiveType and DeviceAddress data
    Guid dtmDeviceTypeId = (Guid)binaryFormatter.Deserialize(stream);
    _activeDtmDeviceType = _supportedTypes.Find((item) =>
        item.Id == dtmDeviceTypeId);

    // deserialize DeviceAddress data
    int shortAddress = (int)binaryFormatter.Deserialize(stream);
    _deviceAddress = new DeviceAddress<HartDeviceAddress>(1,
        new HartDeviceAddress(shortAddress, "SHORTTAG", "Long Tag ....",
                HartDeviceAddress.AddressingModeSelection.ShortAddress,
                new HartLongAddress()));
    stream.Close();
}
```

Figure 61 — Example: Reading of DTM data from a DTMDataSubset

The DTM vendor shall consider loading of data created by an “older” version of the DTM. Even if the format of data has not changed also the deserialization of data to new class versions shall be considered. In the example in Figure 61 this is achieved by setting the BinaryFormatter in an assembly version insensitive mode.

5.14.4 Searching for data

The DTMDataSubset dictionaries provide several methods to find a particular DTMDataSubset:

- By Key
- By Descriptor

The optional DTMDataSubset.Descriptor property can be utilized by the DTM to implement advanced searching algorithms.

The content of the Descriptor property is DTM specific. A DTM can use this property to provide further information about the DTMDataSubset content. Figure 62 shows an example how a DTM may save some trend data in the BulkData dictionary with additional descriptor information.

```
protected void SaveTrend(SomeTrendData someTrendData, DateTime createdAt)
{
    // start transaction (needed for adding of DTMDataSubset)
    _dataset.StartTransaction();

    // create binary formatter and serialize TrendData
    MemoryStream stream = new MemoryStream();
    BinaryFormatter binaryFormatter = new BinaryFormatter();
    binaryFormatter.Serialize(stream, someTrendData);
    stream.Close();

    // create trend data datasubset with Descriptor containing current date / time
    byte[] data = stream.GetBuffer();
    _dataset.BulkData.Add(Guid.NewGuid().ToString(), data,
                          "TrendData - " + createdAt.ToString("yyyy:MM:dd hh:mm:ss"));

    // close transaction with auto commit = true
    _dataset.CloseTransaction(true);
}
```

Figure 62 — Example: Creation of a BulkData.DTMDataSubset with descriptor

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

The Descriptor property shall be used by the DTM to search for specific DTMDataSubsets without reading the binary data. This enables the Frame Application to read the binary data from the persistence storage only if really needed by the DTM. Thus the searching algorithm is fast and has a low memory footprint.

Figure 63 shows an example on how a DTM can search for DTMDataSubsets with specific Descriptors by using a .NET LINQ query.

```
protected List<SomeTrendData> GetTrendsOfDay(DateTime date)
{
    // (LINQ) query for all DTMDataSubsets containing trend data for a specific day
    IEnumerable<IDataSubset> dataSubsets = from item in _dataset.BulkData
                        where item.Value.Descriptor.Contains("TrendData - " +
                              date.ToString("yyyy:MM:dd"))
                        select item.Value;

    // deserialize found trend data and return list to caller
    List<SomeTrendData> trends = new List<SomeTrendData>();
    BinaryFormatter binaryFormatter = new BinaryFormatter();
    binaryFormatter.AssemblyFormat = FormatterAssemblyStyle.Simple;
    foreach (IDataSubset dataSubset in dataSubsets)
    {
        MemoryStream stream = new MemoryStream(dataSubset.ReadData());
        trends.Add(binaryFormatter.Deserialize(stream) as SomeTrendData);
    }
    return trends;
}
```

Figure 63 — Example: Searching for DTMDataSubsets with specific descriptor

5.15 Comparison of DTM and device data

5.15.1 Comparison of datasets using IDeviceData / IInstanceData

If a DTM does not provide the IComparison interface, then it shall publish all data relevant for comparison in the IDeviceData / IInstanceData interfaces (at least).

Some of the published data may not be relevant for comparison, for example dynamic data or process data. Therefore the Frame Application should provide means (e.g. user interface) to select data which is relevant for comparison.

The Frame Application shall read the data via the IDeviceData and/or the IInstanceData interface and compare the values of data items with the same identifier. If a data item with the same identifier is missing, then this shall be evaluated as not equal.

5.15.2 Comparison of datasets using IComparison

DTMs which do not publish all data shall implement the interface IComparison. If a DTM implements this interface, then the Frame Application shall use this interface for comparison.

The IComparison interface provides methods to compare

- the currently persisted dataset with the data in the device (Online Comparison)
- the currently persisted dataset with another persisted datasets (Offline Comparison)

A DTM can only compare a dataset which has a supported format (format ID is equal the current format ID or to a supported format ID). The comparison shall include only the dataset of the DTM. Related FDT Objects (e.g. Child DTMs or Parent DTMs) are not included in the comparison provided by IComparison.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

If it is necessary to compare multiple DTMs, the Frame Application is responsible to execute the comparison method on all respective DTMs. For example the comparison of a Composite Device DTM may require also the comparison for the attached Module DTMs.

5.16 Tracing

For troubleshooting or debugging trace information (logging) is essential. Whenever multiple components need to interact it is of advantage if all components have a common place to put the trace information. This makes it easier to detect and resolve problems where several components are involved.

An FDT Frame Application shall implement a dedicated interface ITrace (see 6.2), which is used by DTMs to send trace messages.

A trace message can be either a human readable description or data as an array of objects. An array of objects is useful if a DTM developer wants to trace a complete exception object and not only a description. If a DTM sends a trace message with an array of objects it shall also provide a corresponding additional message with a human readable description.

A trace message includes an assessment of severity (e.g. verbose, warning or error) and a classification. To limit the amount of trace messages sent by a DTM an FDT Frame Application can set the minimum trace level using IDtm3.

How messages are collected, stored or displayed to the user is Frame Application specific. It is not in the scope of this specification.

A trace message is not intended to be shown to the user directly. It is dedicated to debugging and troubleshooting. If a message is intended to be displayed to the user one of the message box methods of interface IFrameUi shall be used.

A trace message shall be in English. It shall not contain a timestamp, because the timestamp is provided by the Frame Application if necessary.

5.17 Report generation

5.17.1 Introduction

Due to the shared responsibilities for data management in an FDT system, the generation of a comprehensive report requires the compilation of report fragments delivered by different components in the system. While the topological information is managed by the Frame Application, all the device-specific information is to be delivered by the constituent DTMs in a project. To generate a report, a Frame Application uses the IReporting interface provided by DTMs to request report fragments with the device-specific presentation of configuration or parameterization data from each DTM.

5.17.2 Report types

Complex devices may have a huge amount of configuration and parameter information. Frame Applications shall be able to access only a subset of this data for the generation of context-specific reports, e.g. a report only of network management related data, offline or online data, IO signal information, bulk data, etc.

A DTM shall offer different types of reports, each covering a distinct subset of its device data. If the report corresponds to a DTM function (a Function ID or Application ID), it shall be referenced in the report type. A DTM may provide additional report types for specific purposes without reference to specific functions. The DTM informs the Frame Application by means of its ReportInfo property about the available report types. The list of available reports shall not change over the lifetime of a DTM BL instance.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

A Frame Application may use only the report types with an associated Application ID to generate a standardized report on a specific aspect of a system, collect the data of all report types from all DTMs to create a full report or offer a user interface based on the ReportInfo properties to let the end user decide about which data to include in the report.

5.17.3 DTM report data format

A DTM shall deliver its report fragment in form of a strictly conforming XHTML (XTHML 1.0 strict) document as specified in [19]. (see Figure 64)

Note: Since XHTML 1.0 is a reformulation of HTML 4 conforming to the XML 1.0 standard, documents with this type of markup can be processed by any XML compliant tool or library. This includes the XSL transformation to paginated output formats like XSL-FO, that can be postprocessed for example to the ISO 19005 1 (PDF/A 1) or Rich Text Format (RTF). The final report format of a Frame Application is out of scope of this specification.

The report fragment shall not contain any script or style (CSS) elements nor frames. It shall be self contained, that is it shall contain all the mandatory parts of an XHTML document, so that any XHTML compliant rendering engine can display it without further modification:

- XML Prolog with character encoding declaration. The default character encoding is UTF 8.

Note: different from the XHTML specification, this element is declared mandatory here to simplify the postprocessing with standard XML tools and libraries.

- Document type declaration (DOCTYPE) according to the XHTML 1.0 strict standard.
- root <html> element with XHTML 1.0 namespace declaration and declaration of the content language. The content language shall be the same the DTM uses in its BL and UI; xml:lang and lang-attributes shall always have the same value.
- <head> section with content type declaration including the character encoding. The encoding shall be equal to the encoding defined in the XML Prolog; the declaration in the prolog takes precedence. This declaration is for compatibility with older XHTML rendering engines.
- <body> section with presentation of the device-specific data. As any other XHTML document, report fragments may reference external resources, e.g. images.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html PUBLIC
   "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>Title of report fragment</title>
  </head>

  <body>
    … (Device-specific data presentation here)
  </body>

</html>
```

Figure 64 — Skeleton of a DTM-specific report fragment

5.17.4 Report data exchange

DTM and Frame Application exchange report fragments by means of a file system folder which can be accessed by the DTM BL. When a Frame Application requests the report fragment for a device with a call to <GenerateReport()> on the IReporting interface of a DTM BL, it specifies a destination folder path, the Base-URI. A DTM BL shall store the report result with an arbitrary filename in the destination folder and return the filename to the Frame Application as the result of the asynchronous call.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

If the report fragment needs to reference external resources, e.g. images, the DTM shall store them likewise under the specified Base-URI. The DTM is free to create subfolder under the Base-URI to organize the external resources. A report fragment shall always use relative references to link to its external resources. A DTM shall assume that the Base-URI is a temporary identifier – it is only valid until the DTM returns completion of the <GenerateReport()> call and may change between subsequent calls to <GenerateReport()>.

Frame Applications have to take appropriate measures to prevent name clashes between the URIs of report fragments and accordingly external resources of different DTMs; e.g. use different base URIs for the reports of individual instances. Furthermore they are responsible to dispose the report fragments and external resources when they are not used any more. A typical implementation of a Frame Application creates an individual subfolder as Base-URI for each DTM to be included in the report.

5.18 Security

5.18.1 General

A Frame Application hosts DTM BL or DTM WebUI which are, from the Frames perspective, external components provided by third parties. Therefore the system is exposed to possibly unknown code. The system shall be protected against security threads originating from unknown code.

5.18.2 Strong naming of assemblies

Strong naming of assemblies allows checking if an assembly was tampered after it was published.

All assemblies which are part of a DTM package shall have a strong name.

5.18.3 Identification of origin

Microsoft Authenticode [20] is a digital signature format to sign executable code, which allows i.e. checking the origin of an assembly using a public-key cryptography approach. Authenticode shall be used to allow for verification of origin and genuineness of a DTM.

DTM vendors shall obtain a code-signing certificate issued by one of the certification authorities (CAs) that are trusted by default in Windows [21]. Such a CA is referred to as “Windows root certificate program member”. Updates of the trusted root certificates in Windows are automatically installed during Windows updates or can be downloaded from the Microsoft website (see [22]).

The following DTM assemblies (DTM binaries) shall be signed using a code-signing certificate:

- DTM BL assembly
- DtmInfoBuilder if implemented in a separate assembly
- StaticFunction assemblies (if available)
- DTM package

For further information on verification of origin refer to [29].

5.18.4 Code access security

The .NET Framework provides a security mechanism referred to as “Code Access Security (CAS)” [23]. This mechanism allows to limit the access permissions (e.g. to file system, registry or network) of an assembly.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

As this could mean limiting essential capabilities of a DTM, a Frame Application is not allowed to limit code access permissions. That means Code Access Security shall not be used.

5.18.5 Validation of FDT compliance certification

The FDT Group defines an FDT compliance certification procedure for DTMs.

FDT supports the means to enable a Frame Application to validate the compliance certification of a DTM. For certified DTMs a conformity record file is provided, which is generated by an authorized FDT certification laboratory and signed using a public-key cryptography approach.

The conformity record file is digitally signed using a private key which is only known to the FDT Group and authorized certification labs. Frame Applications can check the conformity record file by checking the signature using the corresponding public key of the FDT Group.

The certification record file shall be signed according to the W3C XML Signature Syntax and Processing recommendation [24].

A DTM certification record shall use the exact same DTM vendor name as used in the Authenticode signatures of DTM BL and UI assemblies included in the DTM deployment package.

Figure 65 shows an example for a conformity record file. This is an instance of the datatype “ConformityRecord”, which is signed according to [30] (signature data of example is incomplete).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```xml
<?xml version="1.0" encoding="utf-8"?>
<ConformityRecord xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ConformityRecord.xsd">
  <UniqueCertificateNumber>Cert_1234</UniqueCertificateNumber>
  <TestedDtmName>Name of DTM</TestedDtmName>
  <TestedDtmVersion
    Build="0"
    Major="1"
    Minor="0"
    Revision="0">
  </TestedDtmVersion>
  <TestedDtmId>6d0ffd65-0936-420e-9e40-42d039fd8a98</TestedDtmId>
  <TestedTypeId>00000000-0000-0000-0000-000000000000</TestedTypeId>
  <TestedProtocolId>036d1498-387b-11d4-86e1-00e0987270b9</TestedProtocolId>
  <DateOfTest>2014-10-13</DateOfTest>
  <TestedOSVersion>
    <OSVersionNumber
      Build="7601"
      Major="6"
      Minor="1"
      Revision="65536">
    </OSVersionNumber>
    <ServicePack>Service Pack 1</ServicePack>
  </TestedOSVersion>
  <VendorName>Vendor Ltd.</VendorName>
  <TestLabName>014</TestLabName>
  <DtmInfos>
    <DtmInfo deviceType="XYZ" deviceVersion="1.0"
             deviceTypeId="8e7bda40-a042-466c-9bf8-8cabc8f06626">
      <ClassificationIds>
        <ClassificationId classificationId="Temperature"/>
      </ClassificationIds>
    </DtmInfo>
  </DtmInfos>
  <DeclaredDtmInfos>
    <DtmInfo deviceType="ABC" deviceVersion="1.0"
             deviceTypeId="7faaba0d-e6f7-4ee7-9720-fc5fa9c769aa">
      <ClassificationIds>
        <ClassificationId classificationId="Flow"/>
        <ClassificationId classificationId="Level"/>
      </ClassificationIds>
    </DtmInfo>
    <DtmInfo deviceType="BCD" deviceVersion="1.0"
             deviceTypeId="1a988802-acce-4f3b-8835-cdca8e00ec37">
      <ClassificationIds>
        <ClassificationId classificationId="Temperature"/>
      </ClassificationIds>
    </DtmInfo>
  </DeclaredDtmInfos>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
      <Reference URI="">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <DigestValue>JfVQR8C//MuCpxyqItJCVNBIeM8=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>IwCJLPq6r3zLc12Auk3ast8KrXXICLWmxHjWSYE61lLpqQQPGFwgQP2…</SignatureValue>
    <KeyInfo><X509Data>
      <X509Certificate>MIIEzTCCA7WgAwIBAgIQIvsWg8…</X509Certificate>
    </X509Data></KeyInfo>
  </Signature>
</ConformityRecord>
```


Figure 65 — Example: Conformity record file

Figure 66 shows how a Frame Application can check a certification record file.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
public static Boolean VerifyXmlFile(String fileName, X509Certificate2 certificate)
{
    // Create a new XML document.
    XmlDocument xmlDocument = new XmlDocument();

    // Load the passed XML file into the document.
    xmlDocument.Load(fileName);

    // Find the "Signature" node and create a new
    // XmlNodeList object.
    XmlNodeList nodeList = xmlDocument.GetElementsByTagName("Signature");
    if (nodeList == null || nodeList.Count != 1)
    {
        return false;
    }

    // Create a new SignedXml object and pass it
    // the XML document class.
    SignedXml signedXml = new SignedXml(xmlDocument);

    // Load the signature node.
    signedXml.LoadXml((XmlElement)nodeList[0]);

    // Check the signature and return the result.
    return signedXml.CheckSignature(certificate, false);
}
```

Figure 66 — Example: checking conformity record file

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

6 FDT Objects and interfaces

6.1 General

The FDT interface specification includes the following FDT Objects:

- DTM Business Logic
- DTM User Interface objects
    - DTM WebUI
- Communication Channel
- Frame Application
    - Frame Application Business Logic
    - Frame Application User Interface
    - Frame Application WebUI (FA WebUI)

The behavior of these objects and their interfaces are described in this clause. Developers implementing DTMs or parts of Frame Application like storage or communication objects shall implement the functionality as defined in this clause.

This clause also references and defines expected behavior of FDT-specific interfaces that FDT compliant objects shall implement.

In order to describe the availability of interfaces for the different FDT objects, following abbreviations are used:

M: mandatory – the interface shall be provided

C: conditional – the interface shall be provided depending on conditions

O: optional – the interface may be provided based on product decisions

-: not allowed – the interface shall not be provided

6.2 Frame Application

6.2.1 General

A Frame Application in general may be structured into 2 tiers: Business Logic and Graphical User Interface (GUI). In order to support different use cases (see for example section 11), this specification supports different type of GUI for a Frame Application. That is why, the document describes 3 main components of a Frame Application:

- Frame Application Business Logic, represents the business logic of the application, which implements the structure and behavior (business rules) according to its supported use cases (see 4.3.1).
- Frame Application User Interface, represents the conventional GUI of the application, which may be used for example in a standalone architecture (see 11.2) or a distributed application (see 11.4).
- Frame Application WebUI (FA WebUI), represents an HTML5 based GUI of the application, which is intended to support distributed application scenarios (see 11.3 and 11.4).

6.2.2 Frame Application Business Logic

The following class diagram (Figure 67) shows the interfaces, which shall be implemented by a Frame Application Business Logic.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

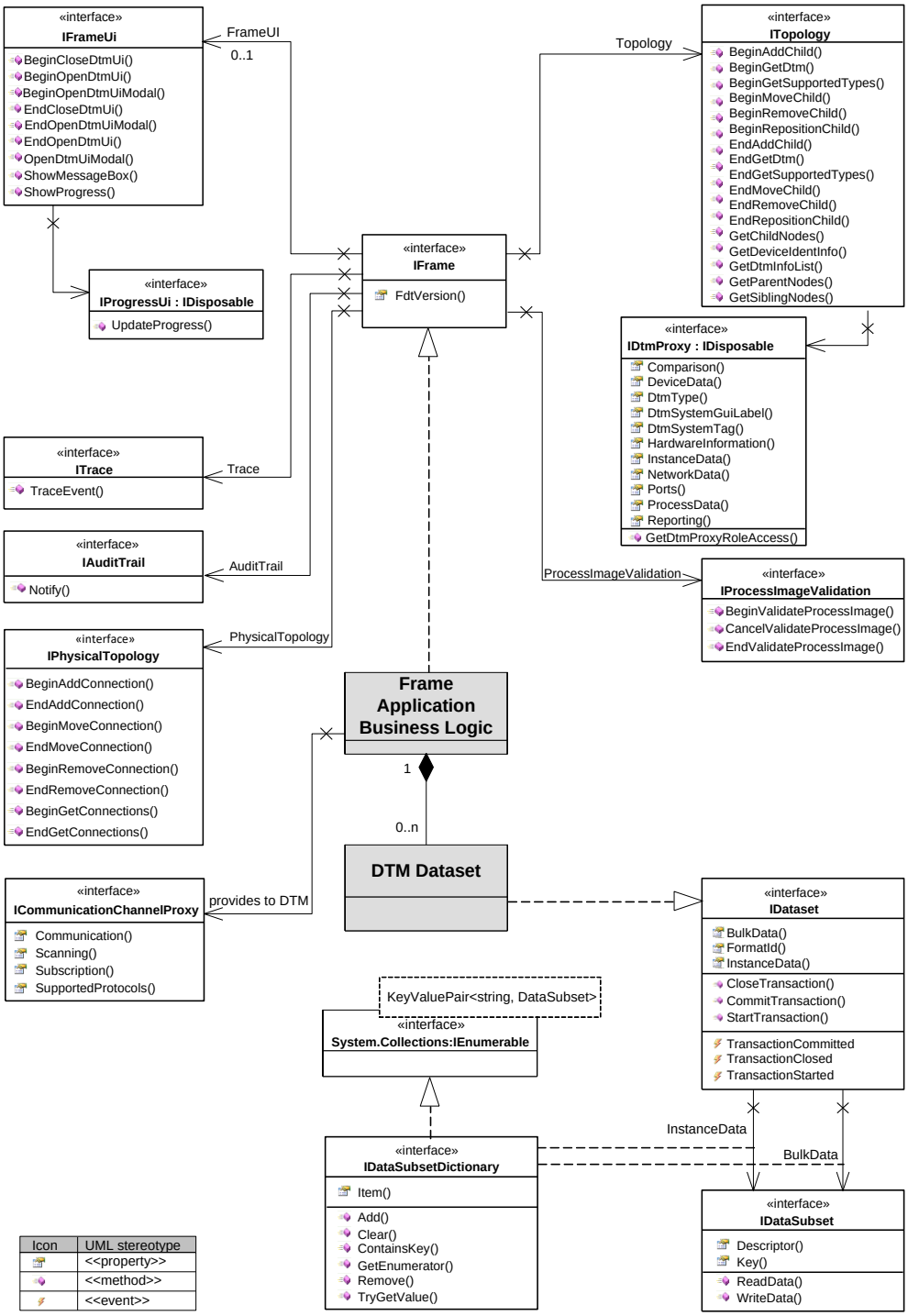


Figure 67 — Frame Application BL interfaces

The Frame Application BL implements the IFrame interface which is passed to the DTM Business Logic. The properties of IFrame interface, named FrameUi, Topology, Trace and AuditTrail, provide access to the corresponding interfaces IFrameUi, ITopology, ITrace and IAuditTrail (see Table 7).

ICommunicationChannelProxy is implemented as part of the Frame Application and is provided to the DTM in IDtm3.EnableCommunication2().

DTMDataSet instances are provided from Frame Applications internal persistent storage for each device node (DTM instance) in the topology. The corresponding IDataset interface is implemented by Frame-Application-specific instances (shown as DTMDataSet class in the

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

diagram) for each device node and passed to the DTM Business Logic. The DTMDataSubsets collected in DTM InstanceData or in DTM BulkData implement the interface IDataSubset.

Table 7 — Frame Application BL interfaces

Interface	Availability	Description
IAuditTrail	M	Interface used to receive audit trail events from DTMs in order to record changes and actions performed on a device.
ICommunicationChannelProxy	M	Proxy interface which enables a DTM to interact with the linked Communication Channel provided by the Parent DTM in the FDT topology.
IDataset	M	Interface used to read and store DTM instance-specific data in a dataset
IDataSubset	M	The DTMDataSubsets contain the actual DTM persistent data.
IDataSubsetDictionary	M	Represents a collection of data subsets.
IDisposable		.NET interface for disposable objects.
IDtmProxy	M	This interface is provided by DTM proxy objects. These objects enable a DTM to interact with another DTM instance (represented by the proxy object).
IFrame	M	The IFrame interface is the main interface of a Frame Application. It includes the services that shall be provided by the Frame Application to the DTM Business Logic and the DTM WebUIs. The reference to this interface is passed to the DTM Business Logic and to the DTM WebUI in the call. The interface provides references to further interfaces.
IFrameUi	C	This interface provides access to the Frame Application user interface. A Frame Application that provides a local user interface shall provide this interface. If the Frame Application does not provide this interface, the DTM knows explicitly that no GUI is available. A DTM shall be able to adapt to the situation where it cannot show a user interface.
IPhysicalTopology	O	Interface used to manage physical connections between DTMs. The ability to manage physical connections depends on the availability of the IPorts interface at a DTM.
IProcessImageValidation	O	In some automation systems it is a requirement to apply changes to the process image while the PLC is running. This interface provides the methods needed to validate whether a potential change can be applied while the PLC is running.
IProgressUi	C	Interface to a Frame Application progress user interface opened by IFrameUi.ShowProgress. A Frame Application that provides a local user interface shall provide this interface. The interface can be used by DTM WebUI to show progress information. For asynchronous operations that are executed in the DTM BL the progress mechanism of extended AsyncResult pattern shall be used.
ITopology	M	This interface provides the access to the FDT topology. A DTM can request and release references to other DTM instances as well as create and remove Child DTMs.
ITrace	M	Trace interface that shall be used by DTMs to inform a Frame Application about trace message.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

6.2.3 Frame Application WebUI

In order to support distributed architectures (e.g. in context of IIoT), a Frame Application may provide a GUI based on web-server technologies. This FA WebUI will host the WebUIs provided by DTMs and provides the IDtmWebUiMessaging interface to the DTM WebUi, so that the DTM WebUI may interact with its DTM BL.

The FA WebUI and the DTM WebUI interact not only graphically, but also by means of JavaScript APIs.

The Frame WebUI provides the WebData connector for accessing the data of the DTM BL (see Figure 68).

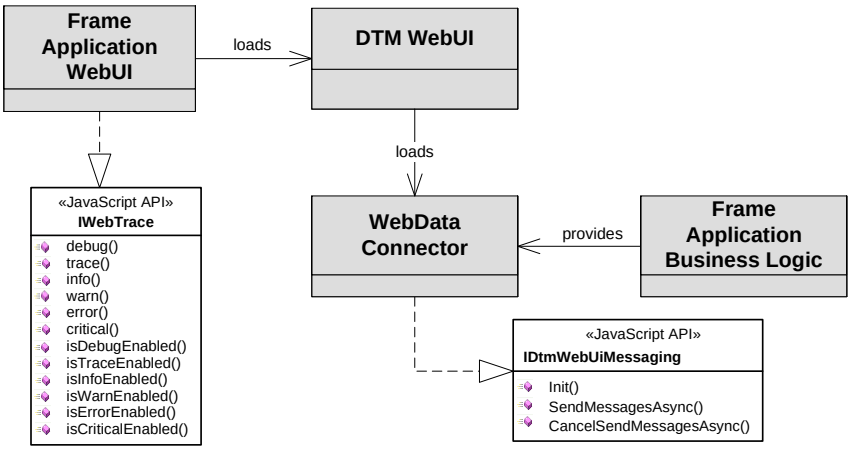


Figure 68 — Frame Application WebUI

Table 8 — Frame Application WebUI interfaces

Web Interface	Availability	Description
IDtmWebUiMessaging	M	This JavaScript API is provided by the Frame Application. It enables the DTM WebUI to interact with the DTM BL.
IWebTrace	M	JavaScript Trace interface that shall be used by DTM WebUIs to inform a Frame Application about trace message.

6.3 DTM Business Logic

6.3.1 DTM BL interfaces

The following class diagrams (Figure 69 and Figure 70) show the interfaces, which shall be implemented by a DTM Business Logic class. IDtm3 is implemented by the DTM Business Logic and provides access to all other interfaces by corresponding properties.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

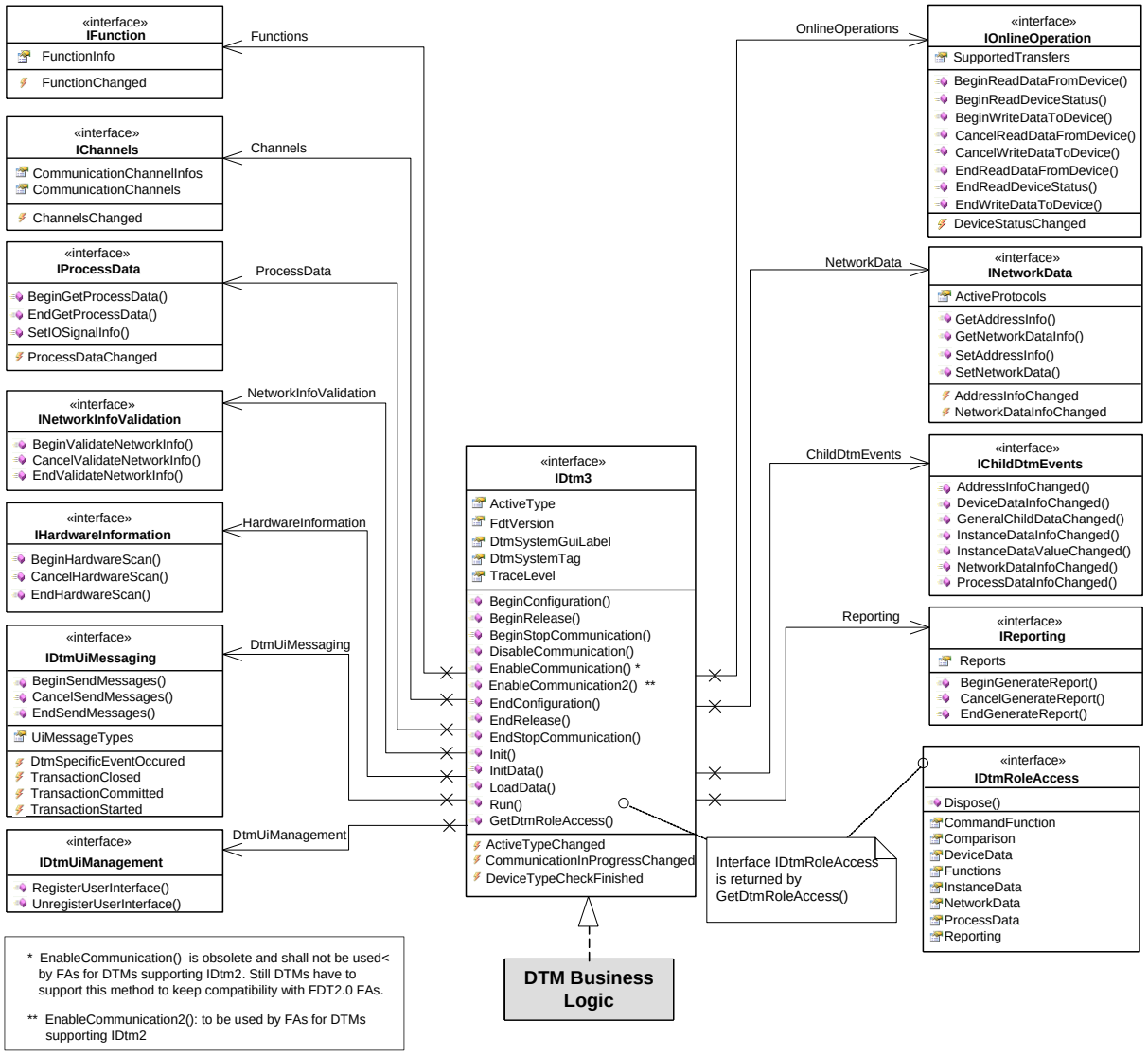


Figure 69 — DTM Business Logic interfaces (Part 1)

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

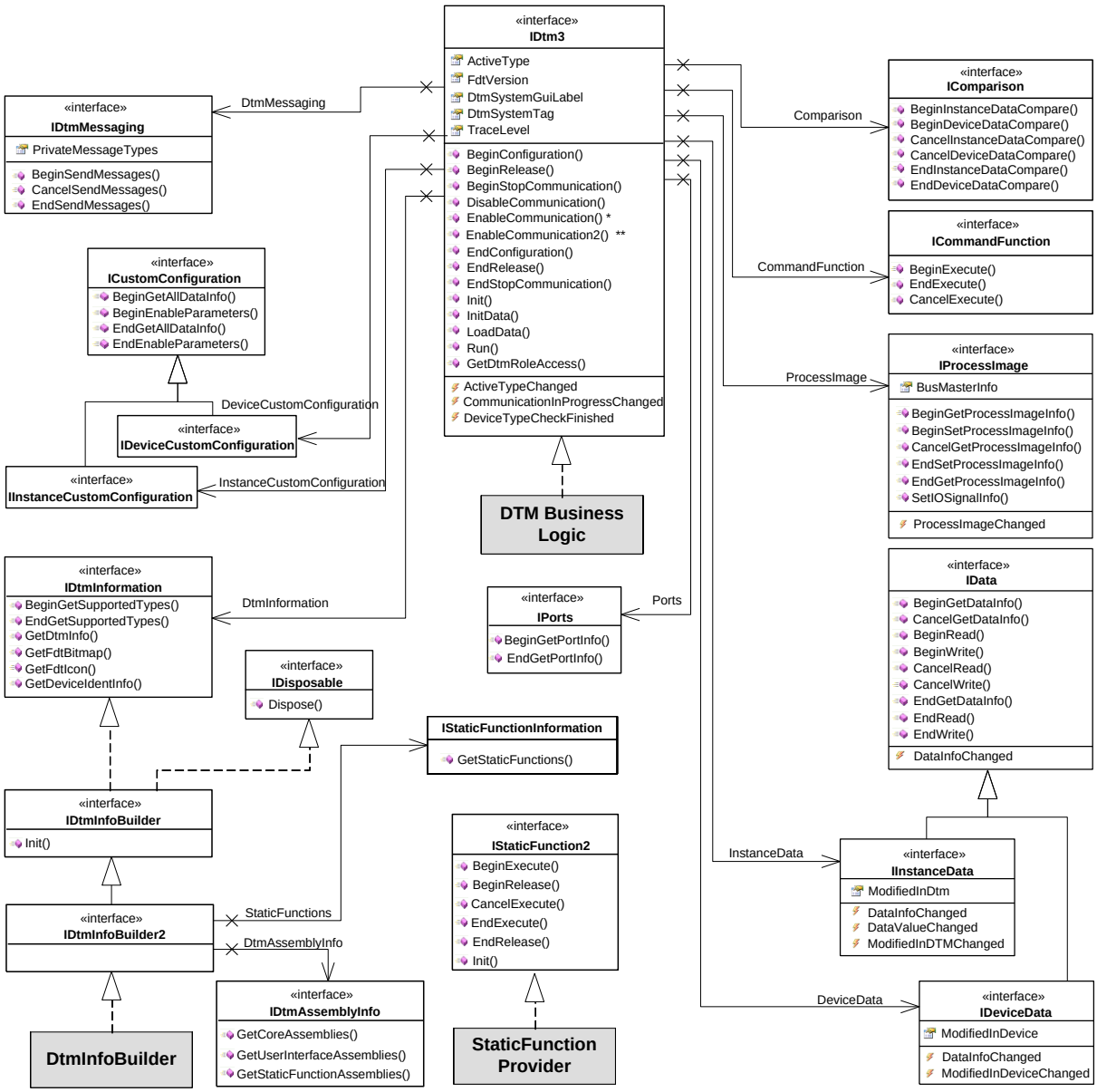


Figure 70 — DTM Business Logic interfaces (Part 2)

There is no state machine defined for DtmInfoBuilder instances. DtmInfoBuilder objects are created with new() and destroyed with Dispose().

Table 9 provides an overview on DTM interfaces.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 9 — DTM Business Logic interfaces

Interface	Description
IChannels	This interface is used for accessing the Communication Channel objects of a DTM.
IChildDtmEvents	Interface used by the Frame Application to inform the DTM about events occurred in a Child DTM in the FDT topology.
ICommandFunction	This interface is used to execute command functions.
IComparison	This interface allows a Frame Application to request the DTM to compare the dataset with another dataset or with the data in the physical device.
IDeviceData	This interface provides online access to specific parameters of a device.
IDtm3	This is the main interface of a DTM according FDT3.0. This interface is mandatory for implementations according to FDT3.0. It defines the methods to control the DTM state-machine and general properties. The interface is the successor of the FDT2.1 interfaces IDtm2 and IDtm. These interfaces are documented in [3].
IDtmAssemblyInfo	This interface is used to provide information about the main DTM assemblies. (In some situations, for instance if the DTM is built using the DTM-CC, the Frame Application has no access to the assemblies.) The Frame Application shall use this interface to retrieve information about the main DTM assemblies (assemblies providing objects with IDtm3, IDtmInfoBuilder2, IStaticFunction2 interfaces). The target is to verify the assemblies against the DTM certification record. The interface shall be implemented by the DtmInfoBuilder.
IDtmInfoBuilder2	This interface is used to initialize and shutdown a DtmInfoBuilder object according to FDT3. This interface is mandatory for implementations according to FDT3. The Frame Application can use a DtmInfoBuilder to retrieve information about the DTM and its supported types.
IDtmInformation	This interface provides general information about the DTM itself and the supported device types.
IDtmMessaging	This interface is used for interaction between the DTM Business Logic of two DTMs (Composite and Module DTM).
IDtmRoleAccess	This is the main interface for role based access (for a DTM representing a multirole device). It provides access to the role based interfaces.
IDtmUiManagement	Interface used for managing the session between DTM WebUI and DTM BL.
IDtmUiMessaging	Interface used for interaction between the Business Logic and DTM WebUIs.
IFunction	This interface provides access to functions, user interfaces and documents provided by a DTM.
IHardwareInformation	This interface is used by Frame Application to request hardware information from a device.
IInstanceData	This interface provides access to DTM instance data parameters.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Interface	Description
INetworkData	This interface provides network management relevant information.
INetworkInfoValidation	In some automation systems it is a requirement to apply changes to the Network Info (which leads to a change in process image) while the PLC is running. This interface provides the needed methods to validate if a potential change can be applied while the PLC is running.
IOnlineOperation	This interface allows a Frame Application to request the DTM to exchange online data with the device.
IPorts	The interface allows to request a list of ports from the DTM.
IProcessData	This interface provides information related to process data of a field device for the integration of the device into the control system like datatype, signal direction, engineering units, and ranges etc.
IProcessImage	This interface provides access to the description of the process image provided by a fieldbus master.
IReporting	This interface is used to report the current instance or device dataset of a DTM (online data allowed here), e.g. for printing or documentation. Note: The Frame Application may generate reports using IInstanceData/IDeviceData interfaces
IStaticFunction2	This interface is used to execute Static Functions independently of the DTM.
IDeviceCustomConfiguration IInstanceCustomConfiguration	These are optional interfaces. Only DTMs that allow customization of parameter access for User Level “Expert” need to implement these interfaces. These interfaces are supported only when the DTM is in the running state, before any function is invoked on the DTM. In all other states, the DTM shall restrict access to these interfaces.

Table 10 defines under which conditions interfaces shall be implemented.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 10 — Availability of interfaces depending of type of DTM

Interface	Condition	Device DTM	Communication DTM	Gateway DTM	Composite Device DTM	Module DTM	BTM
IChannels	Interface shall be provided by all DTMs that provide communication access to other DTMs.	O *)	M	M	M	O *)	-
IChildDtmEvents	Interface shall be provided by all DTMs that provide communication access to other DTMs.	O *)	M	M	M	O *)	-
ICommandFunction		O	O	O	O	O	O
IComparison	Interface shall be provided if not all parameters of the DTM/device can be accessed by IInstanceData / IDeviceData interfaces.	C	C	C	C	C	C
IDeviceData	Interface shall be provided for all devices which have online data.	C	C	C	C	C	C
IDeviceCustom-Configuration		O	O	O	O	O	O
IDtm3		M	M	M	M	M	M
IDtmAssemblyInfo		M	M	M	M	M	M
IDtmInfoBuilder2		M	M	M	M	M	M
IDtmInformation		M	M	M	M	M	M
IDtmMessaging	May be implemented in case a tight coupling between two DTMs of the same vendor is required.	O	O	O	O	O	O
IDtmRoleAccess	Interface shall be provided when a multirole device is supported.	C	C	C	C	C	C
IDtmUiManagement	Interface shall be provided for DTMs with user interfaces.	C	C	C	C	C	C
IDtmUiMessaging	Interface shall be provided for DTMs with user interfaces.	C	C	C	C	C	C
IFunction		M	M	M	M	M	M
IHardwareInformation		M	M	M	M	M	M
InstanceCustom-Configuration		O	O	O	O	O	O
IInstanceData		M	M	M	M	M	M
INetworkData		M	M	M	M	M	M
INetworkInfo-Validation	Only implemented by DTMs which represent a fieldbus Master and which are used in automation systems with specific requirements.		O	O			
IOnlineOperation	Interface shall be provided for all devices which have online data and shall be loaded during commissioning.	C	O	O	C	C	C
IPorts	The protocol-specific specification annex defines the rules for this interface.	M	M	M	M	M	-

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Interface	Condition	Device DTM	Communication DTM	Gateway DTM	Composite Device DTM	Module DTM	BTM
IProcessData	The protocol-specific specification annex, defines the rules for this interface. If the protocol supports process data and the respective device provides process data the DTM shall provide this interface.	C	C	C	C	-	-
IProcessImage	Interface shall be provided by Communication- / Gateway-DTMs that provide the layout of a process image of a master device.	-	C	C	-	-	-
IReporting	Interface shall be provided to support advanced reporting capabilities.	M	M	M	M	M	M
*) Optional for DeviceDTM and ModuleDTM for instance because of possible BTM support.							

For a DTM, which has set the flag TypeInfo.CommunicationSupport to value ‘PassiveDevice’, all interfaces related to communication (i.e. IChannels, IDeviceData, IHardwareInformation, IOnlineOperation, IProcessImage) shall not be supported.

6.3.2 State machines related to DTM BL

6.3.2.1 General

The following state machines describe the behavior of the DTM in regard to its interfaces. The states are defined mainly to describe how a DTM is guided through different stages by a Frame Application and which interface methods can be used at a specific stage of the lifetime of a DTM instance. The state machine is based on the general state machine as defined in IEC 62453-2. It is extended to accommodate the specific needs of .NET based implementation. The state machines provided here are intended as a general specification for all types of DTM and are not intended as implementation design (e.g. in order to support the “{enter state}” triggers an implementation might define additional states).

For information on which interface methods can be used at specific states refer to 6.6.

6.3.2.2 DTM state machine

The DTM State Machine in Figure 71 shows the states and transitions that are controlled by a Frame Application (the Frame Application has full control).

The diagram uses following notation:

- Method(): denotes a method used as a trigger for a state transition, the transition taken only, when the respective method returns
- [condition expression]: denotes a condition (guard) that has to evaluate true for the transition to be taken
- <method_name>: denotes an asynchronous operation

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

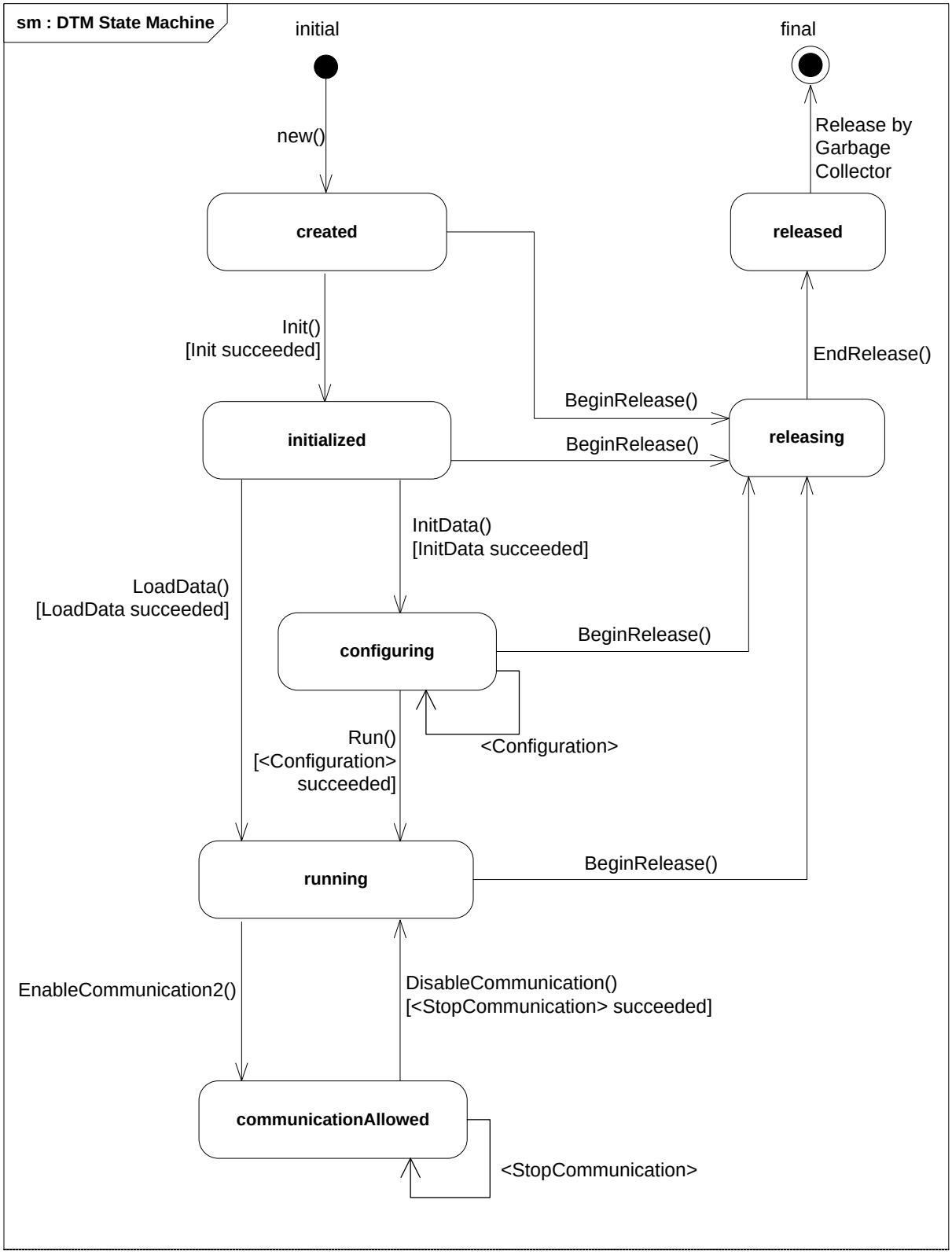


Figure 71 — State machine of DTM BL

Table 11 provides a description of the state transitions with their conditions and actions.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 11 — Definition of DTM BL state machine

#	Start state	End state	Trigger	Condition
1	initial	created	new()	new() succeeded
2	created	initialized	Init()	Init() succeeded
3	created	releasing	BeginRelease()	BeginRelease() succeeded
4	initialized	running	LoadData()	LoadData() succeeded
5	initialized	configuring	InitData()	InitData() succeeded
6	initialized	releasing	BeginRelease()	BeginRelease() succeeded
7	configuring	configuring	BeginConfiguration()/ EndConfiguration()	<Configuration> succeeded
8	configuring	running	Run()	<Configuration> succeeded and Run() succeeded
9	configuring	releasing	BeginRelease()	BeginRelease() succeeded
10	running	communication Allowed	EnableCommunication2()	Reference to parent CommunicationChannel is valid and EnableCommunication2() succeeded
11	running	releasing	BeginRelease()	all user interfaces are closed, all operations are finished
12	communication Allowed	communication Allowed	BeginStopCommunication()/ EndStopCommunication()	<StopCommunication> succeeded
13	communication Allowed	running	DisableCommunication()	<StopCommunication> succeeded and DisableCommunication() succeeded
14	releasing	released	EndRelease() *)	
15	released	final	Removal by .NET GarbageCollector	
*) This transition is taken, even if the method failed.				

6.3.2.3 Online state machine

The following state machine (Figure 72) shows the internal states of state “communicationAllowed”. The DTM controls the internal states according to this state machine. The DTM does not expose the substate, but fires events which inform the Frame Application about internal state transitions (OnlineStateChanged event). In order to prepare an exit from the state “communicationAllowed”, the Frame Application performs the asynchronous <StopCommunication()> operation. The actual exit from state “communicationAllowed” is triggered by DisableCommunication() (see Table 11).

The state machine is used to define state-dependent interface and method availability in chapter 6.6.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

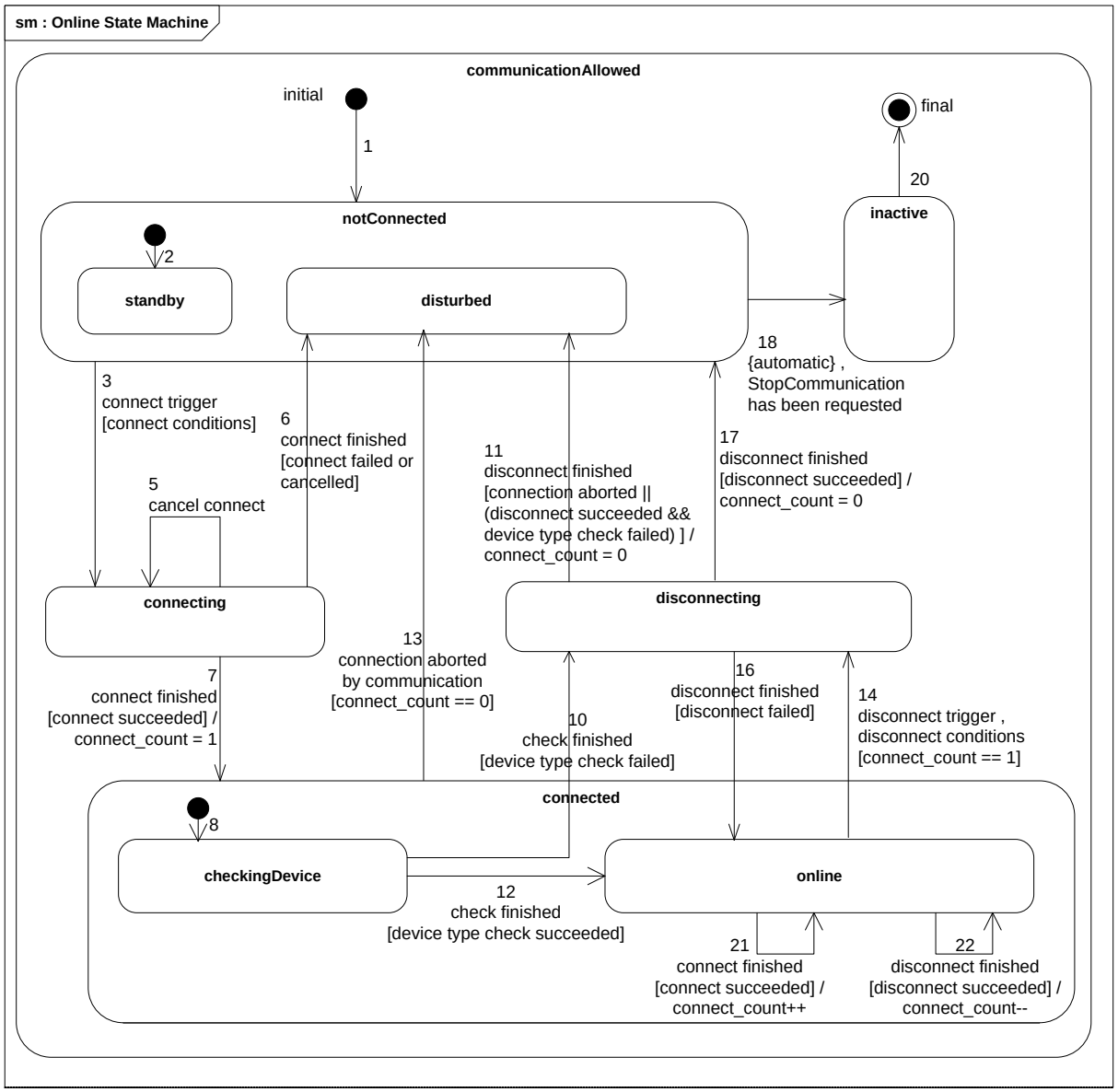


Figure 72 — Online state machine of DTM

Additional to the diagram notation explained above, Table 12 also shows triggers, that occur automatically.

The trigger {automatic} is a trigger, that activates automatically after the start state of the transition has been reached (spontaneous transition).

The trigger {enter state} fires when a state is reached.

These triggers are not associated to specific transitions, but fire every time, when a transition leads into the state. That is why those triggers are shown in the table without number and only with the start state.

When the DTM is in ‘online’ state, it may request additional connections to the device (e.g. by calling <Connect()> on the parent channel). These requests do not trigger a transition to another state, but if the creation of the additional connection succeeds, the count of connections (connect_count) is incremented (transition 21).

If the DTM wants to terminate a connection and the count of connections is larger than 1, the request to terminate the connection (e.g. a call to <Disconnect()> on the parent channel) does

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

not trigger a state transition, the DTM stays in state 'online' until it terminates the last connection. If the termination of an additional connection succeeds, the count of connections (connect_count) is decremented (transition 22).

Table 12 — Definition of online state machine

#	Start state	End state	Trigger	Condition	Action
1	communicationAllowed	notConnected	{automatic}		
2	notConnected	standby (notConnectedStandby)	{automatic}		raise OnlineStateChanged(notConnectedStandby)
3	notConnected	connecting	connect trigger Several triggers possible: For Child DTMs: - Immediate connect because "StayConnected" was requested in EnableCommunication2() - online function started (e.g. Download or Online-GUI) - reconnect after lost connection For CommDTM: - Immediate connect because "StayConnected" was requested in EnableCommunication2() - online function started(e.g. Scan, Download or Online-GUI) - Child DTM requested connection +)	<StopCommunication> has not been called	
4	connecting ++)		{enter state}		raise OnlineStateChanged (connecting) initiate connection: For Child DTMs: - Call <Connect()> on parent channel For Comm DTMs +): - Use driver API to connect.
5	connecting	connecting	cancel connect		CancelConnect() +)
6	connecting	disturbed (notConnectedDisturbed)	connect finished	connect failed or cancel succeeded	raise OnlineStateChanged(NotConnectedDisturbed)

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

#	Start state	End state	Trigger	Condition	Action
7	connecting	connected (connect_count = 1)	connect finished	connect succeeded	set connect_count=1
8	connected	checkingDevice (connectedCheckingDevice)	{automatic}		raise OnlineStateChanged(ConnectedCheckingDevice)
9	checkingDevice		{enter state}		Depending on the parameter deviceTypeCheckMode, perform device type check. *)
10	checkingDevice	disconnecting	device type check finished	device type check failed	raise OnlineStateChanged (Disconnecting) raise DeviceTypeCheckFinished(UnsupportedDevice)
11	disconnecting	Disturbed (notConnectedDisturbed)	disconnect finished	connection aborted by communication || (disconnect succeeded && device type check failed)	set connect_count=0
12	checkingDevice (connectedCheckingDevice)	online (connectedOnline)	device type check finished	device type check succeeded	raise OnlineStateChanged(ConnectedOnline) raise DeviceTypeCheckFinished(SupportedDevice)
13	connected	notConnected	connection aborted by communication Abort notification received from parent Communication Channel+)	there are no connections (connect_count == 0)	For Child DTMs: - Handle pending transactions or active online functions. **) For ParentDTMs: - Abort all child connections
14	connected (connect_count = 1)	disconnecting	disconnect trigger or disconnect condition Several triggers possible: - all online functions finished (and DTM is in ConnectionMode “OnDemand”) - call to <StopCommunication> has been received - Child DTM calls <Disconnect>	there is only one established connection (connect_count == 1)	
15	disconnecting		{enter state}		raise OnlineStateChanged(Disconnecting) terminate connection: For Child DTMs: - Call <Disconnect()> on parent channel For Comm DTMs +): - Use driver API to disconnect.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

#	Start state	End state	Trigger	Condition	Action
16	disconnecting	online (connectedOnline)	disconnect finished	disconnect failed	raise OnlineStateChanged(ConnectedOnline)
17	disconnecting	notConnected	disconnect finished	disconnect succeeded	set connect_count=0
18	notConnected	inactive	{automatic} or <StopCommunication> has been requested		
19	inactive		{enter state}		raise OnlineStateChanged(Inactive)
20	inactive	final	{automatic}		<StopCommunication> completed callback
21	online (connect_count > 0)	online (connect_count++)	an additional connection has been established and 'connect finished' callback has been received	connect succeeded	increment connect_count
22	online (connect_count >1)	online (connect_count--)	an additional connection has been terminated and 'disconnect finished' callback has been received	disconnect succeeded	decrement connect_count

Notes:

*) Device type check means, that the DTM checks if it is connected to the correct device type. Device type check is performed depending on the parameter 'deviceTypeCheckMode' when state "checkingDevice" is entered. The Frame Application receives an event DeviceTypeCheckFinished, when the DTM leaves the state "checkingDevice". The Frame Application may request to execute no device type check, to execute the device type check once (when the DTM goes online for the first time), or every time when the DTM goes online. Also, for some devices a device type check may not be feasible. In all cases when no device check is performed, the DTM shall raise DeviceTypeCheckFinished event with 'NotChecked' value.

**) Asynchronous operations shall always be finished by calling the 'Completed' callback method. If a connection is aborted, each aborted transaction will raise an FdtConnectionAbortedException in its 'End'-method.

+) Communication DTMs do not call <Connect> or <Disconnect> as they do not have a parent Communication Channel. Instead Communication DTMs work on a driver API. For the same reason, the abort notification is not valid for Communication DTMs, but a Communication DTM may receive a similar notification from the driver.

++) If <StopCommunication> is called in state 'connecting', then the connection establishment is continued (until it succeeds, fails or until it is cancelled) and <StopCommunication> is handled in the next state (state 'connected' or state 'disturbed').

6.3.3 State machine of instance data

6.3.3.1 General

A DTM BL shall expose the state of the actual instance data to the Frame Application in order to support Frame Applications in synchronizing DTM datasets with their respective devices. Two properties reflect the possible states of the data (instance data and online data) in regard to modifications (see Figure 73):

- modification in DTM: IInstanceData.ModifiedInDtm (see state machine in Figure 74) reflects changes in the instance data and
- modification in device: IDeviceData.ModifiedInDevice (see state machine in Figure 75) reflects changes in the online data.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

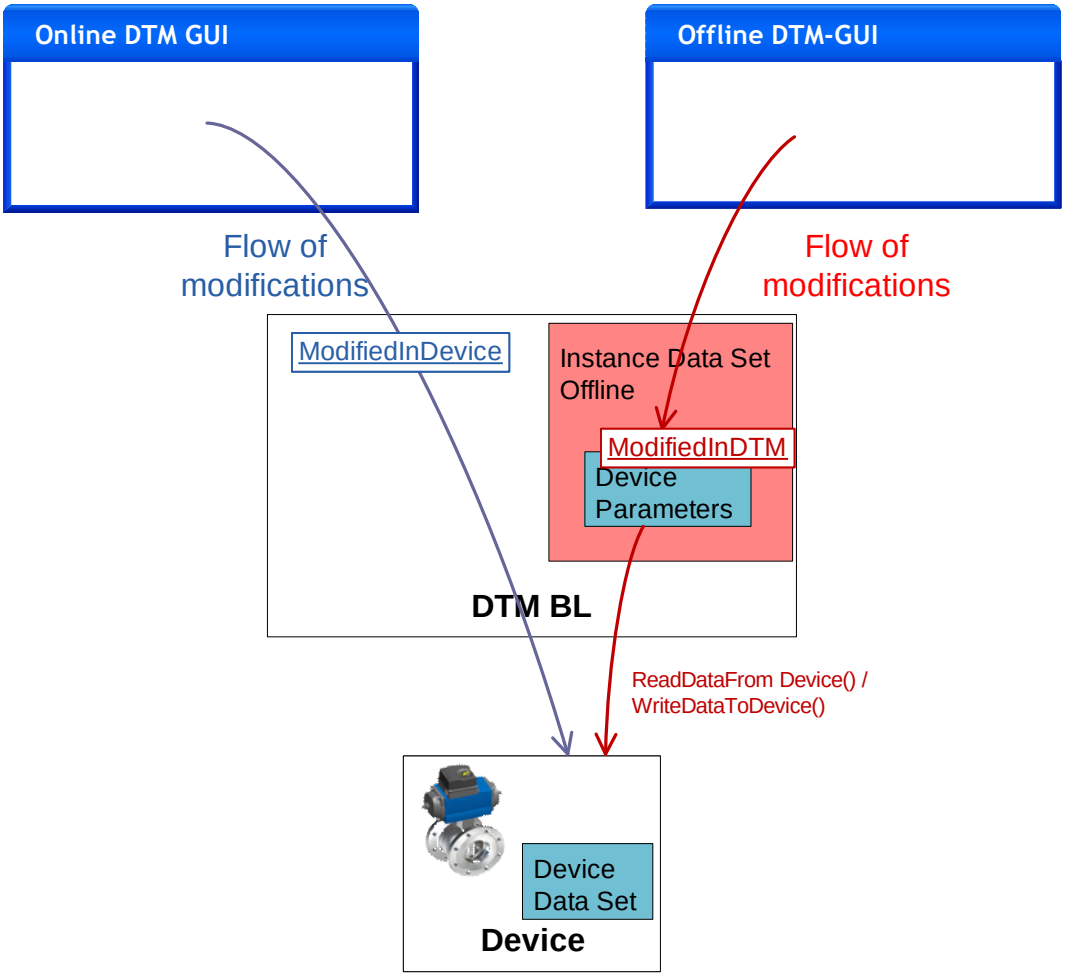


Figure 73 — Modifications of data through a DTM

Note: For description of the concept of Instance Data and Device Data see 4.15.1.

If the DTM supports the methods <ReadDataFromDevice()> and <WriteDataToDevice()>, then the Frame Application may use these methods for synchronization of Instance Data Set and Device Data Set. A DTM indicates in the property IOnlineOperation.SupportedTransfers whether the respective device supports these methods.

6.3.3.2 Modifications in DTM

The property ModifiedInDtm can be used by a Frame Application to detect which DTMs have modification of offline data that are not synchronized with the respective device.

The state “default” indicates the initial status of the dataset (after InitData()).

Any offline modification of device parameters will lead to a state not equal to “default” (device parameters here means subset of offline data that is synchronized with the device).

<ReadDataFromDevice()> or <WriteDataToDevice()> change the state to “dataLoaded”.

The state shall be exposed in IInstanceData property ModifiedInDtm and shall be read only (see Figure 74). The DTM shall include the state in its persisted instance dataset and set the state accordingly in LoadData(). When the state changes, the DTM fires an IInstanceData.ModifiedInDtmChanged() event.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

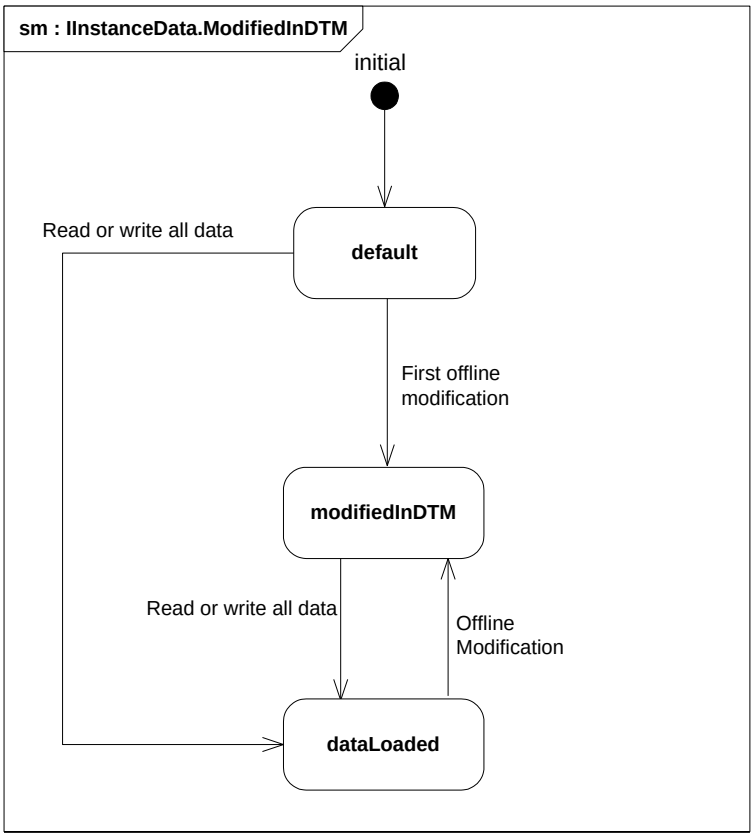


Figure 74 — ModifiedInDtm: State machine of instance data

The meaning of the different states can be seen in Table 13.

Table 13 — Description of instance dataset states

State	Meaning
default	This state is set after creation of a new instance dataset in InitData(). The state is only valid if the newly created dataset contains enough information to establish a proper communication.
modifiedInDTM	The offline instance dataset is modified and not synchronized with the device.
dataLoaded	The offline instance dataset has been synchronized with the device. No further change has been executed on the instance dataset since the synchronization.

6.3.3.3 Modifications in device

The property ModifiedInDevice can be used by a Frame Application to detect which DTMs have modified the data in the device and have not synchronized their DTM dataset. Any change to the device, which is performed or recognized by the DTM will lead to a state “modifiedOnline”.

Note: The status should indicate all changes in data used to configure the device. Changes in dynamic values (e.g. operating hours) should not be reflected.

The state shall be exposed in IDeviceData property ModifiedInDevice and shall be read only. (see Figure 75) The state shall be included in the persisted instance dataset. When the state changes, an IDeviceData.ModifiedInDeviceChanged() event is fired.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

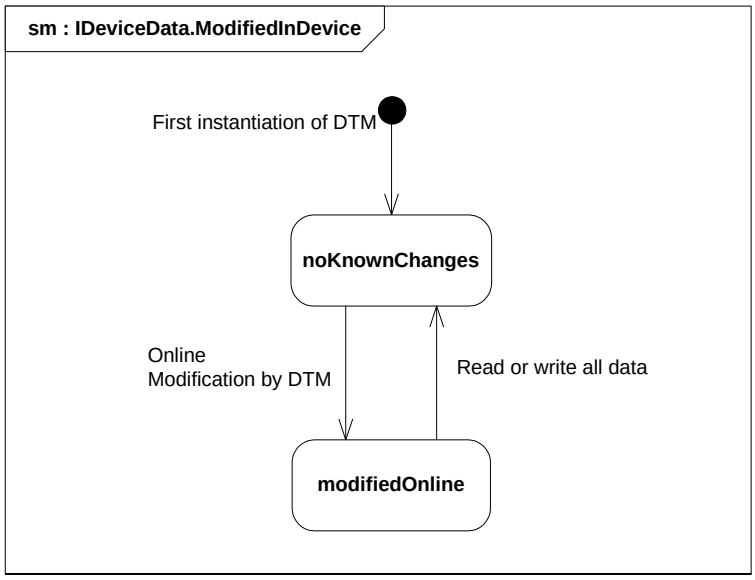


Figure 75 — ModifiedInDevice: State machine related to device data

The meaning of the different states can be seen in Table 14.

Table 14 — Description of dataset states regarding online modifications

State	Meaning
noKnownChanges	The dataset state regarding the device is unknown because • the DTM was not connected to the device or • the DTM has synchronized at some point of time with the device. The dataset has been uploaded (<ReadDataFromDevice()> or downloaded (<WriteDataToDevice()>). No further change has been executed on the device by the DTM. But there may be changes on the device, which were triggered from other sources.
modifiedOnline	Parameters have been changed in the device but not in instance dataset (E.g.: see use case Online parameterization, IDeviceData interface definition) 'modifiedOnline' status shall be set only once in case the data in the device has been changed by the DTM. In case of successful Upload or Download of complete dataset, the state shall be set to "noKnownChanges".

Data in the device can also be modified directly by a tool out of the scope of the FDT. In this case, it is recommended not to set the status to 'modifiedOnline'.

If an application is started, which may need to change the state in ModifiedInDevice (the property is part of the instance dataset and cannot be changed when the dataset is not locked), then the dataset shall be locked (StartTransaction()).

For special operations it is useful to keep the device configuration and the instance dataset in sync. Therefore it is strongly recommended that the DTM should ask the user whether the data should be synchronized. This is necessary for user interface functions like Online Parameterization and Offline Parameterization (see Table A.4).

IDeviceData methods shall not modify the instance dataset, but shall set the state in ModifiedInDevice.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

6.4 DTM User Interface

6.4.1 DTM WebUI

DTM WebUI can be embedded into Frame Application WebUI, which can be executed in a web browser. The DTM WebUI shall be based on HTML and JavaScript (see 5.2 and 5.3 for supported versions).

Figure 76 shows, that DTM WebUI expose a JavaScript API (IDtmWebUiFunction), which is used by the FA WebUI to control the DTM WebUI.

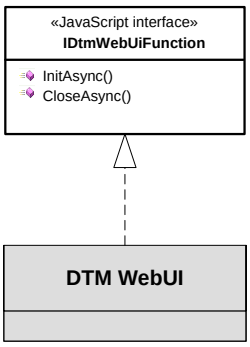


Figure 76 — DTM WebUI interfaces

Table 15 — DTM WebUI interfaces

Interface	Availability	Description
IDtmWebUiFunction	M	This is the main JavaScript API of a DTM WebUI.

The DTM WebUI which implements IDtmWebUiFunction shall be added to the global namespace as object 'DtmWebUi'. This enables the FA WebUI to use this global variable to get access to the DTM WebUI.

Note: Figure 77 shows an example for a DTM WebUI defining the DtmWebUi object.

```
class myDtmUi implements FdtWebUi.IDtmWebUiFunction {

    constructor() {
        window['DtmWebUi'] = this;
        …
    }
}
```

Figure 77 — Example: Definition of DtmWebUi object

Note: Figure 78 shows an example for a Frame Application WebUI accessing the DtmWebUi object.

```
    const dtmWebUi = window['DtmWebUi'] as FdtWebUi.IDtmWebUiFunction;
    dtmWebUi.initAsync(…);
```

Figure 78 — Example: Access to DtmWebUi by Frame WebUI

6.5 Communication Channel

The following class diagram (Figure 79) shows the interfaces, which shall be implemented by a Communication Channel. A Communication Channel implements the main interface ICommunicationChannel and the interfaces ICommunication, ISubscription, IScanning, and ISubTopology, which are accessible by corresponding properties of ICommunicationChannel.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

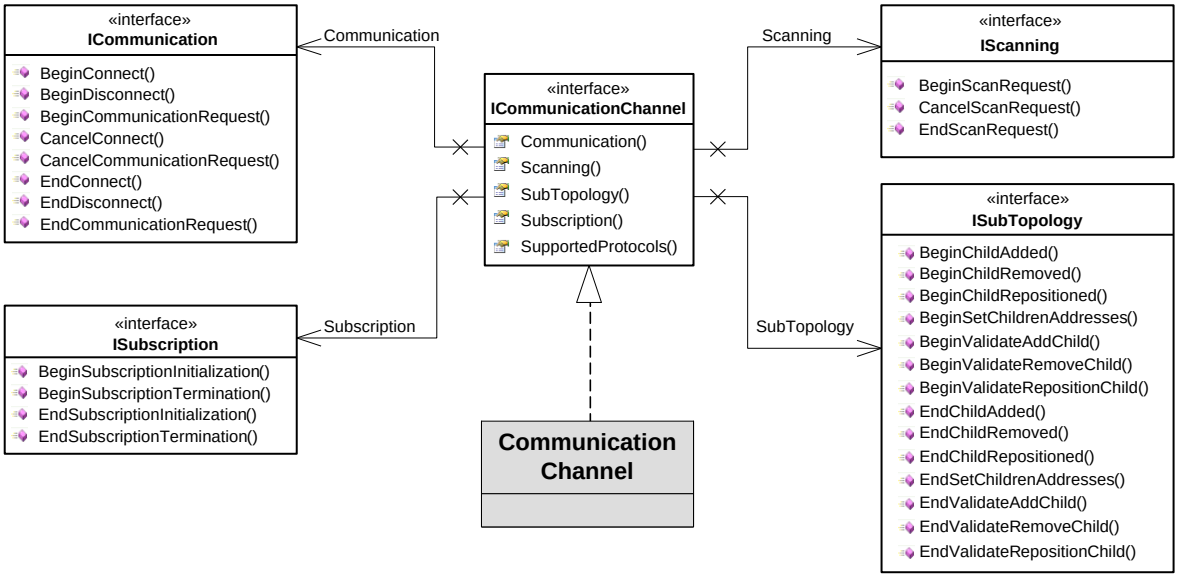


Figure 79 — Communication Channel interfaces

Table 16 provides an overview on the interfaces of a Communication Channel.

Table 16 — Communication Channel interfaces

Interface	Availability	Description
ICommunicationChannel	M	This is the main interface of a Communication Channel. It provides access to all other channel interfaces and to channel related information (e.g. supported protocols).
ICommunication	M	This interface is the communication entry point of a channel.
IScanning	C	This interface is used to request a scan of the sub-topology of a Communication Channel. This interface shall be provided for communication protocols that support scanning. The corresponding FDT Protocol Annex shall define whether this interface is mandatory or not.
ISubscription	C	This interface extends the communication entry point of a channel with device initiated data transfer functionality. This interface should be provided for communication protocols that allow for device initiated data transfer. The corresponding FDT Protocol Annex shall define whether this interface is mandatory or not.
ISubTopology	M	This interface provides methods for management of the sub-topology for a Communication Channel.

As defined in 4.6 a communication channel shall be able to manage multiple connections to one or multiple devices. The state of each connection shall be managed separately. The state machine shown in Figure 80 and Table 17 describes the states related to a connection. Since a connection is not an active object, the states are expected to be managed by the respective communication channel.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

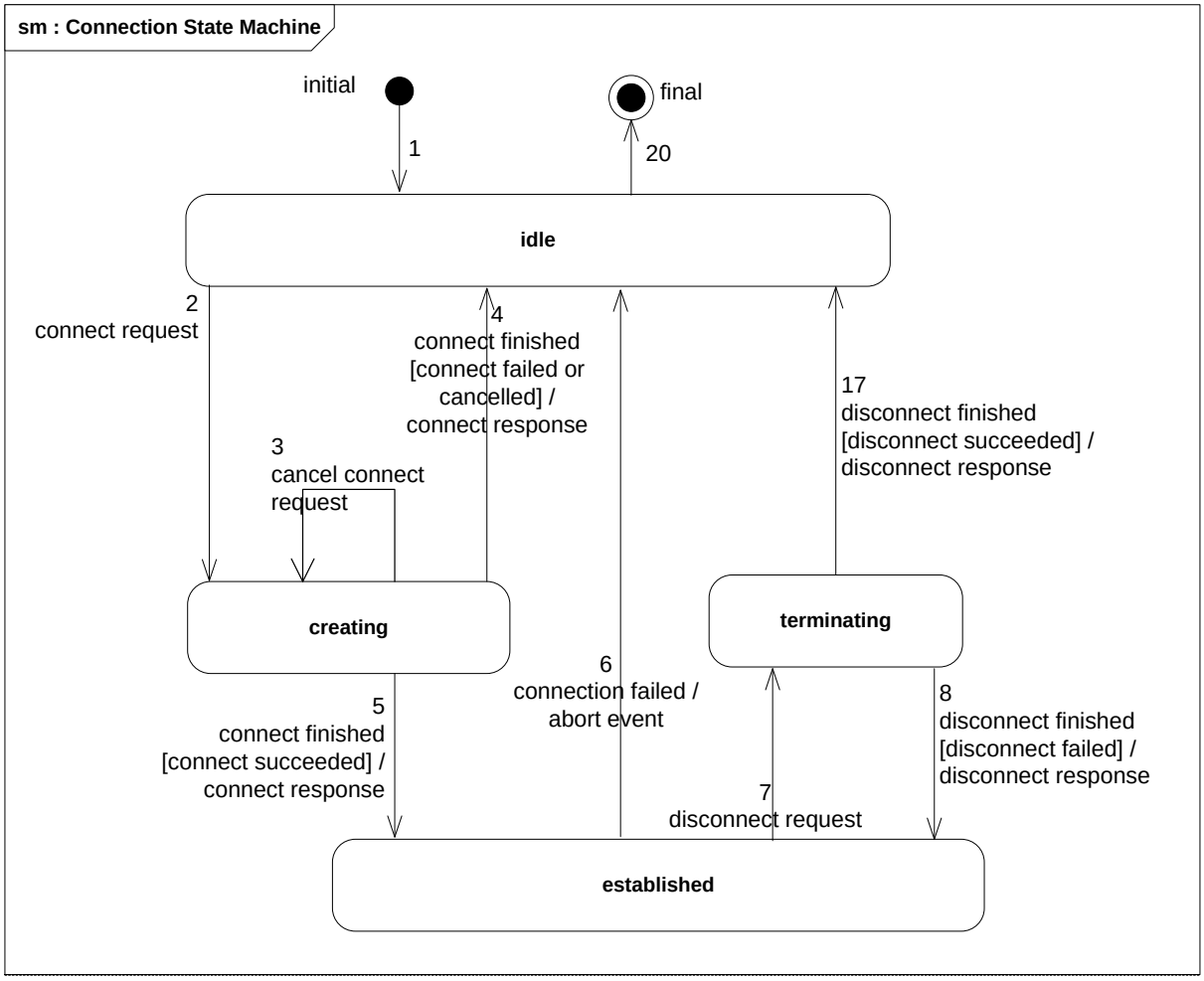


Figure 80 — Connection state machine

Table 17 — Definition of connection state machine

#	Start state	End state	Trigger	Condition	Action
1	initial	idle	{automatic}		
2	idle	creating	connect request has been received		
3	creating	creating	cancel request has been received		start canceling the connection
4	creating	idle	connect has finished	connect failed or has been canceled	send connect response (negative)
5	creating	established	connect has finished	connect succeeded	send connect response (positive)
6	established	idle	connection failed		send abort event
7	established	terminating	disconnect request		start terminating the connection
8	terminating	established	disconnect has finished	disconnect failed	send disconnect response (negative)
9	terminating	idle	disconnect has finished	disconnect successful	send disconnect response (positive)

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

6.6 Availability of interface methods

Frame Application interfaces can always be called from other FDT objects as soon as the Frame Application provides access to these interfaces.

The availability of interface methods of the DTM-related objects may depend on the state of the DTM instance.

The table below defines the interfaces of a DTM BL which can be used by a Frame Application at the shown states, see Table 18.

Table 18 — Availability of DTM BL methods in different states

Interface / Method	created	initialized	configuring	running	communicationAllowed	releasing	released
IChannels +)			X	X	X		
ICommandFunction				X	X		
IComparison.<InstanceDataCompare()>				X	X		
IComparison.<DeviceDataCompare()>					X		
IDeviceData							
<GetDataInfo()>				X	X		
(all other methods)					X		
IDtm3 +)							
Init()	X						
BeginRelease()	X	X	X	X			
LoadData()		X					
InitData()		X					
<Configuration()>			X				
Run()			X				
EnableCommunication2()				X			
<StopCommunication()>					X		
DtmSystemGuiLabel, DtmSystemTag, FdtVersion, TraceLevel		X	X	X	X	X	
All other methods / interface properties				X	X	X	

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Interface / Method		created	initialized	configuring	running	communicationAllowed	releasing	released
	EnableCommunication2()				X			
	DtmUiManagement				X	X		
IDtmInformation +)			X	X	X	X		
IDtmMessaging					X	X		
IDtmUiMessaging					X	X		
IFunction					X	X		
IHardwareInformation						X		
IInstanceData					X	X		
INetworkData					X	X		
INetworkInfoValidation					X	X		
IOnlineOperation						X		
IPorts					X	X		
IProcessData					X	X		
IProcessImage					X	X		
IReporting					X	X		
IDeviceCustomConfiguration IInstanceCustomConfiguration					X			
+) The Frame Application shall not subscribe to events before the DTM is in state 'running'								

Communication Channel interfaces can always be called from other FDT objects as soon as the Communication Channel provides access to these interfaces.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

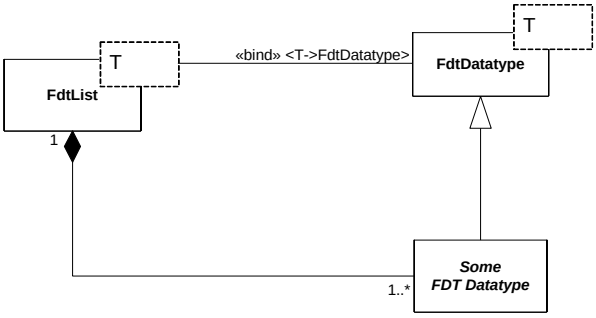
7 FDT datatypes

7.1 General

Datatypes are defined in Annex B. This section provides an overview on top-level datatypes and how they are used. This section (diagrams and tables) does not provide the complete datatype definition, please refer to Annex B for a complete datatype definition.

7.2 Datatypes – Base

FDT defines two basic datatypes: FdtDatatype and FdtList<>. Figure 81 shows examples how FdtDatatype and FdtList<> are used.



Used in:

-/-

Figure 81 — FdtDatatype and FdtList

Table 19 describes the base datatypes.

Table 19 — FDT base datatypes

Datatype	Description
FdtDatatype	Base class for all FDT datatype classes. The class provides the base implementation for the Verify() and Clone() methods. The type parameter T is always set to the derived class and is used to control these methods: • Verify() checks whether all properties are valid (e.g. mandatory properties have a value etc.). • Clone() creates a new object that is a deep-copy of the instance. All objects are duplicated, the top-level objects are duplicated as well as all the lower levels.
FdtList<>	Generic list of FdtDatatypes. The type parameter T defines the type of the list elements, FdtList<> is derived from System.Collections.Generic.List and provides all functions of List. The Verify() method enforces the rule that the list shall not be empty. If an empty list shall be represented, the respective member shall return 'null'. Like FdtDatatype the FdtList<> also provides the methods Verify() and Clone().

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

7.3 General datatypes

General FDT datatypes are used in various other FDT datatypes.

Table 20 lists and describes the general FDT datatypes

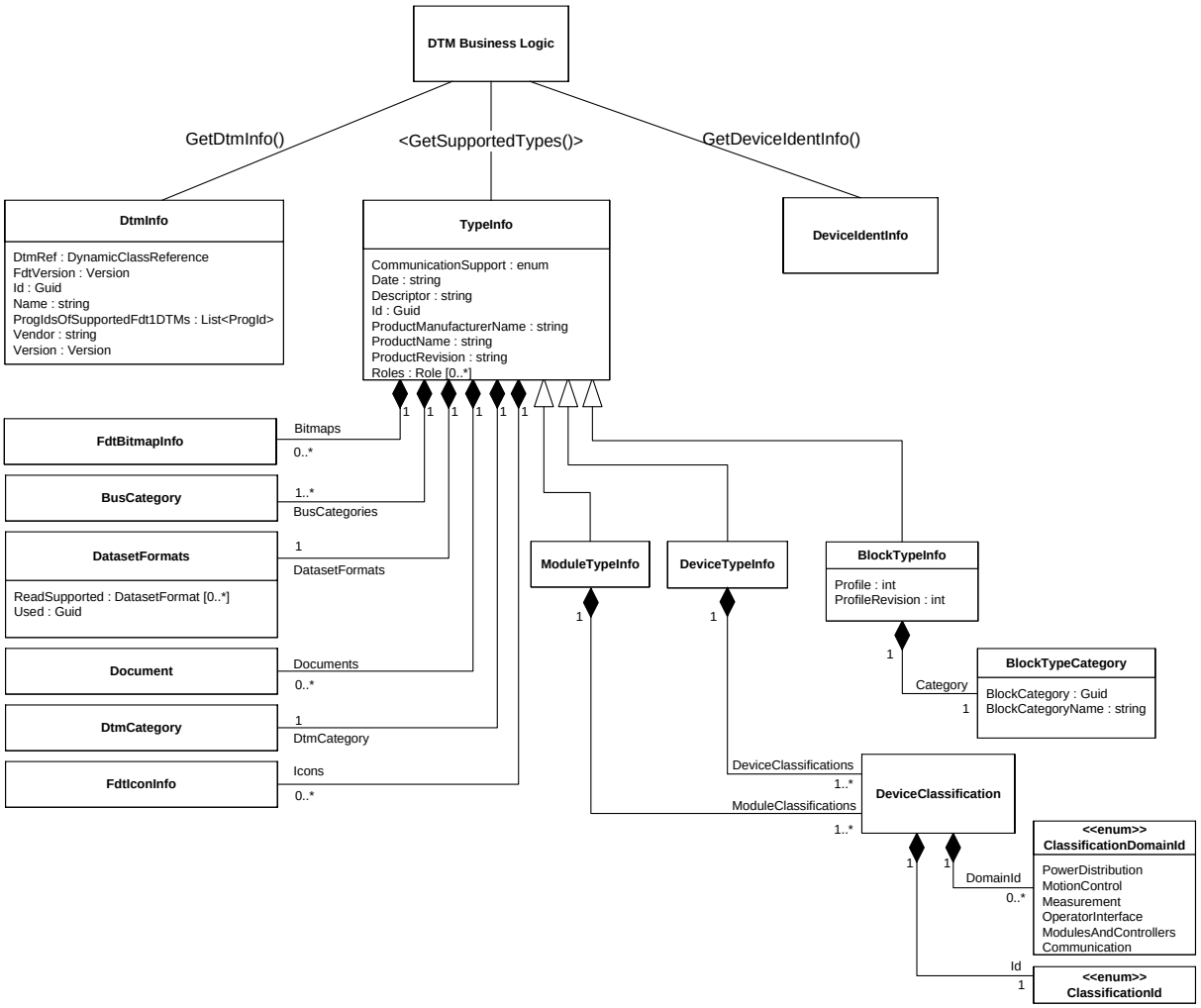
Table 20 — FDT General datatypes

Datatype name	Description
BusCategory	A bus category is a Universally Unique Identifier for a fieldbus protocol (or a point-to-point communication protocol). A property indicates whether the bus category is 'supported' or 'required'.
ChannelReference	Unique identifier of a Communication Channel provided by a DTM.
InvokeId	Unique identifier for an opened user interface.
PhysicalLayer	Unique identifier for a physical layer of a fieldbus like PROFIBUS PA.
PortReference	Unique identifier of a Port provided by a DTM.
ProgressInfo	Information about progress of an operation.
ProgressInfo<T>	Intermediated result and information about progress of an operation.
ProtocolInfoAttribute	This attribute class exposes general information about a protocol-specific assembly.
SemanticInfo	This class provides semantic information for a data object. For a range of predefined SemanticIds see Annex K.
UserInfo	Description of the user level including information about permissions, current session etc.

7.4 Datatypes – DtmInfo / TypeInfo

The following class diagram (Figure 82) describes the relations of DtmInfo, TypeInfo and associated classes.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:

IDtmInformation.GetDtmInfo()
IDtmInformation.BeginGetSupportedTypes() / IDtmInformation.EndGetSupportedTypes()
IDtmInformation.GetDeviceIdentInfo()
IDtm3.ActiveType

Figure 82 — DtmInfo / TypeInfo – datatypes

Table 21 describes datatypes related to DtmInfo.

Table 21 — DtmInfo datatype description

Datatype	Description
BlockTypeCategory	A block type category is a Universally Unique Identifier for a block category (e.g. Analog Input, Digital Output).
BlockTypeInfo	The representation for a particular block type within the DTM is called DTM Block Type. A DTM may contain one or more DTM Block Types. The concrete design and implementation of the DTM Block Types is not in scope of FDT. This class provides only information about these pieces of software like name, version, vendor, supported protocols etc.
BusCategory	A bus category is a Unique Identifier for a fieldbus protocol (or a point-to-point communication). See also 7.8.
ClassificationId	Unique identifier according to its primary measurement (IEC 62390 Annex G).
ClassificationDomainId	Device classification domain groups (IEC 62390 Annex G).
DatasetFormats	Dataset format identifiers of persisted data, used and supported by a DTM
DeviceClassification	Classification of a device according IEC 62390, Annex G

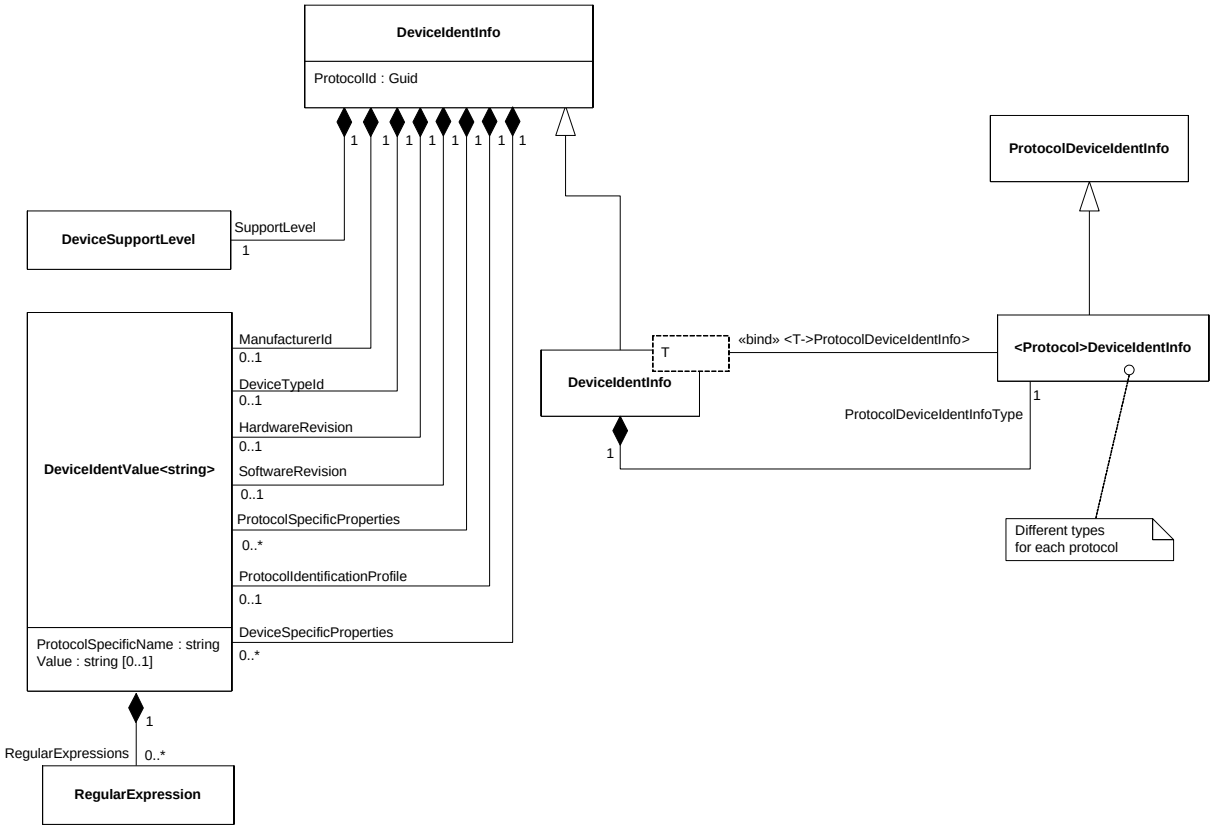
Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Datatype	Description
DeviceIdentInfo	This class is used to describe physical device types which are supported by a Type. It contains identification elements of a physical device type or device type group.
DeviceTypeInfo	The representation for a particular physical device type within the DTM is called DTM Device Type. A DTM may contain one or more DTM Device Types. The concrete design and implementation of the DTM Device Types is not in scope of FDT. This class provides only information about these pieces of software like name, version, vendor, supported protocols etc.
Document	Information about documents on hard disk or in the Web. This could be any device manual, help file, spare part list etc. which is installed together with the DTM or available on the Web. A Document may also provide protocol-specific information for a DeviceType (e.g. EDS). In such cases the document shall be categorized as 'Technical Document' and be marked with an appropriate protocol-specific SemanticId.
DtmInfo	DtmInfo contains general information about a DTM such as name, version, identifier and vendor of the software, the FDT version to which the DTM complies.
FdtBitmapInfo	Description of a bitmap for representation of a device, module or block in BMP format (high resolution, 24 bit color and 8 bit transparency info (alpha channel))
FdtIconInfo	Information about device, module or block icon.
ModuleTypeInfo	The representation for a particular physical module type within the DTM is called DTM Module Type. A DTM may contain one or more DTM Module Types. The concrete design and implementation of the DTM Module Types is not in scope of FDT. This class provides information about DTM Module Types like name, version, vendor, supported protocols etc.
TypeInfo	Abstract base class used for definition of device type, block type or module type. A DTM shall contain one or more TypeInfo objects.

7.5 Datatypes – DeviceIdentInfo

The following class diagram (Figure 83) describes the relations of the DeviceIdentInfo class. DeviceIdentInfo can be requested from IDtmInformation.GetDeviceIdentInfo() for a given TypeIdent and BusCategory.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:
IDtmInformation.GetDeviceIdentInfo()

Figure 83 — DeviceIdentInfo – datatypes

Table 22 describes datatypes related to DeviceIdentInfo.

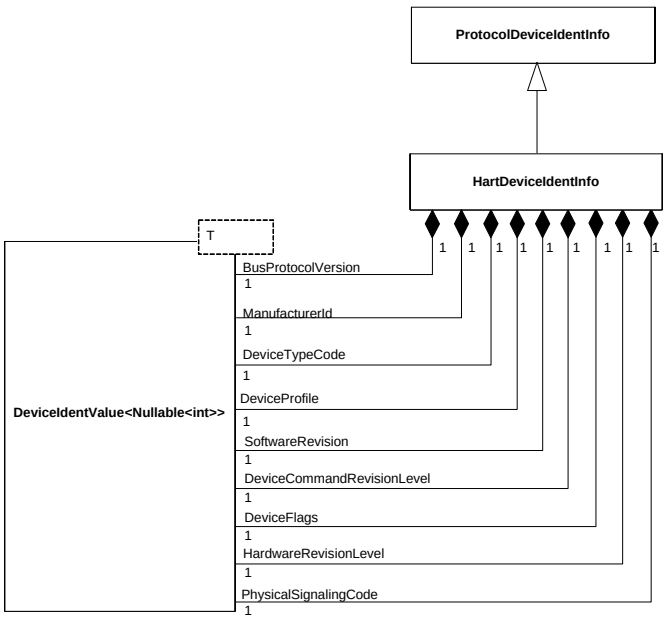
Table 22 — DeviceIdentInfo datatype description

Datatype	Description
DeviceIdentInfo	This class is used to describe physical device types which are supported by a DTM Device Type. It contains identification elements of a physical device type or device type group. Remark: This class provides a protocol neutral access to the information , therefore it typically will be used by the Frame Application if no protocol-specific handling is needed.
DeviceIdentInfo<T>	The derived class DeviceIdentInfo<T> provides a protocol-specific access to the information which is more type-safe. This class should be used whenever possible. The generic type parameter T defines the type of the protocol-specific class which defines the protocol-specific identification properties. These protocol-specific properties are mapped to the properties: • ManufacturerId • DeviceTypeId • SoftwareRevision • HardwareRevision • ProtocolIdentificationProfile • ProtocolSpecificProperties which are defined in the base class DeviceIdentInfo.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Datatype	Description
DeviceIdentValue<T>	Represents a single identification element of a physical device type or group. For example: Device Type Id, Manufacturer Id etc.
DeviceSupportLevel	Enumeration which defines the support level of a DTM Device Type for a physical device.
ProtocolDeviceIdentInfo	Abstract base class for protocol-specific device identification properties. Protocol-specific classes derive from this class and define the specific device identification properties. However, these protocol-specific properties can be accessed in a protocol neutral way by accessing the corresponding properties in the DeviceIdentInfo class.
ProtocolId	Universally Unique Identifier for a fieldbus protocol (or a point-to-point communication).
ProtocolIdentificationProfile	Defines the protocol-specific identification profile which is used for device identification. (examples for PROFIBUS: I&M, PA, DP). If a protocol does not support multiple identification profiles then this property shall be empty.
RegularExpression	Regular expression that defines which physical device types are supported by a DTM Device Type.

The class diagram in Figure 84 shows the protocol-specific datatype DeviceIdentInfo<T> for the example HART protocol.



Used in:

Protocol-specific Device DTM providing values for HART-specific DeviceIdentInfo

Figure 84 — DeviceIdentInfo – Example for HART

Table 23 describes HART datatypes related to DeviceIdentInfo.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 23 — DeviceIdentInfo – Example for HART

Datatype	Description
DeviceIdentValue<T>	Represents an identification element of a device type for a physical device type or group. For example: Device Type Id, Manufacturer Id etc. The generic type parameter T defines the type of the identification value (e.g. int, float, string etc.) corresponding to the format defined in the protocol. The identification element can either be a specific value or a regular expression (e.g. defining a range of supported identification values).
HartDeviceIdentInfo	HART-specific device identification information.
ProtocolDeviceIdentInfo	Abstract base class for protocol-specific device identification properties. Protocol-specific classes derive from this class and define the specific device identification properties. However, these protocol-specific properties can be accessed in a protocol neutral way by accessing corresponding properties in the DeviceIdentInfo class.

The example in Figure 85 demonstrates how a (HART) Device DTM creates and returns a DeviceIdentInfo instance:

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
public DeviceIdentInfo GetDeviceIdentInfo()
{
    // Create the HART-specific identification properties first

    // Manufacturer code of the device vendor is 17
    HartDeviceIdentInfo hartSpecificInfo = new HartDeviceIdentInfo();
    hartSpecificInfo.ManufacturerIdentificationCode =
                                                    new DeviceIdentValue<int?>(17);

    // The ID of the supported device is 123
    var identVal = new DeviceIdentValue<int?>();
    hartSpecificInfo.DeviceTypeCode = new DeviceIdentValue<int?>(123);

    // Device is a HART 5 Device
    hartSpecificInfo.HartMajorRevisionNumber = new DeviceIdentValue<int?>(5);

    // This DTM is able to handle the software versions 1,2 and 3 of the device
    identVal = new DeviceIdentValue<int?>();
    identVal.RegularExpressions = new FdtList<RegularExpression>(
                                            new RegularExpression("1|2|3"));
    hartSpecificInfo.SoftwareRevision = identVal;

    // This DTM is able to handle only the command revision level 5 of the device
    hartSpecificInfo.DeviceRevisionLevel = new DeviceIdentValue<int?>(5);

    // This DTM is able to handle all hardware versions of this device
    identVal = new DeviceIdentValue<int?>();
    identVal.RegularExpressions = new FdtList<RegularExpression>(
                                            new RegularExpression(".*"));
    hartSpecificInfo.HardwareRevisionLevel = identVal;

    // Physical Signaling Code is not relevant for identification
    identVal = new DeviceIdentValue<int?>();
    identVal.RegularExpressions = new FdtList<RegularExpression>(
                                            new RegularExpression(".*"));
    hartSpecificInfo.PhysicalSignalingCode = identVal;

    // Device Flags are not relevant for identification
    identVal = new DeviceIdentValue<int?>();
    identVal.RegularExpressions = new FdtList<RegularExpression>(
                                            new RegularExpression(".*"));
    hartSpecificInfo.DeviceFlags = identVal;

    // Device ident information (protocol neutral)
    DeviceIdentInfo<HartDeviceIdentInfo> deviceIdentInfo =
                                new DeviceIdentInfo<HartDeviceIdentInfo>();

    // This DTM is designed to support a specific device
    deviceIdentInfo.SupportLevel = DeviceSupportLevel.SpecificSupport;

    // Set the protocol-specific info
    deviceIdentInfo.ProtocolSpecificIdentInfo = hartSpecificInfo;

    return deviceIdentInfo;
}
```

Figure 85 — Example: DeviceIdentInfo creation

The example in Figure 86 demonstrates how a Frame Application requests and uses the DeviceIdentInfo instance created in Figure 85. The protocol-specific properties shown in Figure 85 are mapped automatically to the protocol-independent properties which are used in Figure 86.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
public void ShowDeviceIdentInfo(IDtmInformation dtm, DeviceTypeInfo deviceTypeInfo)
{
   FdtList<DeviceIdentInfo> deviceIdentInfo = dtm.GetDeviceIdentInfo(deviceTypeInfo.Id,
                                   deviceTypeInfo.BusCategories[0]);
   // Standard FDT3 ident properties
   MessageBox.Show("Manufacturer ID = " + deviceIdentInfo[0].ManufacturerId.Value + "\n" +
                   "Device Type ID = " + deviceIdentInfo[0].DeviceTypeId.Value + "\n" +
                   "Software Rev. = " + deviceIdentInfo[0].SoftwareRevision.Value + "\n" +
                   "Hardware Rev. = " + deviceIdentInfo[0].HardwareRevision.Value + "\n");

    // Ident properties only defined in the protocol
    // (for HART: DeviceCommandRevisionLevel and Device Flag)
    foreach (DeviceIdentValue<string> identValue
        in deviceIdentInfo[0].ProtocolSpecificProperties)
    {
        MessageBox.Show(identValue.ProtocolSpecificName + " = " + identValue.Value);
    }
}
```

Figure 86 — Example: Using DeviceIdentInfo

The example in Figure 87 demonstrates how the HART-specific datatype assembly (Fdt.Datatypes.Hart.dll) exposes the type information over the DeviceIdentInfoTypeAttribute:

```
[assembly: DeviceIdentInfoType
(
DeviceIdentInfoType = typeof(DeviceIdentInfo<HartDeviceIdentInfo>),
ProtocolDeviceIdentInfoType = typeof(HartDeviceIdentInfo),
       DeviceScanInfoType = typeof(DeviceScanInfo<HartDeviceScanInfo>),
       ProtocolDeviceScanInfoType = typeof(HartDeviceScanInfo)
)
]
```

Figure 87 — Example: DeviceIdentInfoTypeAttribute

7.6 Datatypes for installation and deployment

7.6.1 Datatypes – DtmPackageManifest

A DtmPackageManifest describes the installation of a DTM. It is used for installation and deployment. (see 9.6). Figure 88 shows a class diagram with related classes of DtmPackageManifest.

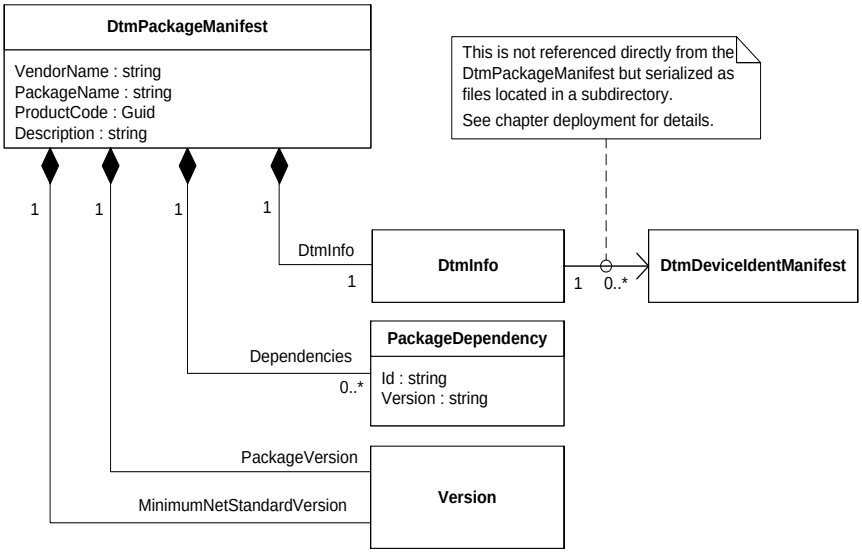


Figure 88 — DtmPackageManifest – datatypes

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 24 describes DtmPackageManifest class and its related classes.

Table 24 — DtmPackageManifest datatype description

Datatype	Description
DtmDeviceIdentManifest	A DtmDeviceIdentManifest describes additional physical device parameters that are required for device identification.
DtmInfo	DtmInfo contains general information about a DTM such as name, version, identifier and vendor of the software, the FDT version to which the DTM complies.
DtmPackageManifest	A package manifest describes the installation of a DTM, including identification of the product, the vendor, version and included DTMs.
PackageDependency	Provides the information about a dependency according to NuGet version 2.0. This member can be used to indicate dependency to NuGet packages (FDT library or protocol libraries).
Version	This class represents a version (e.g. of the package or the required minimal version of .NET Standard library).

7.6.2 Datatypes – DtmManifest

A DtmManifest describes the components of a DTM (see 9.5.3). Figure 89 shows a class diagram with related classes of DtmManifest.

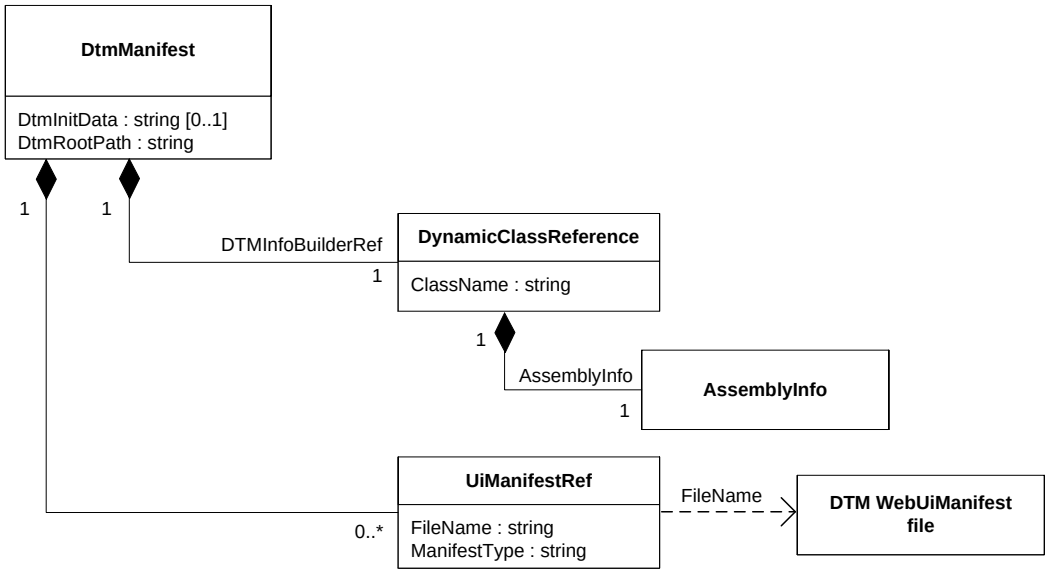


Figure 89 — DtmManifest – datatypes

Table 25 describes DtmManifest datatype and its related classes.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 25 — DtmManifest datatype description

Datatype	Description
AssemblyInfo	Information about a .NET assembly.
DtmManifest	A DTM manifest describes the assembly of a DTM and the included DTM itself. The manifest is used to register an installed DTM in order to enable Frame Applications to find it.
DynamicClassReference	Information about a class e.g. a DtmInfoBuilder or a DTM.
DTM WebUI manifest file	A DTM WebUI manifest file is used to register a DTM WebUI in the system in order to enable Frame Applications to find it. The file contains a DtmWebUiManifest (see 7.6.3).
UiManifestRef	Reference to a DTM WebUI manifest file.

7.6.3 Datatypes – DtmWebUiManifest

The DtmWebUiManifest describes a DTM WebUI container file and the included DTM User Interface function. Manifests are used to register installed DTM WebUI functions in order to enable the Frame Applications to find and execute them.

Figure 90 shows a class diagram with related classes of DtmWebUiManifest.

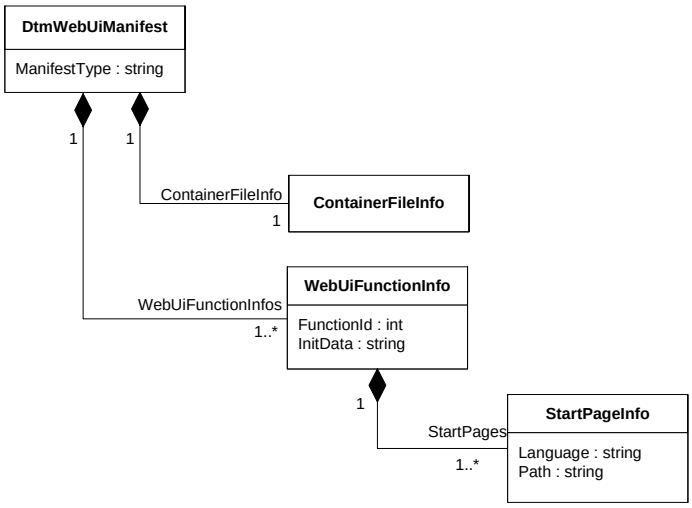


Figure 90 — DtmWebUiManifest – datatypes

Table 26 describes DtmWebUiManifest classes and the related classes.

Table 26 — DtmWebUiManifest datatype description

Datatype	Description
ContainerFileInfo	Information about a WebUI container file.
DtmWebUiManifest	This manifest describes a DTM WebUI function. The manifest is used to register installed DTM WebUI functions in order to enable the Frame Applications to find and execute them.
StartPageInfo	Provides the path and the supported language of a DTM WebUI start page.
WebUiFunctionInfo	Description of a DTM WebUI. Frame Applications shall use this information to find the DTM WebUI for a specific function.

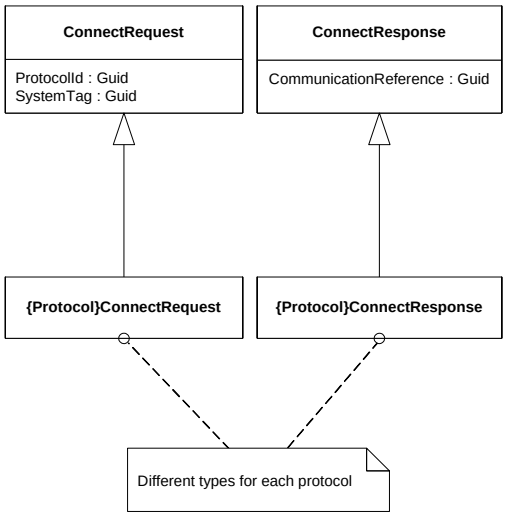
Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

7.7 Datatypes – Communication

The communication datatypes are used to exchange data between a DTM and its parent Communication Channel in order to:

- Establish a connection to the device
- Perform data exchange transactions with the device
- Release the connection
- Subscribe device initiated data transfer between a DTM and its parent Communication Channel
- Request scanning of bus topology
- Request address setting of Child DTM

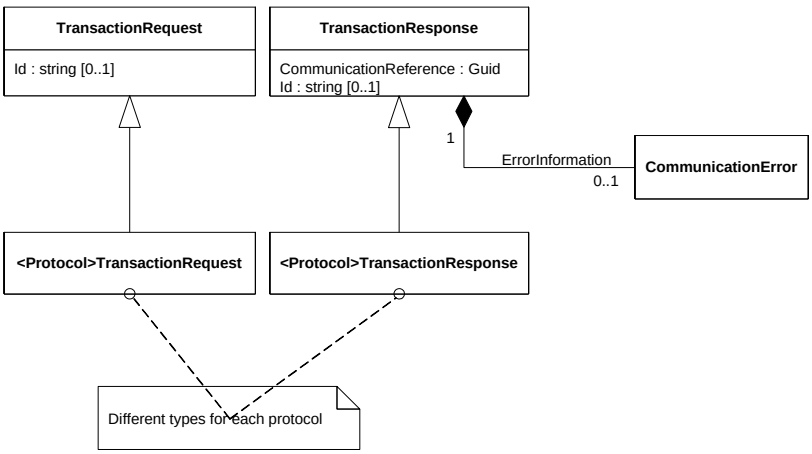
Figure 91 shows a class diagram with datatypes used to establish a connection to the device.



Used in:
ConnectRequest: ICommunication.BeginConnect()
ConnectResponse is returned in ICommunication.EndConnect()

Figure 91 — Communication datatypes – Connect

Figure 92 shows a class diagram with datatypes used to exchange data with the device.

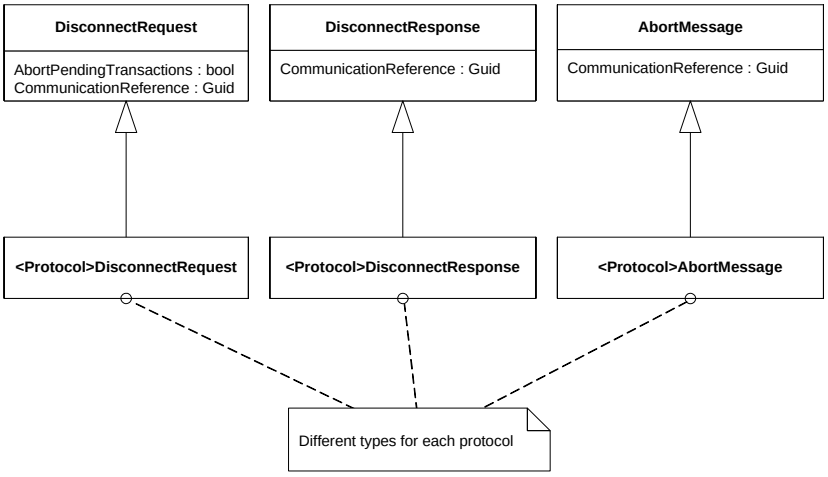


Used in:
TransactionRequest: ICommunication.BeginCommunicationRequest()
TransactionResponse is returned in ICommunication.EndCommunicationRequest()

Figure 92 — Communication datatypes – Transaction

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Figure 93 shows a class diagram with datatypes used to release a connection to the device.

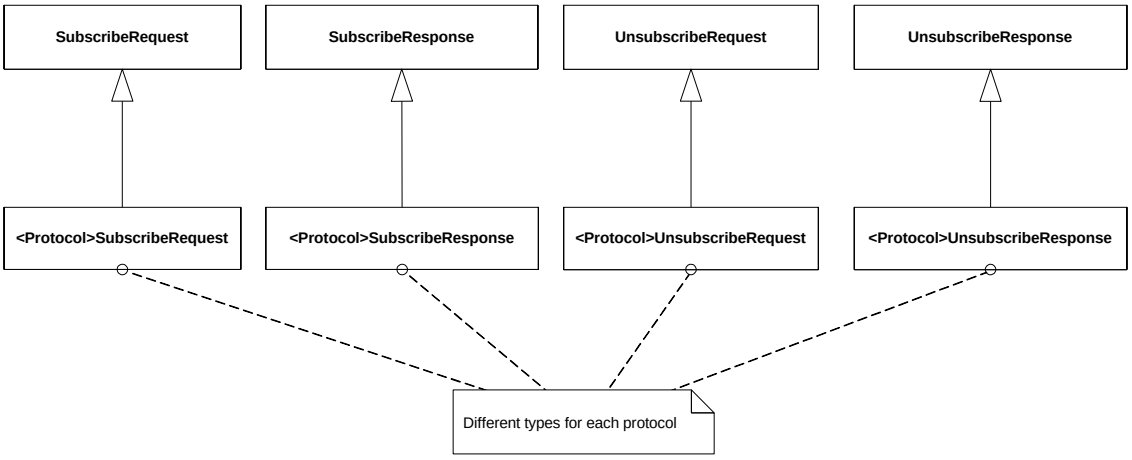


Used in:

DisconnectRequest: ICommunication.BeginDisconnect()
DisconnectResponse is returned in ICommunication.EndDisconnect()
AbortMessage: AbortCallback()

Figure 93 — Communication datatypes – Disconnect

Figure 94 shows a class diagram with datatypes used to subscribe and unsubscribe device initiated data transfer.



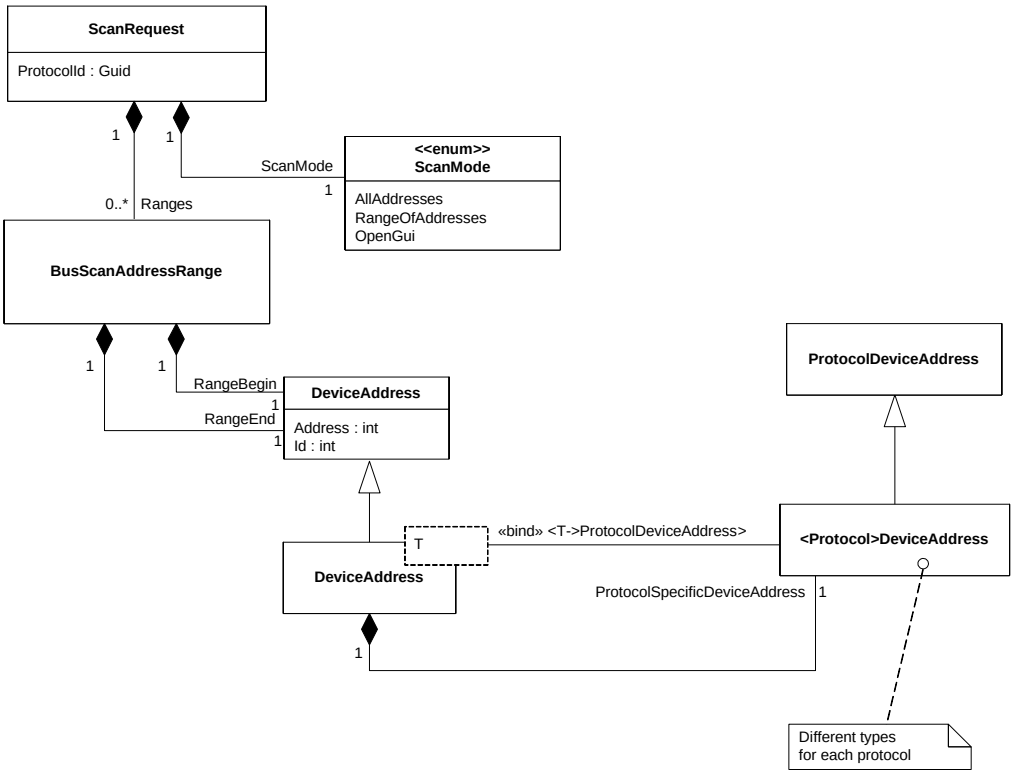
Used in:

SubscribeRequest: ISubscription.BeginSubscriptionInitialization()
SubscribeResponse is returned in ISubscription.EndSubscriptionInitialization()
UnsubscribeRequest: ISubscription.BeginSubscriptionTermination()
UnsubscribeResponse is returned in ISubscription.EndSubscriptionTermination()

Figure 94 — Communication datatypes – Subscribe

Figure 95 shows a class diagram with datatypes used to request scanning of the sub-topology of a Communication Channel.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

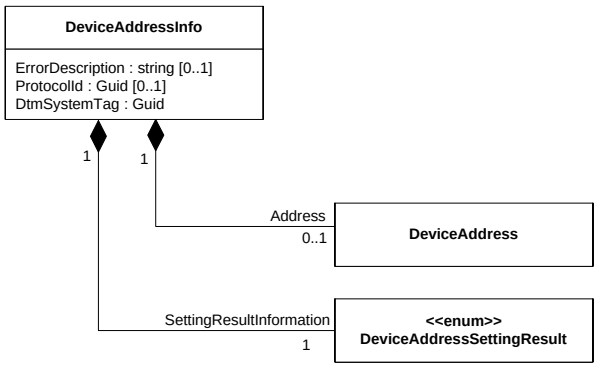


Used in:

ScanRequest: IScanning.BeginScanRequest()

Figure 95 — Communication datatypes – Scanning

Figure 96 shows a class diagram with datatypes used to request setting of device address of Child DTM of a Communication Channel.



Used in:

DeviceAddressInfo: ISubTopology.BeginSetChildrenAddresses()
DeviceAddressInfo Is returned from ISubTopology.EndSetChildrenAddresses()

Figure 96 — Communication datatypes – Address setting

Table 27 describes the communication datatypes.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 27 — Communication datatype description

Datatype	Description
ConnectRequest	Fieldbus protocol independent base class for information needed to establish a communication link.
ConnectResponse	Fieldbus protocol independent base class for response information about an established communication link.
TransactionRequest	Fieldbus protocol independent base class for transaction request information.
TransactionResponse	Fieldbus protocol independent base class for transaction results.
DisconnectRequest	Fieldbus protocol independent base class for disconnection information.
DisconnectResponse	Fieldbus protocol independent base class for results of disconnect operation.
AbortMessage	Information to specify an abort of a communication link.
SubscribeRequest	Fieldbus protocol independent base class with information for initialization of device initiated data transfer.
SubscribeResponse	Fieldbus protocol independent base class for information about communication data subscription.
UnsubscribeRequest	Fieldbus protocol independent base class for termination of subscription of device initiated data transfer.
UnsubscribeResponse	Fieldbus protocol independent base class for response to termination of subscription of device initiated data transfer.
ScanRequest	Information for a request to scan the sub-topology of a Communication Channel
BusScanAddressRange	Information about the address range of the requested scan
DeviceAddressInfo	Address information which is used to request the Communication Channel to set the address of its Child DTMs.
DeviceAddress	Address of the device in the network or fieldbus.

The following example (Figure 97) demonstrates how a (HART) Device DTM may connect to a device:

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
bool Connect(Guid mySystemTag, ICommunication commChannel,
              HartDeviceAddress myAddress, ref Guid communicationReference)
{
    //Create ConnectRequest
    //The required SystemTag is set by the Frame Application
    //during creation of the DTM instance
    //The Address will be set by the Communication Channel
    var request = new HartConnectRequest(mySystemTag, myAddress);

    HartConnectResponse response;

    try
    {
        //Request connection from Communication Channel
        var asyncResult =
            commChannel.BeginConnect(request, abortCallback, null, null, null);

        //Wait for finalization of the connect request
        response = commChannel.EndConnect(asyncResult) as HartConnectResponse;
    }
    catch(Exception ex)
    {
        MessageBox.Show("Connection failed\n" + "Details: " + ex.Message);
        return false;
    }

    if (response != null)
    {
        //verify response
        try
        {
            response.Verify();
        }
        catch(Exception ex)
        {
            MessageBox.Show("Connection failed\n" + "Details: " + ex.Message);
            return false;
        }

        //Connection established
        //the response contains the complete address information
        //and the communication reference of the connection
        communicationReference = response.CommunicationReference;
        MessageBox.Show("Successfully connected with device\n" +
            "Short Address: " + response.Address.ShortAddress + "\n" +
            "Short TAG: " + response.Address.ShortTag + "\n" +
            "Long TAG: " + response.Address.LongTag + "\n");
        return true;
    }

    return false;
}
```

Figure 97 — Example: Communication – Connect for HART

The following example (Figure 98) demonstrates how the HART-specific datatype assembly(Fdt.Datatypes.Hart.dll) exposes the type information over the CommunicationType attribute:

```
[assembly: CommunicationType(
                            AbortMessageType = typeof(HartAbortMessage),
                            ConnectRequestType = typeof(HartConnectRequest),
                            ConnectResponseType = typeof(HartConnectResponse),
                            DisconnectRequestType = typeof(HartDisconnectRequest),
                            DisconnectResponseType = typeof(HartDisconnectResponse),
                       ...
                            SubscribeRequestType = typeof(HartSubscribeRequest),
                            SubscribeResponseType = typeof(HartSubscribeResponse),
                            UnsubscribeRequestType = typeof(HartUnsubscribeRequest),
                            UnsubscribeResponseType = typeof(HartUnsubscribeResponse))
]
```

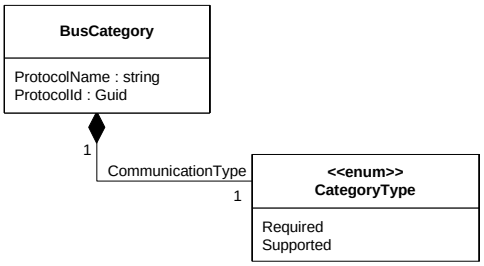
Figure 98 — Example: Communication – CommunicationType for HART

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Note: The above examples demonstrate how a protocol-specific datatype may be derived from the datatypes defined in this document and how such a protocol-specific datatype is intended to be used. For definition of the protocol-specific datatypes please refer to the respective specification document.

7.8 Datatypes – BusCategory

The following class diagram (Figure 99) describes the relations of the BusCategory class.



Used in:
TypeInfo

Figure 99 — BusCategory – datatypes

Table 28 describes the datatype BusCategory and its elements

Table 28 — BusCategory datatype description

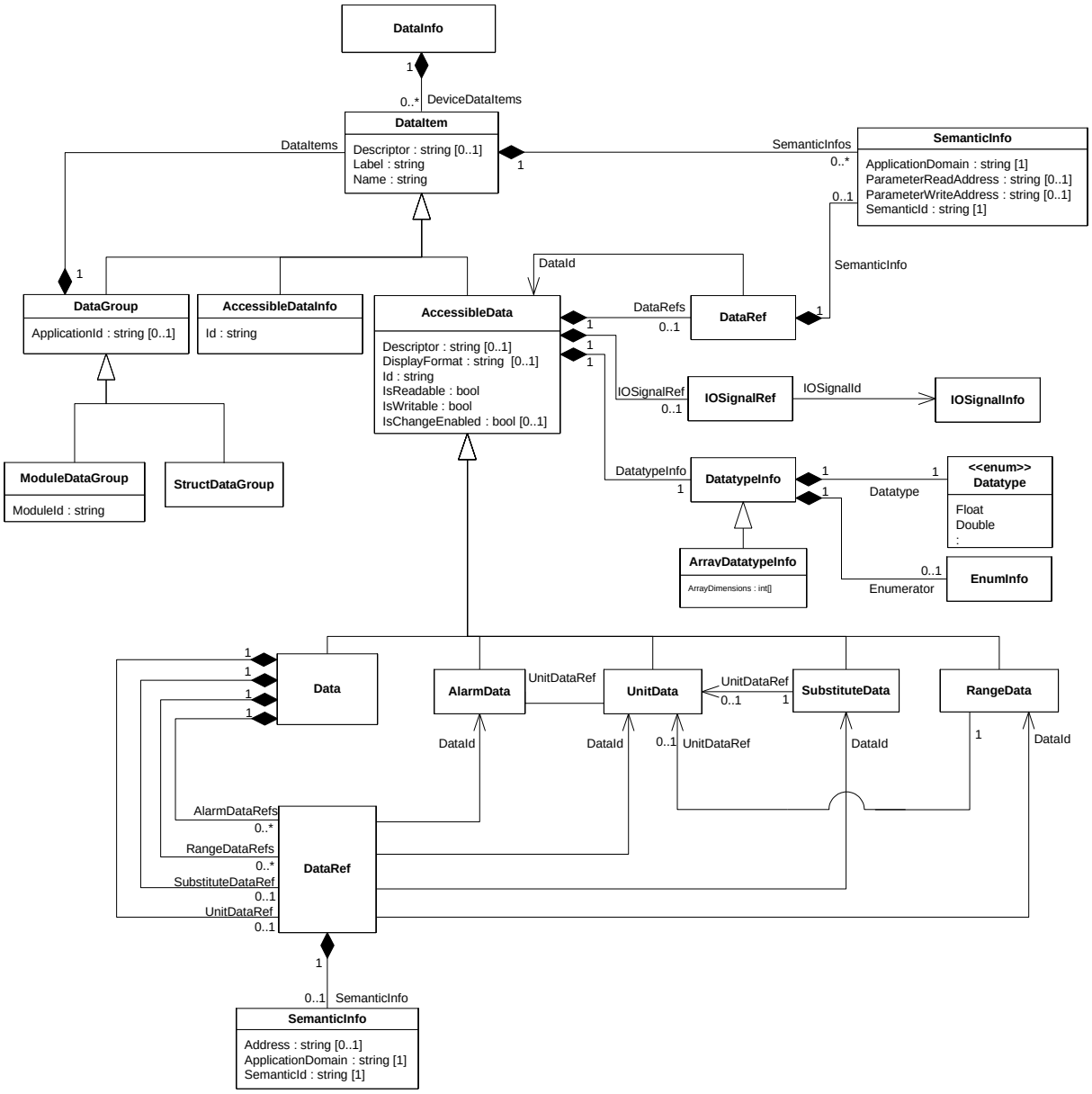
Datatype	Description
BusCategory	Bus category is a Unique Identifier for a fieldbus protocol (or a point-to-point communication).
CategoryType	Defines whether BusCategory is supported or required.

7.9 Datatypes – Device / Instance Data

7.9.1 General

The Device / Instance Data classes describe device parameters or process values that can be read from the device / instance data or written into the device / instance data. The class diagram in Figure 100 shows the classes and relations.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:

returned in IDeviceData.EndGetDataInfo() / IInstanceData.EndGetDataInfo()

Figure 100 — Device / Instance data – datatypes

Table 29 describes classes related to Device / Instance Data

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 29 — DeviceData datatype description

Datatype	Description
AccessibleData	Abstract base class for data which is readable or writable. The DTM shall provide a DisplayFormat for all AccessibleData variables with numerical values
AlarmData	Representation of an alarm parameter. An alarm shall always be of a numeric type (Float, Double, Int, Long, Uint, Ulong) or Enumerator (checked by the Verify() method).
ArrayDataTypeInfo	If the type is an array, additional information regarding the number of dimensions and size shall be provided.
Data	Describes a device parameter or a process value that can be read or written. The information contains descriptive attributes like name as well as information how the item is accessible.
DataGroup	Groups information about available device-specific parameters and process values.
DataInfo	Contains information about available device-specific parameters and process values.
DataItem	Abstract base class for device and instance data info classes.
DataRef	Reference to an item in DataInfo identified by its Id and optionally also information about the type (semantic) of the reference.
Datatype	List of possible datatypes.
DatatypeInfo	Information about type of data (see DataValue).
IOSignalInfo	Information about a single device IO signal.
IOSignalRef	Reference to an IO signal identified by its identifier.
ModuleDataGroup	Groups information about available module-specific parameters and process values.
RangeData	Representation of a range parameter. A range shall always be of a numeric type (Float, Double, Int, Long, Uint, Ulong) or DateTime (checked by the Verify() method). The RangeData may provide a reference to a UnitData. If no reference is provided, the same unit is applied as in the Data that references the RangeData.
SemanticInfo	This class provides semantic information for a data object.
StructDataGroup	Represents a data structure containing specific parameters and/or process values.
SubstituteData	Describes the value which shall be used as a fall back e.g. in case there is a disturbed communication. The SubstituteData may provide a reference to a UnitData. If no reference is provided, the same unit is applied as in the Data that references the SubstituteData.
UnitData	Representation of a unit parameter. Unit shall always be of type Enumerator (checked by the Verify() method).

Figure 101 shows how DataInfo may expose information on data of a HART device.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

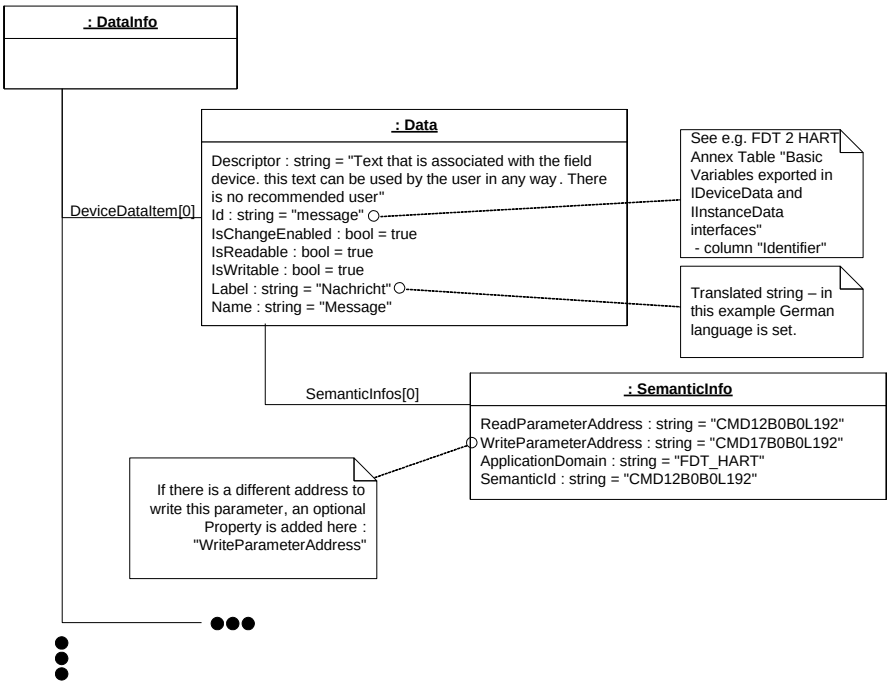
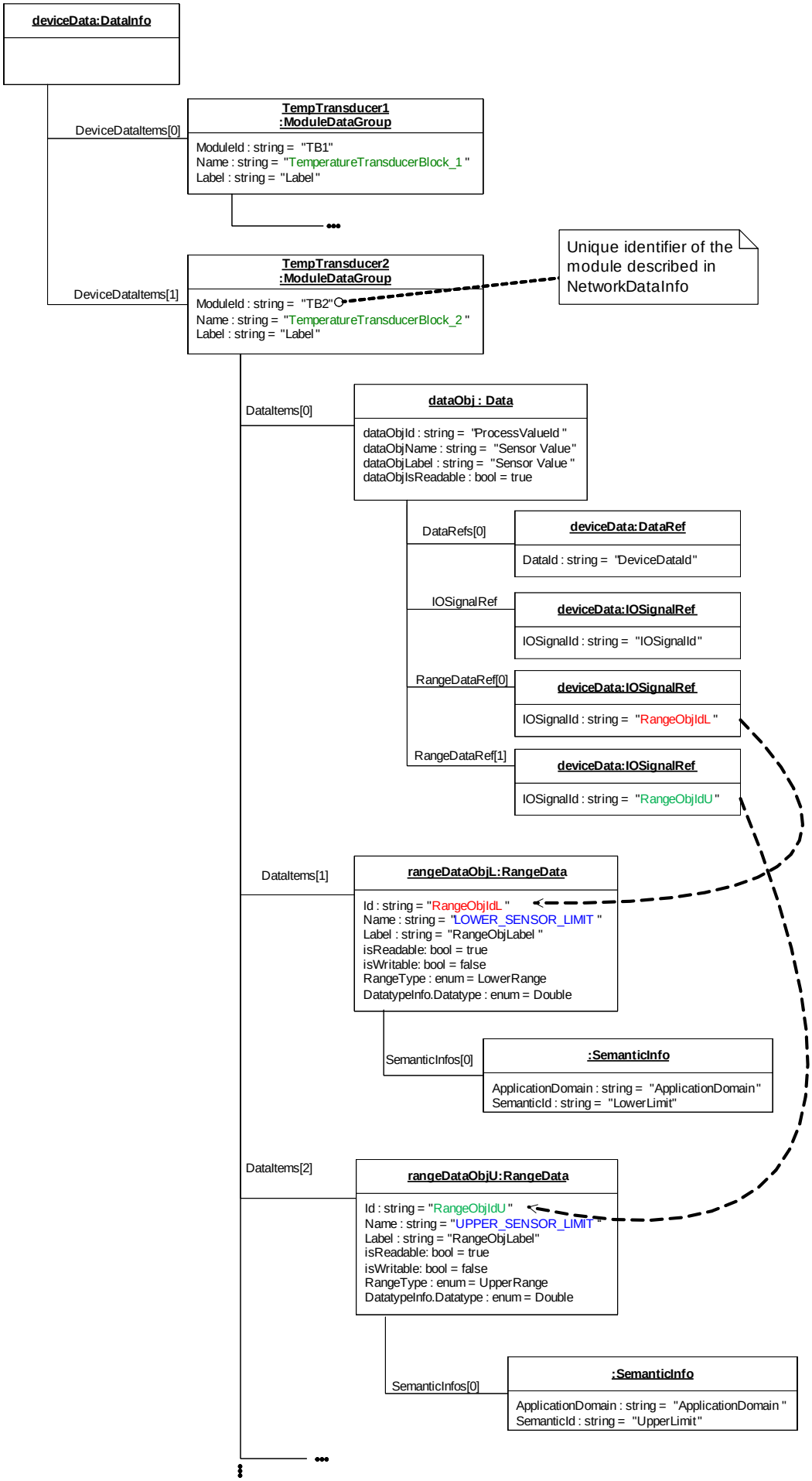


Figure 101 — Example: Providing information on data of a HART device

Figure 102 shows how DataInfo may expose information on data of a PROFIBUS device.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Figure 102 — Example: Providing information on module data of a PROFIBUS device

The following example (Figure 103) shows how to create DataInfo with one data object and a ModuleDataGroup that contains RangeData items for lower and upper limit.

```
public DataInfo GetDataInfo()
{
    DataInfo deviceData = new DataInfo();

    // Create a Data object (inherits from AccessibleData and DataItem)
    string dataObjId = "DataObjID";
    string dataObjName = "DataObjName";
    string dataObjLabel = "DataObjLabel";
    bool dataObjIsReadable = true;
    bool dataObjIsWritable = false;
    DatatypeInfo dataObjDatatypeInfo = new DatatypeInfo(Datatype.Long);
    Data dataObj = new Data(dataObjId, dataObjName, dataObjLabel,
                            dataObjIsReadable, dataObjIsWritable,
                            dataObjDatatypeInfo);
    // Define refences to other DataItems (Optional members)
    dataObj.DataRefs = new FdtList<DataRef>() { new DataRef("DeviceDataID") };
    dataObj.IOSignalRef = new Fdt.Dtm.IO.IOSignalRef("IOSignalID");

    // Create a (Lower)RangeData object
    // (inherits from AccessibleData and DataItem)
    string rangeObjIdL = "RangeObjIdL";
    string rangeObjNameL = "LOWER_SENSOR_LIMIT";
    string rangeObjLabelL = "RangeObjLabel1";
    bool rangeObjIsReadableL = true;
    bool rangeObjIsWritableL = false;
    DatatypeInfo rangeObjDatatypeInfoL = new DatatypeInfo(Datatype.Double);
    RangeType rangeTypeL = RangeType.LowerRange;

    RangeData rangeDataObjL = new RangeData(rangeObjIdL, rangeObjNameL,
                                            rangeObjLabelL, rangeObjIsReadableL,
                                            rangeObjIsWritableL,
                                            rangeObjDatatypeInfoL, rangeTypeL);
    // Define SematicInfo-object for rangeDataObj
    rangeDataObjL.SemanticInfos = new FdtList<SemanticInfo>(
                          new SemanticInfo("ApplicationDomain", "LowerLimit"));

    // Create an (Upper)RangeData object
    // (inherits from AccessibleData and DataItem)
    string rangeObjIdU = "RangeObjIdU";
    string rangeObjNameU = "UPPER_SENSOR_LIMIT";
    string rangeObjLabelU = "RangeObjLabel2";
    bool rangeObjIsReadableU = true;
    bool rangeObjIsWritableU = false;
    DatatypeInfo rangeObjDatatypeInfoU = new DatatypeInfo(Datatype.Double);
    RangeType rangeTypeU = RangeType.UpperRange;

    RangeData rangeDataObjU = new RangeData(rangeObjIdU, rangeObjNameU,
                                            rangeObjLabelU,
                                            rangeObjIsReadableU,
                                            rangeObjIsWritableU,
                                            rangeObjDatatypeInfoU, rangeTypeU);

    // Define SematicInfo-object for rangeDataObj
    rangeDataObjU.SemanticInfos = new FdtList<SemanticInfo>(
                  new SemanticInfo("ApplicationDomain", "UpperLimit"));

    //Create a ModuleDataGroup with the two RangeData-items.
    FdtList<DataItem> dataItemsInGroup = new FdtList<DataItem>() {rangeDataObjL,
                                                     rangeDataObjU};

    ModuleDataGroup rangeDataGroupObj = new ModuleDataGroup("TB1",
                                                            "TemperatureTransducerBlock_1",
                                                            "Label",
                                                            dataItemsInGroup);
    // Put DataItem objects into the list
    deviceData.DeviceDataItems = new FdtList<DataItem>() { dataObj,
                                            rangeDataGroupObj };

    return deviceData;
}
```

Figure 103 — Example: Providing information on data

If the data is structured data, then the StructDataGroup may be used to show the structure of the data (see Figure 104).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

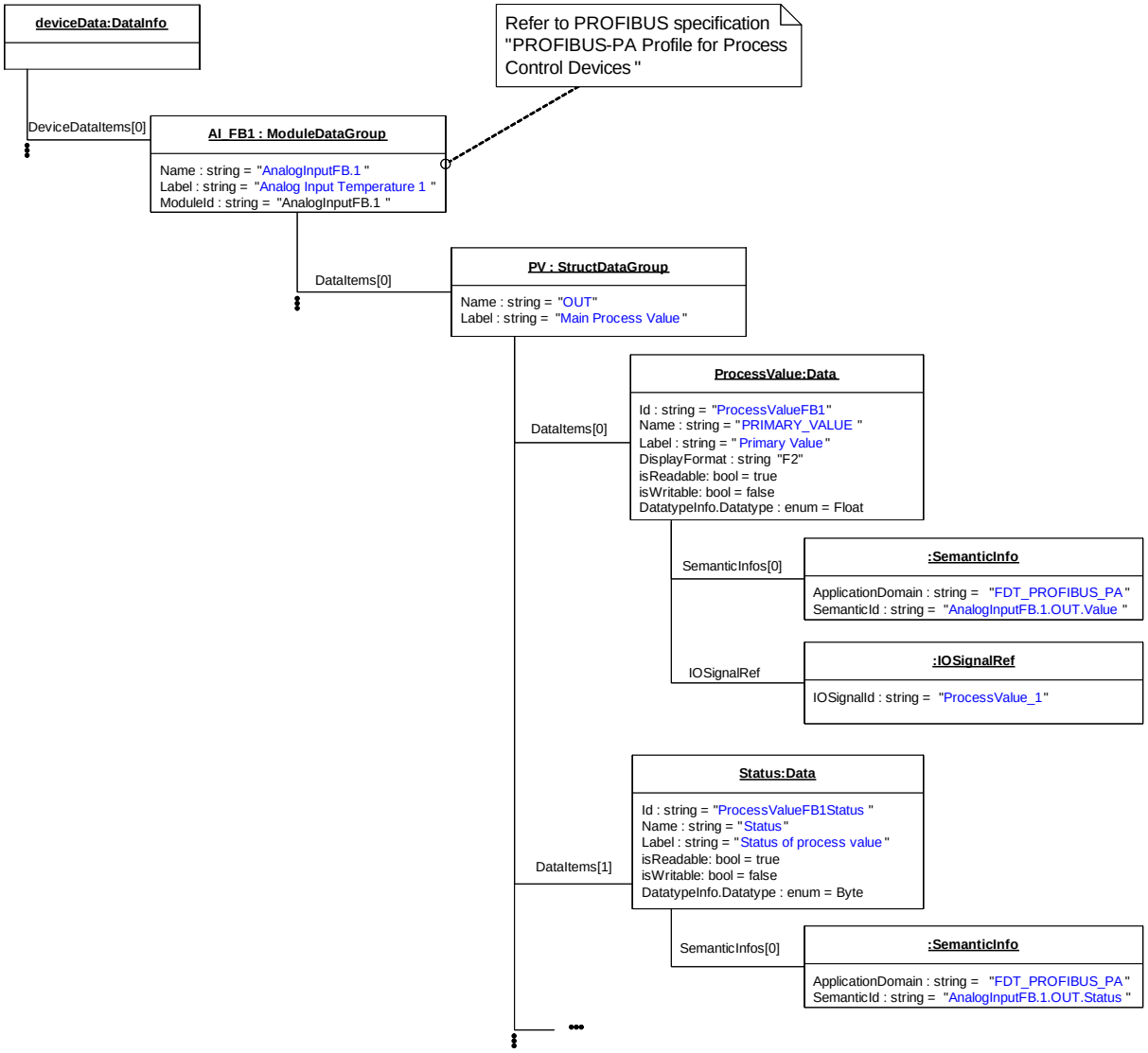
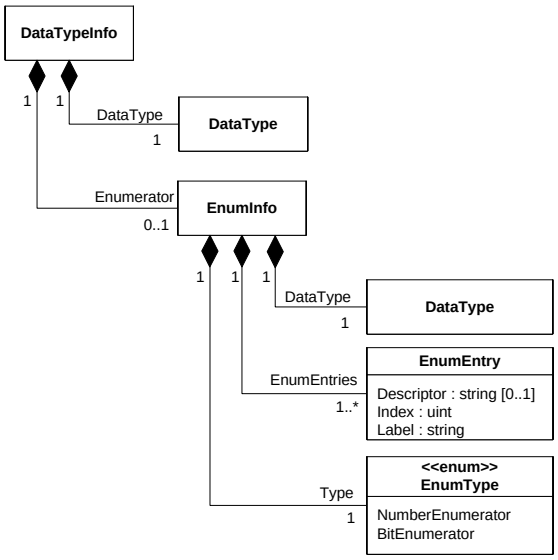


Figure 104 — Example: Providing information on structured data

If the data described in the DataInfo is provided as enumeration (DataValue = EnumValue), the EnumInfo class is used to provide the description of the value range (see Figure 105).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:
IDeviceData.EndGetDataInfo() / IInstanceData.EndGetDataInfo()

Figure 105 — EnumInfo – datatype

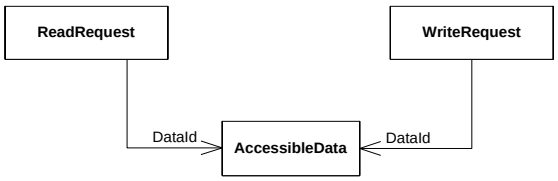
7.9.2 Datatypes used in reading and writing DeviceData

7.9.2.1 General

The IDeviceData interface provides online access to specific parameters of a device. The following chapters define datatypes used in methods for reading and writing device data.

7.9.2.2 ReadRequest and WriteRequest Datatypes

ReadRequest datatype and the WriteRequest datatype (see Figure 106 and Table 30) are used to define specific parameters which shall be read or written.



Used in:
ReadRequest: IDeviceData.BeginRead() / IInstanceData.BeginRead()
WriteRequest: IDeviceData.BeginWrite() / IInstanceData.BeginWrite()

Figure 106 — Read and Write Request – datatypes

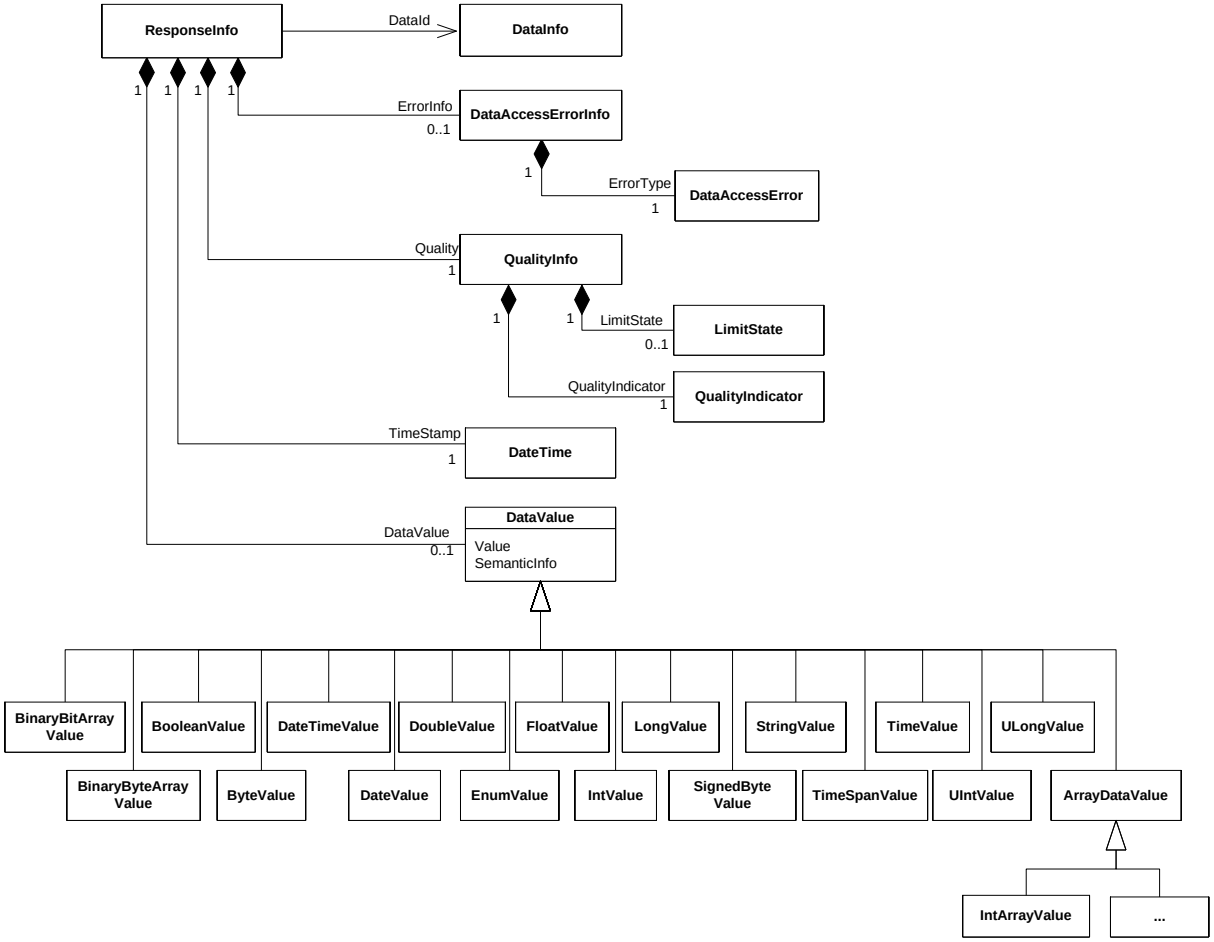
Table 30 — Reading and Writing datatype description

Datatype	Description
AccessibleData	Abstract base class for device data which is readable or writable. (See Figure 100)
ReadRequest	Read request for a single entry in DataInfo addressed by its Id.
WriteRequest	Write request for a single entry in DataInfo addressed by its Id.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

7.9.2.3 ResponseInfo Datatype

The ResponseInfo datatype (see Figure 107 and Table 31) is used to return read or written data requested by ReadRequest or WriteRequest.



Used in:

IDeviceData.EndRead()
IInstanceData.EndRead()
IDeviceData.EndWrite()
IInstanceData.EndWrite()

Figure 107 — ResponseInfo – datatype

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 31 — Reading and Writing datatype description

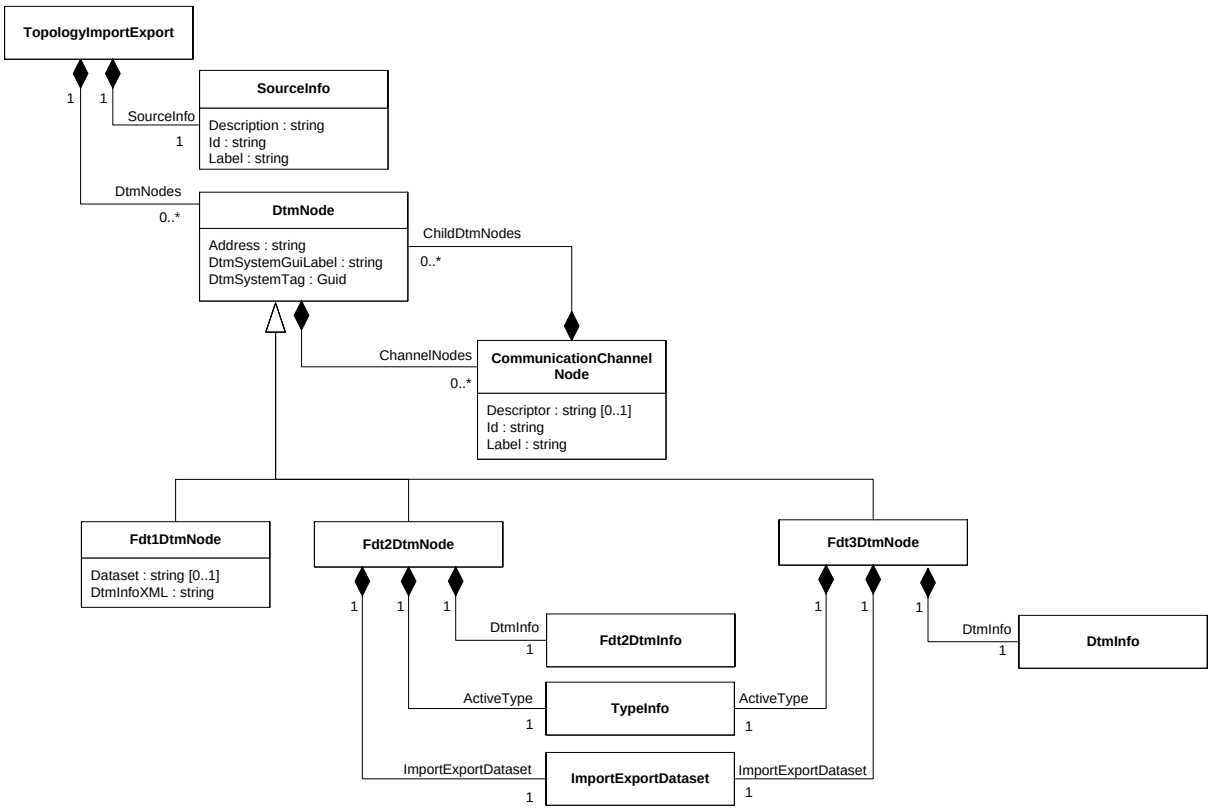
Datatype	Description
ArrayDataValue	Abstract base class for array data values provided by a DTM
BinaryBitArrayValue	A compact array of bit values.
BinaryByteArrayValue	A compact array of byte values.
ByteValue	An 8-bit unsigned integer.
DataAccessError	Information about the type of occurred error
DataAccessErrorInfo	Information about a data access error that occurred.
DataInfo	Information about available data, e.g. parameters and process values
DataValue	Abstract base class for data values provided by a DTM
DateTime	.NET System namespace: Represents an instant in time, typically expressed as a date and time of day
DateTimeValue	DataValue with a DateTime value
DoubleValue	A double-precision (64-bit) floating-point number.
EnumValue	DataValue with an enumerator value
FloatValue	A single-precision (32-bit) floating-point number.
IntArrayValue	Example for an array data value. An array of 32-bit signed integer values.
IntValue	A 32-bit signed integer.
LimitState	Limit status of device data.
LongValue	A 64-bit signed integer.
QualityIndicator	Quality status of device data.
QualityInfo	Description of the quality of device data
ResponseInfo	Read or write response for a single entry in DataInfo addressed by its Id
StringValue	A string of Unicode characters.
TimeSpanValue	DataValue with a TimeSpan value
UintValue	A 32-bit unsigned integer. Not CLS-compliant.
UlongValue	A 64-bit unsigned integer. Not CLS-compliant.

7.10 Datatypes for export and import

7.10.1 Datatypes – TopologyImportExport

The class TopologyImportExport can be used for the data exchange between different Frame Applications. The export contains the FDT topology structure information as well as information about contained (FDT1.2.x / FDT2 / FDT3) DTMs and their datasets.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:

<product-specific function of Frame Application>

Figure 108 — TopologyImportExport – datatypes

Table 32 describes the TopologyImportExport class and its related classes.

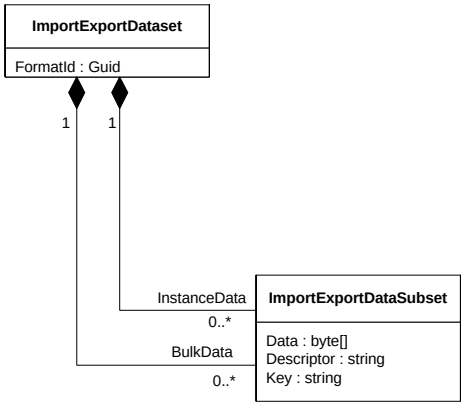
Table 32 — TopologyImportExport datatype description

Datatype	Description
CommunicationChannelNode	Represents a CommunicationChannel in the FDT Topology that is linked to further Child DTM nodes.
DtmInfo	DtmInfo contains general information about a DTM such as name, version, identifier and vendor of the software, the FDT version to which the DTM complies.
Fdt2DtmInfo	DtmInfo for an FDT2 DTM.
DtmNode	Abstract class for a DTM node in the FDT topology.
Fdt1DtmNode	This class represents a FDT1.2.x DTM in a topology export.
Fdt2DtmNode	This class represents a FDT2 DTM in a topology export.
Fdt3DtmNode	This class represents a FDT3 DTM in a topology export.
ImportExportDataset	Dataset containing the exported DTM Data Subsets.
SourceInfo	Information about the source of a topology export like unique identifier, label and description in the Frame Application that has exported the data.
TopologyImportExport	This class can be used for the data exchange between different Frame Applications. It contains the FDT topology structure information as well as information about contained (FDT 1.2.x / 2.x) DTMs and their datasets.
TypeInfo	Abstract base class used for definition of device type, block type or module type. A DTM shall contain one or more TypeInfo objects.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

7.10.2 Datatypes – ImportExportDataset

The class ImportExportDataset can be used for the data exchange between different Frame Applications. The dataset contains a DTM dataset (see Figure 95).



Used in:
DataContractSerializer.WriteObject()

Figure 109 — ImportExportDataset – datatypes

Table 33 describes the ImportExportDataset class and its related classes.

Table 33 — ImportExportDataset datatype description

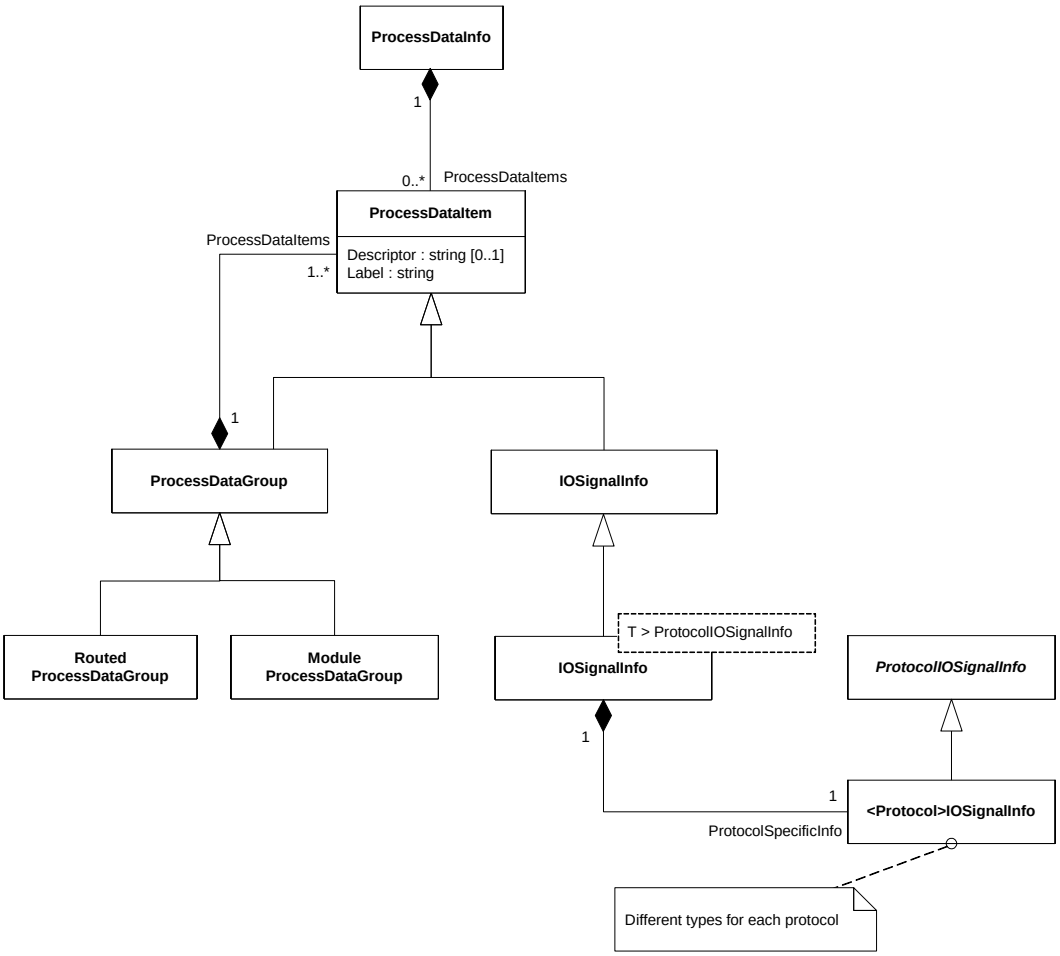
Datatype	Description
ImportExportDataset	Dataset containing the exported DTM Data Subsets.
ImportExportDataSubset	The DTMDataSubsets contains the exported binary DTM data.

7.11 Datatypes for process data description

7.11.1 Datatypes – ProcessDataInfo

The ProcessDataInfo class provides IO process data related information for the integration of the device into the control system. Figure 110 shows a class diagram with related classes of ProcessDataInfo.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:
Returned in IProcessData.EndGetProcessData()

Figure 110 — ProcessDataInfo – datatypes

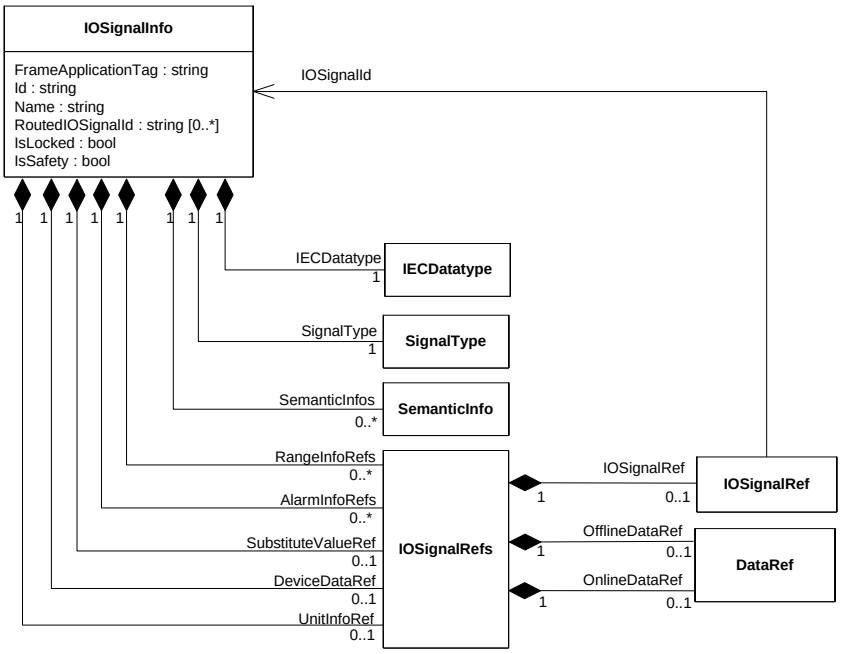
Table 34 describes the ProcessDataInfo class and its related classes.

Table 34 — ProcessDataInfo datatype description

Datatype	Description
ProcessDataInfo	Process data related information for the integration of the device into the control system like datatype, signal direction, engineering units, and ranges etc.
ProcessDataItem	Abstract base class for process data information.
ProcessDataGroup	Group of ProcessData.
RoutedProcessDataGroup	Information about routed IO signals which are originally provided by a sub-device (corresponding Child DTM in the FDT topology).
ModuleProcessDataGroup	Information about IO signals provided by a DTM module.
IOSignalInfo	Information about a single device IO signal.
IOSignalInfo<T>	Information about a single device IO signal where T is protocol-specific ProtocolIOSignalInfo
ProtocolIOSignalInfo	Abstract base class for protocol-specific IO signal class.
< Protocol>IOSignalInfo	Protocol-specific IO signal class.

The following diagram (Figure 111) provides more details on IOSignalInfo, which is used not only for ProcessDataInfo but also for ProcessImage information.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:

ProcessDataInfo class
ProcessImageSection class
IProcessData.SetIOSignalInfo()

Figure 111 — IOSignalInfo – datatypes

Table 35 — IOSignalInfo datatype description

Datatype	Description
DataRef	Reference to an item in DataInfo identified by its Id and optionally also information about the type (semantic) of the reference.
IECDatatype	IEC datatype of the IO signal. (automatically set by protocol-specific IO signal class).
IOSignalInfo	Information about a single device IO signal.
IOSignalInfo<T>	Information about a single device IO signal where T is protocol-specific ProtocolIOSignalInfo
IOSignalRef	Reference to an IO signal identified by its identifier.
IOSignalRefs	Reference to another IOSignalInfo, and/or DeviceDataInfo. The meaning of the references depends on the context where this class is used.
SemanticInfo	This class provides semantic information for a data object.
SignalType	Type of the IO signal.

The following object diagram (Figure 112) shows for example a ProcessDataInfo describing analog input values provided by a HART device. Please be aware that the example shows the expected use of datatypes defined in this document, the definition of HART related datatypes may differ from this example.

Note: The examples demonstrate how a protocol-specific datatype may be derived from the datatypes defined in this document and how such a protocol-specific datatype is intended to be used. For definition of the protocol-specific datatypes please refer to the respective specification document.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

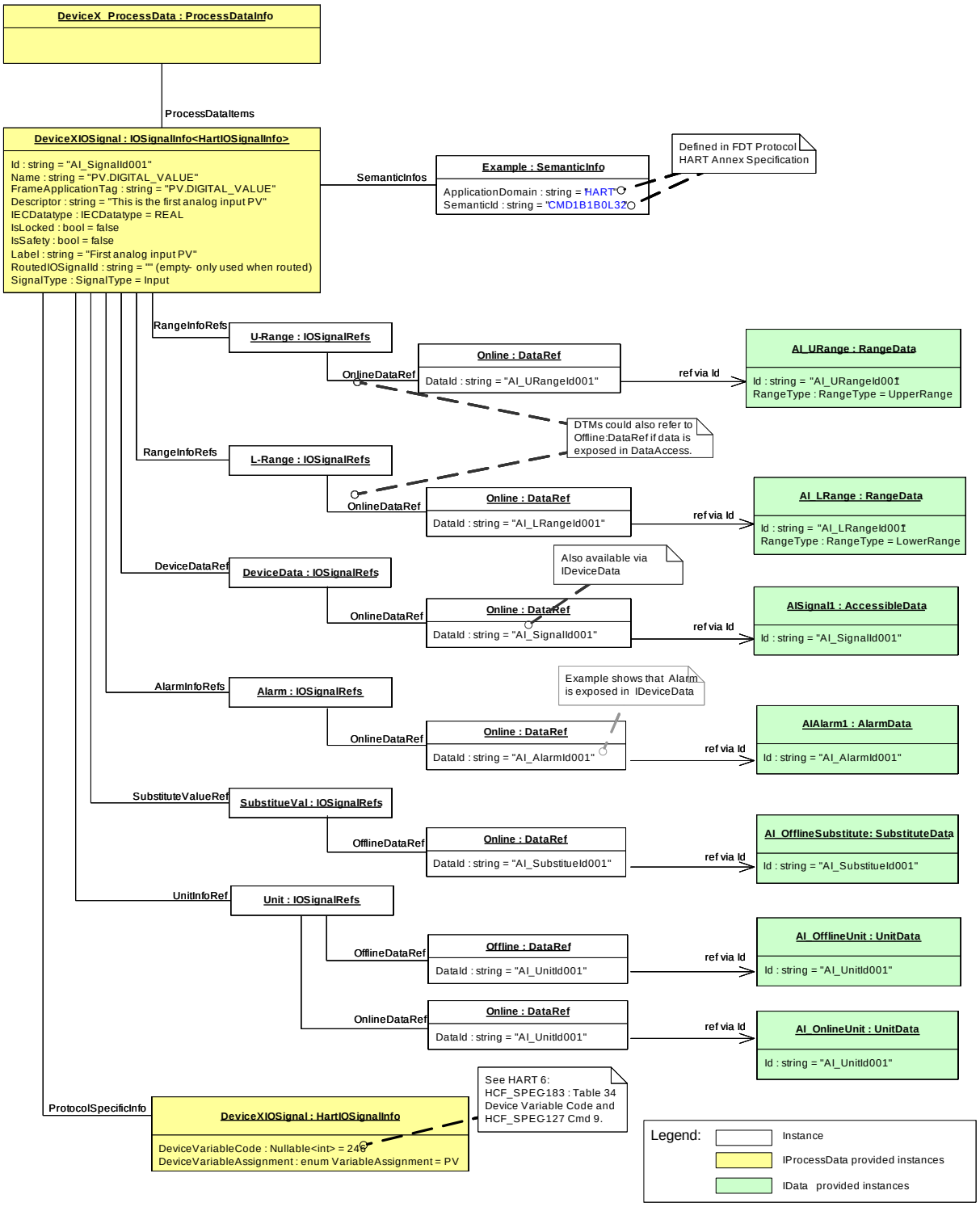


Figure 112 — Example: ProcessDataInfo for HART (UML)

The example in Figure 113 demonstrates how a (HART) Device DTM creates and returns a ProcessDataInfo instance:

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
protected ProcessDataInfo GetProcessData()
{
    // HART PV information
    HartIOSignalInfo hartPVInfo = new HartIOSignalInfo();
    //This value is the Primary Variable
    hartPVInfo.ProcessVariableAssignment = HartIOSignalInfo.VariableAssignment.PV;
    //Specify the index that is needed to read the value via command #9
    //or command #33
    hartPVInfo.Index = 0;

    // HART PV information
    IOSignalInfo<HartIOSignalInfo>  pvInfo = new IOSignalInfo<HartIOSignalInfo>();
    pvInfo.ProtocolSpecificInfo = hartPVInfo;
    pvInfo.Id = "AI_SignalId001";
    pvInfo.Name = "PV.DIGITAL_VALUE";
    pvInfo.Label = "First analog input PV";
    pvInfo.Descriptor = "This is the first analog input PV";
    pvInfo.SignalType = SignalType.Input;
    pvInfo.IECDatatype = IECDatatype.REAL;
    pvInfo.IsLocked = false;
    pvInfo.IsSafety = false;
    //Sematic info as defined in HART FDT Annex
    pvInfo.SemanticInfos = new FdtList<SemanticInfo>(new SemanticInfo("HART",
                                                                      "CMD1B1B0L32"));

    // HART PV unit information
    var dataRefPvUnit = new IOSignalRefs();
    dataRefPvUnit.OfflineDataRef = new DataRef("AI_UnitId001");
    pvInfo.UnitInfoRef = dataRefPvUnit;

    // HART PV range information
    var ioURefPvUnit = new IOSignalRefs {IOSignalRef = new IOSignalRef("AI_URangeId001")};
    var ioLRefPvUnit = new IOSignalRefs {IOSignalRef = new IOSignalRef("AI_LRangeId001")};
    pvInfo.RangeInfoRefs = new FdtList<IOSignalRefs>(){ioLRefPvUnit, ioURefPvUnit};
    // other references would come here..
    // pvInfo.SubstituteValueRef = ...

    // Process Data Info
    ProcessDataInfo processData = new ProcessDataInfo();
    processData.ProcessDataItems = new FdtList<ProcessDataItem>();
    processData.ProcessDataItems.Add(pvInfo);

    // other Process Variables may follow here ....
    // ...
    // processData.ProcessDataItems.Add(svInfo);

    return processData ;
}
```

Figure 113 — Example: ProcessDataInfo creation for HART

The example in Figure 114 demonstrates how a Frame Applications requests and uses a ProcessDataInfo instance. The protocol-specific properties shown in Figure 113 are mapped automatically to the protocol-independent properties which are used in Figure 114.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
public void ReadIoSignalInfos(IDtm3 dtm, Guid protocolId)
{
    IAsyncResult asyncResult =
        (dtm as IProcessData).BeginGetProcessData(protocolId, null, null);
    ProcessDataInfo processData =
        (dtm as IProcessData).EndGetProcessData(asyncResult);

    ShowIoSignalInfos(processData.ProcessDataItems);
}

public void ShowIoSignalInfos(FdtList<ProcessDataItem> processDatas)
{
    foreach (ProcessDataItem data in processDatas)
    {
        if (data is IOSignalInfo)
        {
            IOSignalInfo ioSignalInfo = (IOSignalInfo)data;
            MessageBox.Show("ID = " + ioSignalInfo.Id +
                            "Name = " + ioSignalInfo.Name +
                            "Label = " + ioSignalInfo.Label +
                            "SignalType = " + ioSignalInfo.SignalType);
        }
        else
        {
            ShowIoSignalInfos((data as ProcessDataGroup).ProcessDataItems);
        }
    }
}
```

Figure 114 — Example: Using ProcessData for HART

The example in Figure 115 demonstrates how the HART-specific datatype assembly (Fdt.Datatypes.Hart.dll) exposes the type information over the IOSignalInfoType attribute:

```
[assembly: IOSignalInfoType(
       IOSignalInfoType = typeof(IOSignalInfo<HartIOSignalInfo>),
       ProtocolIOSignalInfoType = typeof(HartIOSignalInfo))
]
```

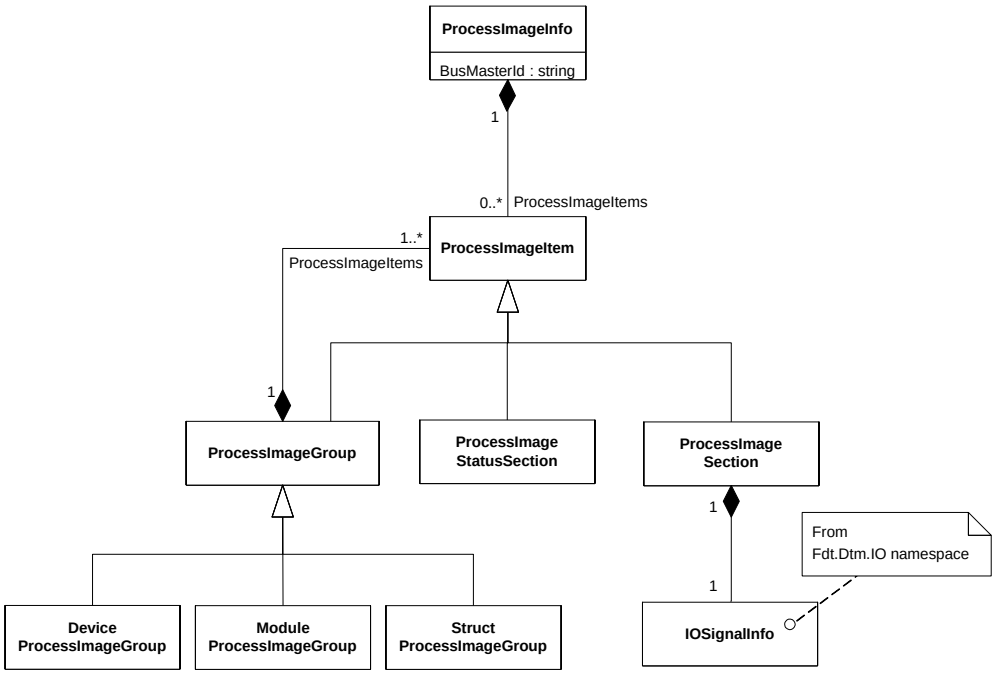
Figure 115 — Example: IOSignalInfoType attribute

Note: The above examples demonstrate how a protocol-specific datatype may be derived from the datatypes defined in this document and how such a protocol-specific datatype is intended to be used. For definition of the protocol-specific datatypes please refer to the respective specification document.

7.11.2 Datatypes – Process Image

The ProcessImageInfo class provides information about the process image by the bus-master device which is represented by the DTM. Figure 116 shows a class diagram with related classes of ProcessImageInfo.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:

Returned in IProcessImage.EndGetProcessImageInfo()

Figure 116 — ProcessImage – datatypes

Table 36 describes ProcessImage classes.

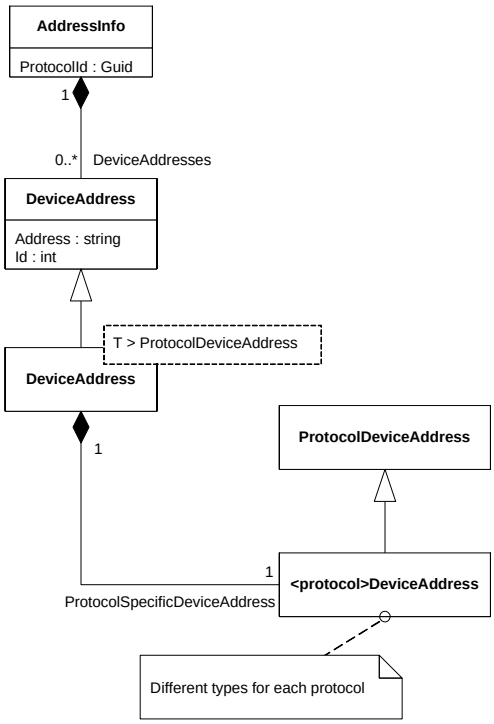
Table 36 — ProcessImage datatype description

Datatype	Description
DeviceProcessImageGroup	Groups process image items belonging to a specific device connected to the fieldbus.
IOSignalInfo	Information about a single IO signal. Note: The class is also used in IProcessData interface.
ModuleProcessImageGroup	Groups process image items belonging to a specific device module connected to the fieldbus.
ProcessImageInfo	Information about the fieldbus master process image, which enables for example engineering tools to map the device I/O signals to variables in an IEC program for a PLC.
ProcessImageItem	Abstract base class for process image information.
ProcessImageGroup	Groups process image information.
ProcessImageSection	Represents a single process image section in which an IO signal is mapped.
StructProcessImageGroup	Groups of process image items belonging to a structure IO signal.

7.12 Datatypes – Address information

The AddressInfo class provides information about address(es) of the device which is represented by the DTM. Figure 117 shows a class diagram with related classes of AddressInfo.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:

AddressInfo is returned by INetworkData.GetAddressInfo()
Single DeviceAddresses can be set by INetworkData.SetAddressInfo()

Figure 117 — AddressInfo – datatypes

Table 37 describes AddressInfo class and its related classes.

Table 37 — AddressInfo datatype description

Datatype	Description
AddressInfo	Information about address(es) of the device which is represented by the DTM.
DeviceAddress	Address of the device in the network or fieldbus. TheDeviceAddress.Id is used to indicate the relation to the corresponding NetworkData (which has the same Id value).
DeviceAddress<T>	Address of the device in the network or fieldbus. Note: T represents the protocol-specific class which defines the protocol-specific address properties.
ProtocolDeviceAddress	Abstract base class for protocol-specific device addresses.

The example in Figure 118 demonstrates how a (HART) Device DTM creates and returns an AddressInfo instance:

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
public AddressInfo CreateAddressInfoForHart(int shortAddress,
                                            string shortTag,
                                            string longTag,
                  HartDeviceAddress.AddressingModeSelection addressingMode,
                  HartLongAddress longAddress)
{
    var hartDeviceAddress = new HartDeviceAddress(shortAddress, shortTag,
                                                  longTag, addressingMode,
                                                  longAddress);
    DeviceAddress<HartDeviceAddress> deviceAddress =
    new DeviceAddress<HartDeviceAddress>();
    deviceAddress.ProtocolSpecificDeviceAddress = hartDeviceAddress;
    deviceAddress.Id = 1;
    AddressInfo addressInfo = new AddressInfo(Fdt.Hart.HartFskInfo.ProtocolId);
    addressInfo.DeviceAddresses = new FdtList<DeviceAddress>(deviceAddress);
    return addressInfo;
}
```

Figure 118 — Example: AddressInfo creation

The example in Figure 119 demonstrates how a Frame Application requests and uses the AddressInfo instance created in Figure 118:

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
public void ShowDeviceAddress(IDtm3 dtm, Guid protocolId)
{
    INetworkData networkData = dtm as INetworkData;
    if(networkData == null)
    {
        //this shall never happen because INetworkData is mandatory
        MessageBox.Show("Failure: DTM does not provide INetworkData");
        return;
    }

    AddressInfo addressInfo = networkData.GetAddressInfo(protocolId);
    if (addressInfo == null)
    {
        //this shall never happen because return value of
        //GetAddressInfo() shall never be null
        MessageBox.Show("Failure: DTM does not provide AddressInfo");
        return;
    }

    //Verify result
    try
    {
        addressInfo.Verify();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Failure in verification of AddressInfo:" + ex.Message);
        return;
    }

    if (addressInfo.DeviceAddresses == null)
    {
        //This may happen if the protocol doesn't define an addressing mechanism
        MessageBox.Show("Device provides no address information.");
    }
    else
    {
        foreach (DeviceAddress deviceAddress in addressInfo.DeviceAddresses)
        {
            //verify DeviceAddress
            try
            {
                deviceAddress.Verify();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Failure in verification of DeviceAddress:"
                    + ex.Message);
                return;
            }

            MessageBox.Show("Device Address = " + deviceAddress.Address);
        }
    }
}
```

Figure 119 — Example: Using AddressInfo

The example in Figure 120 demonstrates how the HART-specific datatype assembly (Fdt.Datatypes.Hart.dll) exposes the type information over the DeviceAddressInfoAttribute:

```
[assembly: DeviceAddressType(
            DeviceAddressType = typeof(DeviceAddress<HartDeviceAddress>),
            ProtocolDeviceAddressType = typeof(HartDeviceAddress))
]
```

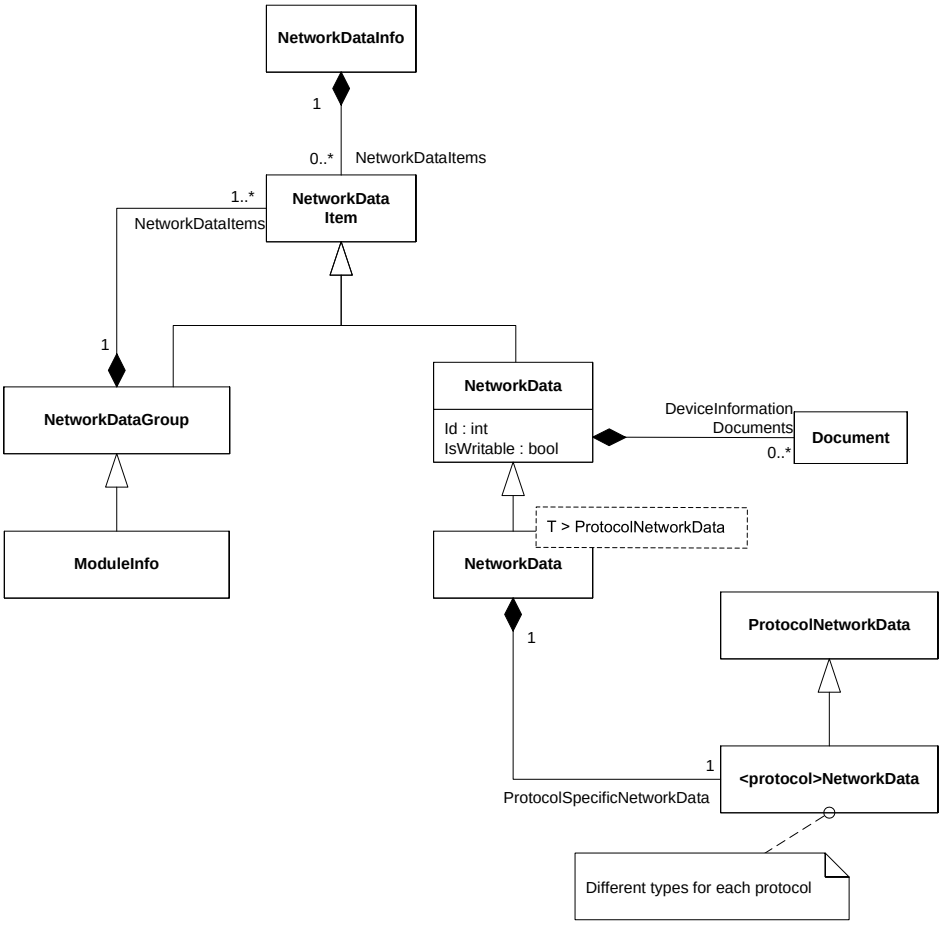
Figure 120 — Example: DeviceAddressTypeAttribute

Note: The above examples demonstrate how a protocol-specific datatype may be derived from the datatypes defined in this document and how such a protocol-specific datatype is intended to be used. For definition of the protocol-specific datatypes please refer to the respective specification document.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

7.13 Datatypes – NetworkDataInfo

The NetworkDataInfo class provides network management information which can for example be used for bus master configuration. Figure 121 shows a class diagram with related classes of NetworkDataInfo.



Used in:

NetworkDataInfo is returned by INetworkData.GetNetworkDataInfo()
Single NetworkData items can be set by INetworkData.SetNetworkData()

Figure 121 — NetworkDataInfo – datatypes

Table 38 describes NetworkDataInfo class and its related classes.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 38 — NetworkDataInfo datatype description

Datatype	Description
ModuleInfo	Represents a hardware or software module of the device. It provides general information like name, version, vendor and may also contain further NetworkDataItems providing protocol-specific information.
NetworkData	Base class for a single protocol independent network data item. The NetworkData.Id is used to indicate the relation to the corresponding DeviceAddress (which has the same Id value). The list in NetworkData.DeviceInformationDocuments is used to provide protocol-specific or device-specific support files.
NetworkData<T>	Represents a single protocol-specific network data item. Note: T represents a protocol-specific class which defines the protocol-specific network data properties.
NetworkDataGroup	Group of network data items provided by the DTM.
NetworkDataInfo	Contains network-specific information about the device. NetworkDataInfo (e.g. Label) may be used to construct the DtmSystemGuiLabel.
NetworkDataItem	Abstract base class for network data classes.
ProtocolNetworkData	Abstract base class for protocol-specific network data information classes.

The example in Figure 122 demonstrates how a (DeviceNet) Device DTM creates and returns a NetworkDataInfo instance:

```
public NetworkDataInfo GetNetworkDataInfo(Guid protocolId)
{
    // verify protocolId
    // ...

    // create network data for DeviceNet and set properties
    DeviceNetNetworkData deviceNetNetworkData = new DeviceNetNetworkData();
    deviceNetNetworkData.NodeId = new DeviceNetNodeAddress(0);
    // ...
    // ... (properties are device specific)
    // ...

    // create a document item for the ESD file
    var uri = new Uri("C:\\Users\\Public\\VendorXYZ\\DeviceABC.eds");
    var classification = Document.DocumentClassification.TechnicalDocumentation;
    var document = new Document(uri, "Device Info", classification, "text/eds");

    // create a network data item with the DeviceNet network data
    var networkData = new NetworkData<DeviceNetNetworkData>(1, deviceNetNetworkData);
    networkData.DeviceInformationDocuments = new FdtList<Document>(document);

    // create and return the network data info with the network data item
    var networkDataInfo = new NetworkDataInfo(protocolId, false);
    networkDataInfo.NetworkDataItems = new FdtList<NetworkDataItem>(networkData);

    return networkDataInfo;
}
```

Figure 122 — Example: NetworkDataInfo creation example

The example in Figure 123 demonstrates how a Communication DTM (representing a bus-master device) requests and uses the NetworkDataInfo instance created in Figure 122:

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
public void CheckNetworkData(INetworkData networkData, Guid protocolId)
{
    var dtmNetworkData = networkData.GetNetworkDataInfo(protocolId);
    if (dtmNetworkData != null)
    {
        // network data available
        // Verify the data (if invalid data, the method exits with an exception)
        dtmNetworkData.Verify();

        foreach (NetworkDataItem item in dtmNetworkData.NetworkDataItems)
        {
            if ((item is NetworkData<DeviceNetNetworkData>))
            {
                var deviceNetDataItem = item as NetworkData<DeviceNetNetworkData>;
                DeviceNetNetworkData deviceNetData =
                                     deviceNetDataItem.ProtocolSpecificNetworkData;
                foreach (var document in deviceNetDataItem.DeviceInformationDocuments)
                {
                    if (document.MediaType == "text/eds")
                    {
                        // use DeviceNet EDS file.
                    }
                }
                // use the network data
                // ...
            }
        }
    }
}
```

Figure 123 — Example: NetworkDataInfo using example

The example in Figure 124 demonstrates how the PROFIBUS-specific datatype assembly (Fdt.Datatypes.Profibus.dll) exposes the type information over the NetworkDataTypeAttribute:

```
[assembly: NetworkDataType(
            NetworkDataType = typeof(NetworkData<ProfibusNetworkData>),
            ProtocolNetworkDataType = typeof(ProfibusNetworkData))
]
```

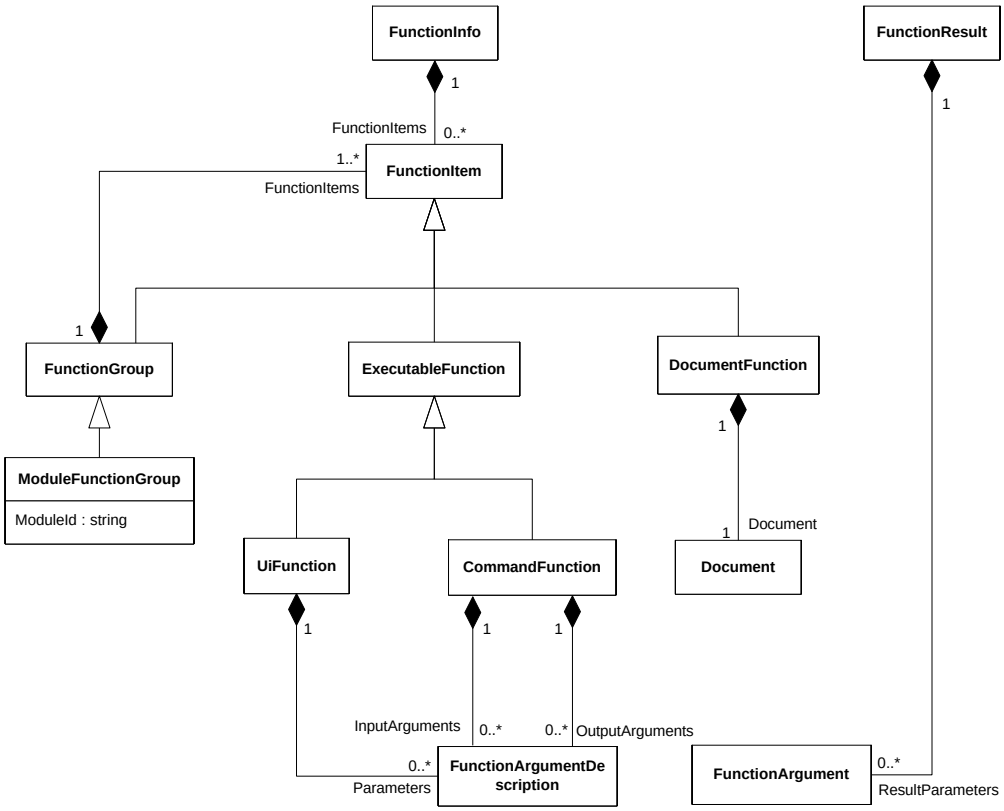
Figure 124 — Example: NetworkDataTypeAttribute example

Note: The above examples demonstrate how a protocol-specific datatype may be derived from the datatypes defined in this document and how such a protocol-specific datatype is intended to be used. For definition of the protocol-specific datatypes please refer to the respective specification document.

7.14 Datatypes – DTM functions

The following class diagram (Figure 125) describes the relations of classes in the context of FunctionInfo.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:
IFunction.FunctionInfo
ICommandFunction.BeginExecute()
ICommandFunction.EndExecute()

Figure 125 — DTM Function – datatypes

A DTM exposes all functions it provides in FunctionInfo, which can contain one or more functions. Each of the functions can be a function providing one or more documents, an executable function or a function group. Function groups contain one or more functions. ModuleFunctionGroup is a special FunctionGroup. Two types of executable functions are distinguished: a function which requires opening a user interface and a function, which is performed in the background without a user interface. UiFunctions provide one or more FunctionArguments, which describe function-specific information. For a list of functions with user interface see 5.11.1. CommandFunctions define InputParameters as well as ResultParameters.

A DTM should not make any assumption in regard to how a Frame Application represents the available functions of a DTM. For different use cases and on different platforms there are alternative ways of presenting this information to the user. That is why a DTM should not provide any customization (e.g. menu accelerators) for menus or for other GUI elements displaying the function list.

Table 39 describes datatypes in Fdt.Dtm.Functions namespace.

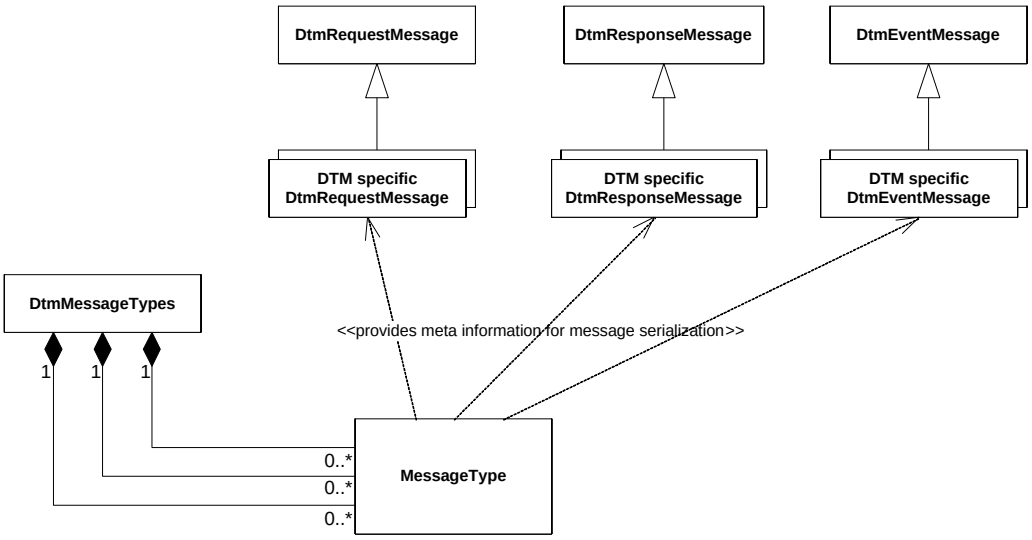
Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 39 — DTM Function datatype description

Datatype	Description
FunctionInfo	Returns information about functions, user interfaces and documents provided by a DTM.
FunctionItem	Abstract base class for a DTM function description class.
ExecutableFunction	Abstract base class for functions of a DTM which are “executable” by calling corresponding interface on the DTM Business Logic or creating a UI object.
DocumentFunction	Description of a document (file) provided by the DTM.
FunctionGroup	Group of DTM function descriptions.
FunctionResult	Result of a command function or a modal user interface.
UiFunction	Description of a graphical DTM WebUI.
CommandFunction	Description of a non-GUI function provided by the DTM Business Logic.
FunctionArgument	Information about a parameter of a CommandFunction or UiFunction.
Document	Information about a document on file disk or in the Web.

7.15 Datatypes – DTM messages

The following class diagram (Figure 126) describes the relation of classes used for interaction between DTM Business Logic and DTM WebUI as well as for interaction between different instances of DTM Business Logic of two related DTMs (e.g. for a Composite DTM).



Used in:

IDtmUiMessaging.BeginSendMessages()
IDtmUiMessaging.EndSendMessages()
Event IDtmUiMessaging.DtmSpecificEventOccured()
IDtmUiMessaging.UiMessageTypes
And
IDtmMessaging.BeginSendMessages()
IDtmMessaging.EndSendMessages()
IDtmMessaging.PrivateMessageTypes

Figure 126 — DTM Messages – datatypes

Table 34 describes datatypes related to DTM Messages.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 40 — DTM Messages datatype description

Datatype	Description
DtmRequestMessage	This abstract class serves as a base for interaction between the DTM WebUI and the DTM Business Logic as well as between DTMs (proprietary DTM to DTM interaction). This class encapsulates a message in which a DTM UI or a DTM Business Logic requests information from a DTM Business Logic. DTMs shall derive own classes from DtmRequestMessage and use these for the interaction.
DtmResponseMessage	This abstract class serves as a base for interaction between the DTM WebUI and the DTM Business Logic as well as between DTMs (proprietary DTM to DTM interaction). This class encapsulates a message where a DTM Business Logic responds to a previous request. DTMs shall derive own classes from DtmResponseMessage and use these for the interaction.
DtmEventMessage	This abstract class serves as a base for interaction between the DTM Business Logic and DTM WebUI. DTMs shall derive own classes from DtmEventMessage and use these for the interaction.

DTM messages are used for interaction between DTMs and their related DTM WebUIs, including DTM WebUI (see 8.5.7). Interaction between DTMs and DTM WebUI requires the serialization of JSON data into a formatted string and deserialization of the formatted string into .NET data (and vice versa).

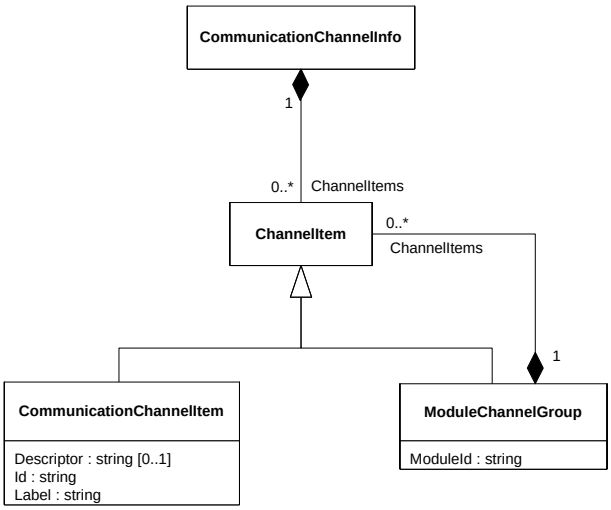
The exchanged message data shall be translatable from JavaScript structures into JSON, from JSON into the corresponding .NET message type and vice versa. Otherwise the interaction between the DTM WebUI and the DTM Business Logic will fail (either by receiving corrupted data on either end or by creating errors during the sequence).

During design of DTM-specific message data types for data exchange with a DTM WebUI this requirements needs to be considered. See F.3 for hints on implementation and on test of DTM-specific message data types.

7.16 Datatypes – CommunicationChannelInfo

The CommunicationChannelInfo class provides information about the modules and Communication Channels of a DTM (see Figure 127).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:
IChannels.CommunicationChannelInfos

Figure 127 — CommunicationChannelInfo – datatypes

Table 41 describes CommunicationChannelInfo class and its related classes.

Table 41 — CommunicationChannelInfo datatype description

Datatype	Description
CommunicationChannelInfo	Information about Communication Channels (and modules) supported by a DTM.
ChannelItem	Abstract base class for module and Communication Channel information
ModuleChannelGroup	Information about a group of Communication Channels or underlying modules of a DTM.

The example in Figure 128 demonstrates how channel information is provided by a DTM with two modules:

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
//Member variables for channel info and channel objects
CommunicationChannelInfo _myChannelInfo;
Dictionary<string,ICommunication> _myCommChannels;

private void buildChannelInfos()
{
    //create first module info
    ModuleChannelGroup module1 = new ModuleChannelGroup("Module1");
    module1.ChannelItems = new FdtList<ChannelItem>();
    CommunicationChannelItem channelModule1 =
        new CommunicationChannelItem("Module1.Chn1", "Channel of Module1");

    module1.ChannelItems.Add(channelModule1);

    //create second module info
    ModuleChannelGroup module2 = new ModuleChannelGroup("Module2");
    module2.ChannelItems = new FdtList<ChannelItem>();
    CommunicationChannelItem channelModule2 =
        new CommunicationChannelItem("Module2.Chn1", "Channel of Module2");

    module2.ChannelItems.Add(channelModule2);

    //create info list
    _myChannelInfo = new CommunicationChannelInfo();
    _myChannelInfo.ChannelItems = new FdtList<ChannelItem>();
    _myChannelInfo.ChannelItems.Add(module1);
    _myChannelInfo.ChannelItems.Add(module2);

    //create Communication Channel objects and add them to dictionary
    _myCommChannels = new Dictionary<string, ICommunication>();

    MyCommChannelType channel1 = new MyCommChannelType();
    _myCommChannels.Add(channelModule1.Id, channel1);

    MyCommChannelType channel2 = new MyCommChannelType();
    _myCommChannels.Add(channelModule2.Id, channel2);

}

//Implementation of IChannels Members
public CommunicationChannelInfo ChannelInfos
{
    get { return _myChannelInfo; }
}

public IEnumerable<KeyValuePair<string, ICommunication>> CommunicationChannels
{
    get { return _myCommChannels; }
}
```

Figure 128 — Example: Channel information

7.17 Datatypes – HardwareIdentification and scanning

7.17.1 General

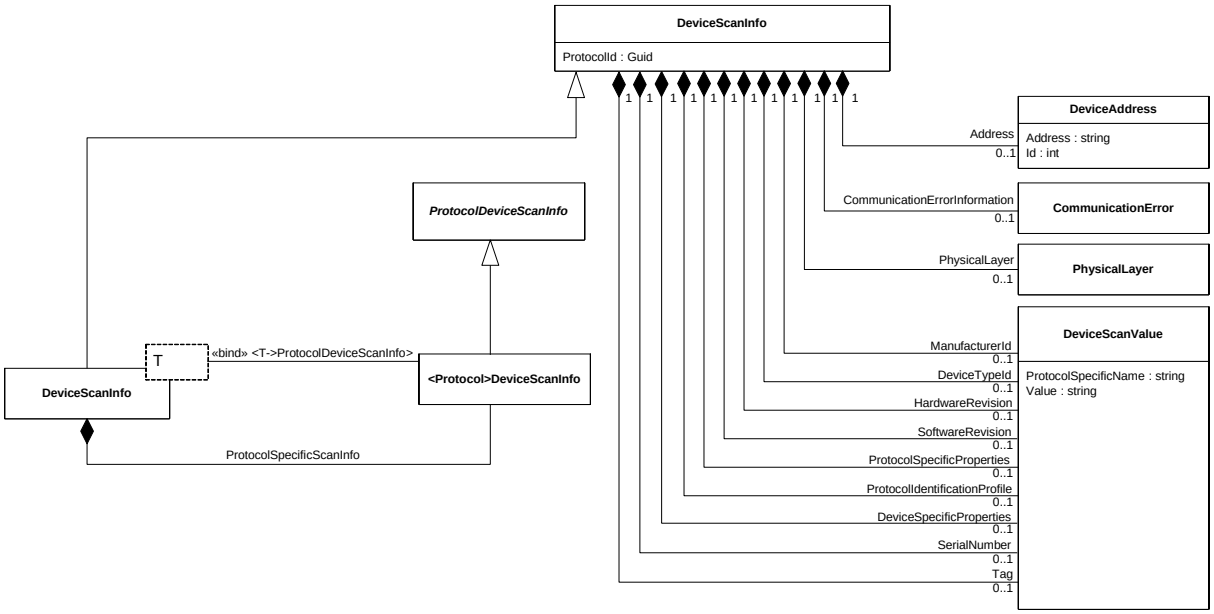
The interface IHardwareInformation is used by a Frame Application to request information from a physical device. The method EndHardwareScan() returns a DeviceScanInfo, which describes the currently connected device.

7.17.2 Datatypes - DeviceScanInfo

The methods of IScanning are used to scan the sub-topology of a Communication Channel. The result of the scan is returned in DeviceScanInfo and contains protocol-specific identification information of found devices (see Figure 129).

Note: DeviceScanInfo properties contain protocol independent identification information as name value pairs in string format. This allows a Frame Application to display basic identification information of a scanned device in a human readable style even if the protocol-specific types are unknown.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:
IHardwareInformation.EndHardwareScan()
IScanning.EndScanRequest()

Figure 129 — DeviceScanInfo – datatypes

Table 42 describes the classes related to DeviceScanInfo

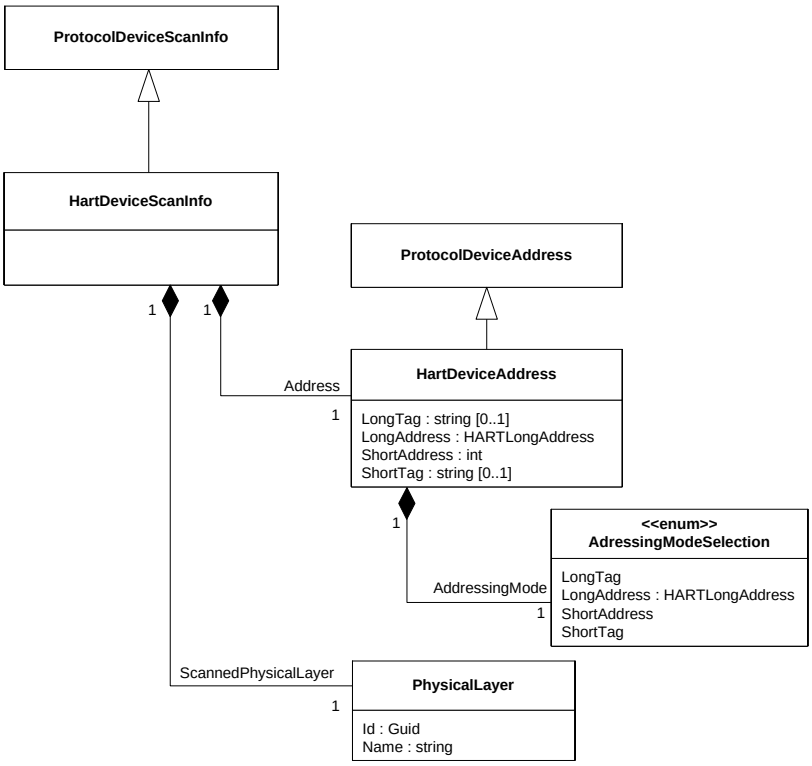
Table 42 — DeviceScanInfo datatype description

Datatype	Description
DeviceAddress	Abstract base class for protocol-specific device address. Note: For scan results the value of DeviceAddress.Id shall be set to 0.
CommunicationError	Description of a fieldbus protocol independent error which occurred during nested communication
DeviceScanInfo	This class is used to describe information from one single scanned physical device
DeviceScanValue	Represents an identification element of a scanned device. For example: Device Type Id, Manufacturer Id etc.
<Protocol>DeviceScanInfo	Is a placeholder for a DeviceScanInfo of a specific protocol. Example: HARTDeviceScanInfo
DeviceScanInfo<T>	This class is used to describe information from scanned physical devices
ProtocolDeviceScanInfo	Abstract base class for protocol-specific scan properties.

7.17.3 Example – HardwareIdentification and scanning for HART

Figure 130 shows for example the properties of the HartDeviceScanInfo Datatype.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:
IHardwareInformation.EndHardwareScan()
IScanning.EndScanRequest()

Figure 130 — Example: HARTDeviceScanInfo – datatype

Table 43 describes classes related to HARTDeviceScanInfo

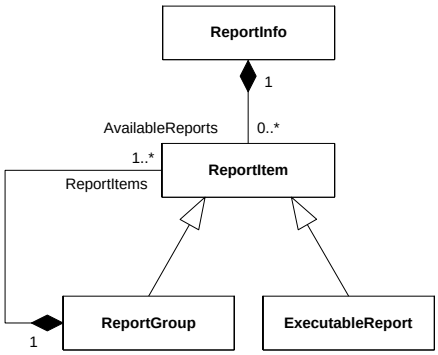
Table 43 — Example: HARTDeviceScanInfo datatype description

Datatype	Description
ProtocolDeviceScanInfo	Abstract base class for protocol-specific scan properties.
HARTDeviceScanInfo	Provides protocol-specific information returned in ScanRequest().
ProtocolDeviceAddress	Abstract base class for protocol-specific device addresses.
HartDeviceAddress	HART-specific device address.

7.18 Datatypes – DTM report types

The ReportInfo class is used by a DTM to expose information about the report types it implements. Figure 131 shows the involved classes and their relations.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used in:
IReporting.Reports

Figure 131 — DTM Report – datatypes

A ReportInfo object comprehends the description of one or many report types. Each report type (ExecutableReport) has a unique identifier, which can be used by a Frame Application to request a specific report from a DTM. Report types may be arbitrarily grouped (ReportGroups). They may have references to an ApplicationId, that associates them with an FDT standard functionality (see definition of ApplicationID in Annex B and in Annex A), or they may have a reference to a FunctionId, that links the report type to a DTM-specific functionality. If the referenced function is hidden (flag 'hidden' has value 'TRUE' for all referenced functions) the Frame Application shall not offer the possibility to generate a report for this function.

Table 44 summarizes the datatypes in the Fdt.Dtm.Reporting namespace.

Table 44 — Reporting datatype description

Datatype	Description
ExecutableReport	Information about one specific report provided by the DTM.
ReportGroup	Group of DTM report descriptions.
ReportInfo	Provides information about reports provided by a DTM.
ReportItem	Abstract base class for a report description class.

7.19 Information related to device modules in a monolithic DTM

A monolithic DTM provides information about a device with all its modules. The information regarding the modules is distributed on different datatypes. Figure 132 shows an example with involved data.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

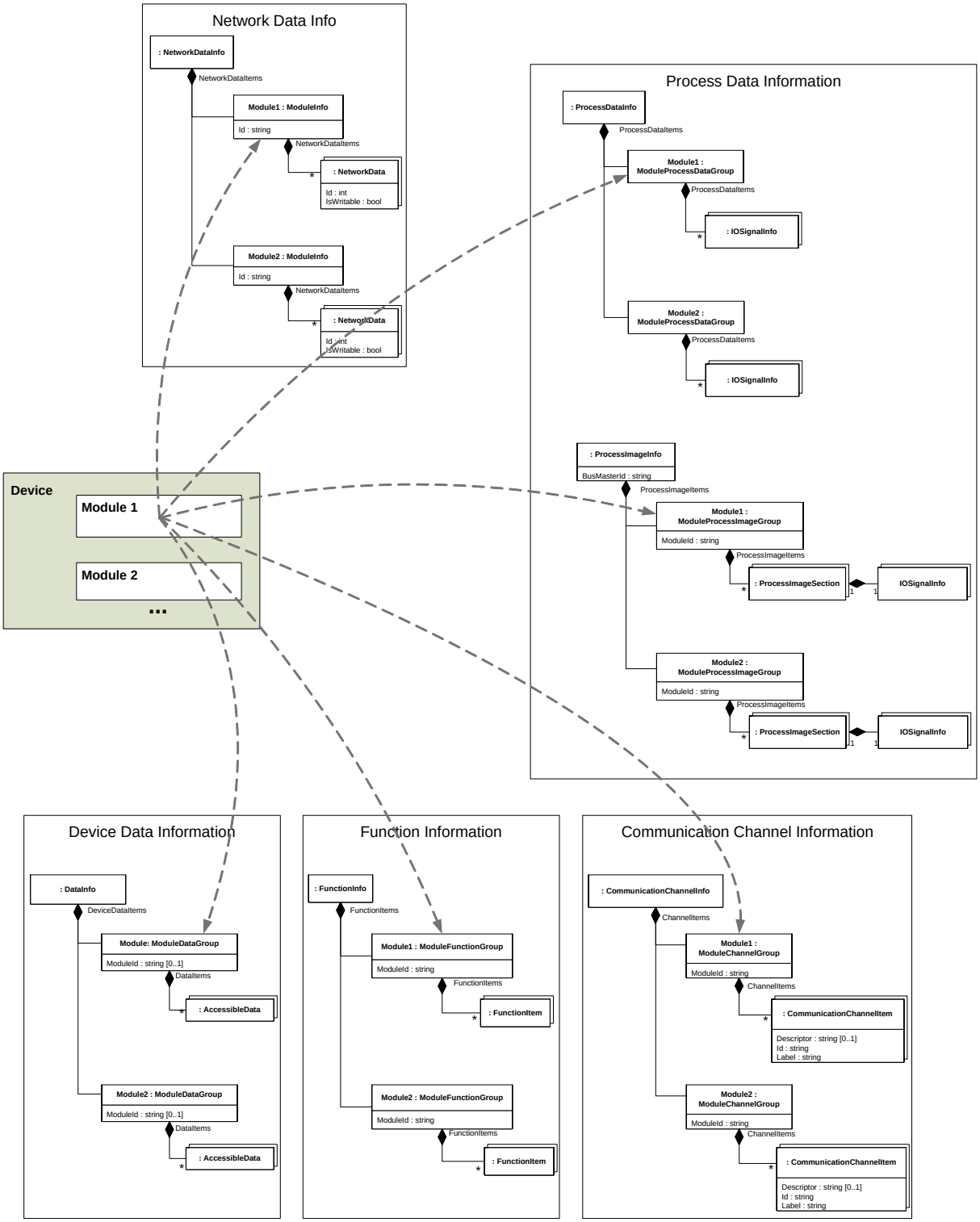


Figure 132 — Information related to device modules

Inside a monolithic DTM the modules of a device are identified by a unique ModuleId.

Note: Examples for monolithic DTMs are DTMs for PROFIBUS PA devices, where a module would represent a function block or transducer block, or DTMs for modular devices, where a module would represent a hardware module.

The same ModuleId is used in the different datatypes (ModuleInfo, ModuleProcessDataGroup, ModuleProcessImageGroup, ModuleDataGroup, ModuleFunctionGroup, and ModuleChannelGroup) in order to show that this information describes the same module.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

For a monolithic DTM it is expected that the data in IInstanceData and IDeviceData are also grouped for the modules (see Figure 102).

With ModuleFunctionGroup it is possible to provide functions specifically for modules.

If the device is a modular device and if modules provide communication, it is possible to use ModuleChannelGroup in order to associate the provided CommunicationChannels to their respective modules.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8 Workflows

8.1 General

The work flows provided in this chapter are intended to explain the expected behavior. Implemented behavior may vary, but should follow the general rules explained here and in the interface definitions.

The conventions for sequence diagrams are explained in I.6.

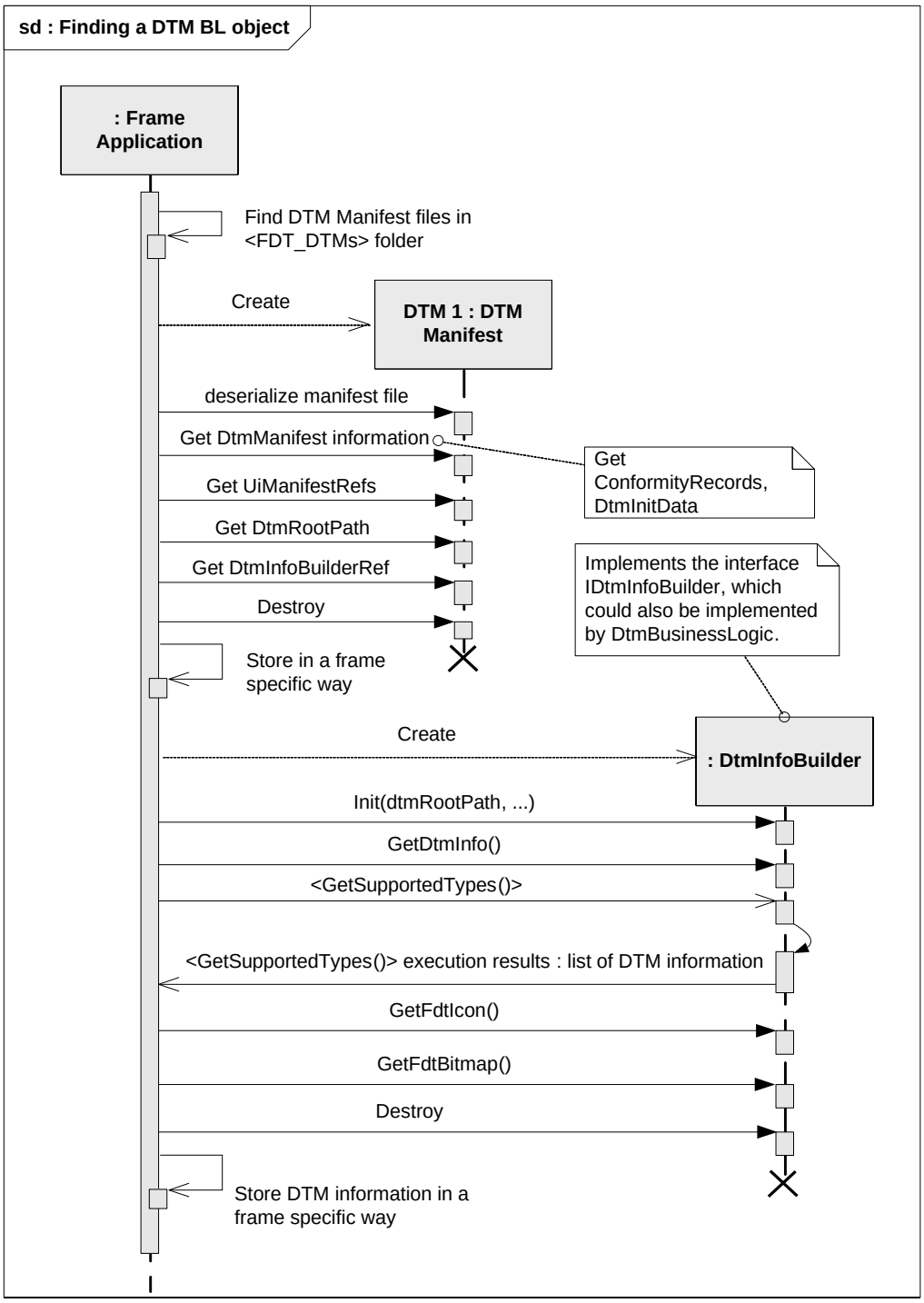
As explained in 5.8.2 all component interactions are passed through the Frame Application or through proxy components. Since this passage shall not change interactions or inject interaction requests, it will not change the general sequence of message calls. In order to simplify the representation of sequences, the proxy objects often are omitted in the sequence diagrams in this section. If the proxy objects are important to understand the sequence of message calls, then they are shown in the sequence diagrams.

8.2 Instantiation, loading and release

8.2.1 Finding a DTM BL object

In order to execute a DTM, the Frame Application needs to find the respective DTM Business Logic object, which is located in an assembly. This section describes the sequence of finding the DTM BL object (see Figure 133).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IDtmInfoBuilder2.Init()
IDtmInfoBuilder2.GetDtmInfo()
IDtmInfoBuilder2.BeginGetSupportedTypes() / IDtmInfoBuilder2.EndGetSupportedTypes()
IDtmInfoBuilder2.GetFdtIcon()
IDtmInfoBuilder2.GetFdtBitmap()

Figure 133 — Finding a DTM BL object

Typically a Frame Application stores the information about the DTM BL in a catalogue. For a more complete sequence for updating the device catalogue refer to 10.4.3.

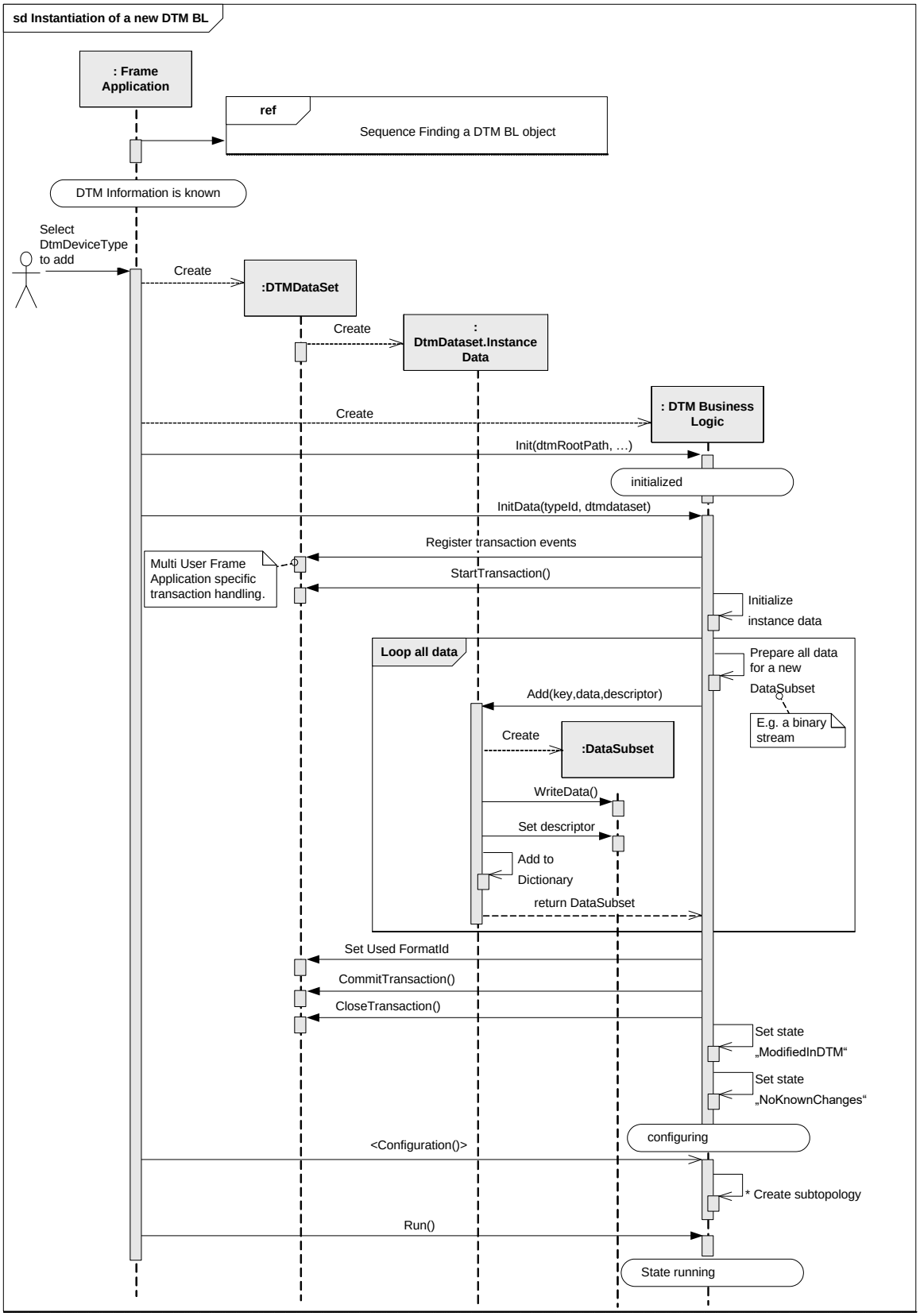
Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8.2.2 Instantiation of a new DTM BL

A new DTM Business Logic is instantiated by the Frame Application with its full assembly class name. The class name can be looked up in the DtmManifest for the selected DtmDeviceType.

The Frame Application shall create a new DTMDataset object and pass a reference to IDataSet as a parameter in IDtm3.InitData() to the newly created DTM Business Logic instance. Within the InitData() call, the DTM Business Logic instance adds new DTMDataSubsets to the DTMDataset and writes its instance data into the DTMDataSubsets (see Figure 134).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IDtm3.Init()
IDtm3.InitData()
IDataset.StartTransaction()
IDataset.CommitTransaction()
IDataset.CloseTransaction()

Figure 134 — Instantiation of a new DTM BL

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

A Frame Application may use the IDtmAssemblyInfo interface of DtmInfoBuilder object to verify the digital signatures of the DTM assemblies (see chapters 5.18.3 and 6.3).

8.2.3 Configuring access rights

The Frame Application can provide a separate function to configure the access rights for individual users or for group of users, which will be working with the DTM. The function for access rights configuration is usually available only for system administrators, which are responsible for the security of the plant.

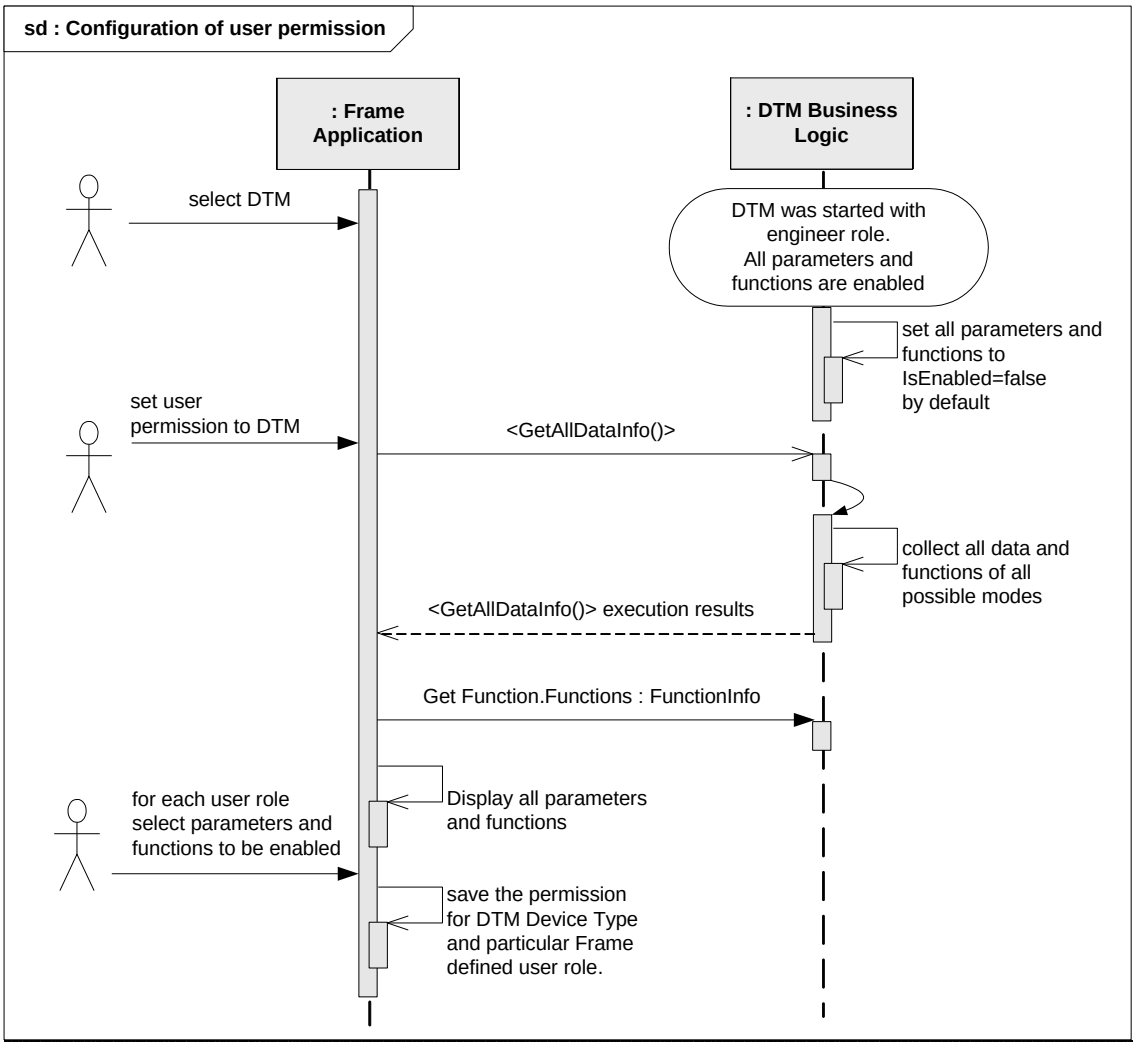
To configure the access rights, the administrator has to instantiate the DTM with access rights set to Engineer, get the list of all data provided by the DTM by using ICustomConfiguration.<GetAllDataInfo()>. The Administrator will get the list of all functions from IFunction.FunctionInfo property. The Administrator will use a specialized user interface, provided by the Frame Application to define the permissions for changing data and invoking functions.

This information shall be saved by the Frame Application and used when the DTM of this type is instantiated.

The user can invoke the DTM with Expert user level (see 8.2.5) and verify the correctness of the settings. The administrator may come back to the specialized user interface, provided by the Frame Application, correct the permissions, save the data with the rest frame data and invoke the DTM again.

The following sequence diagram illustrates the configuration of the user permissions, when custom role is invoked (see Figure 135).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

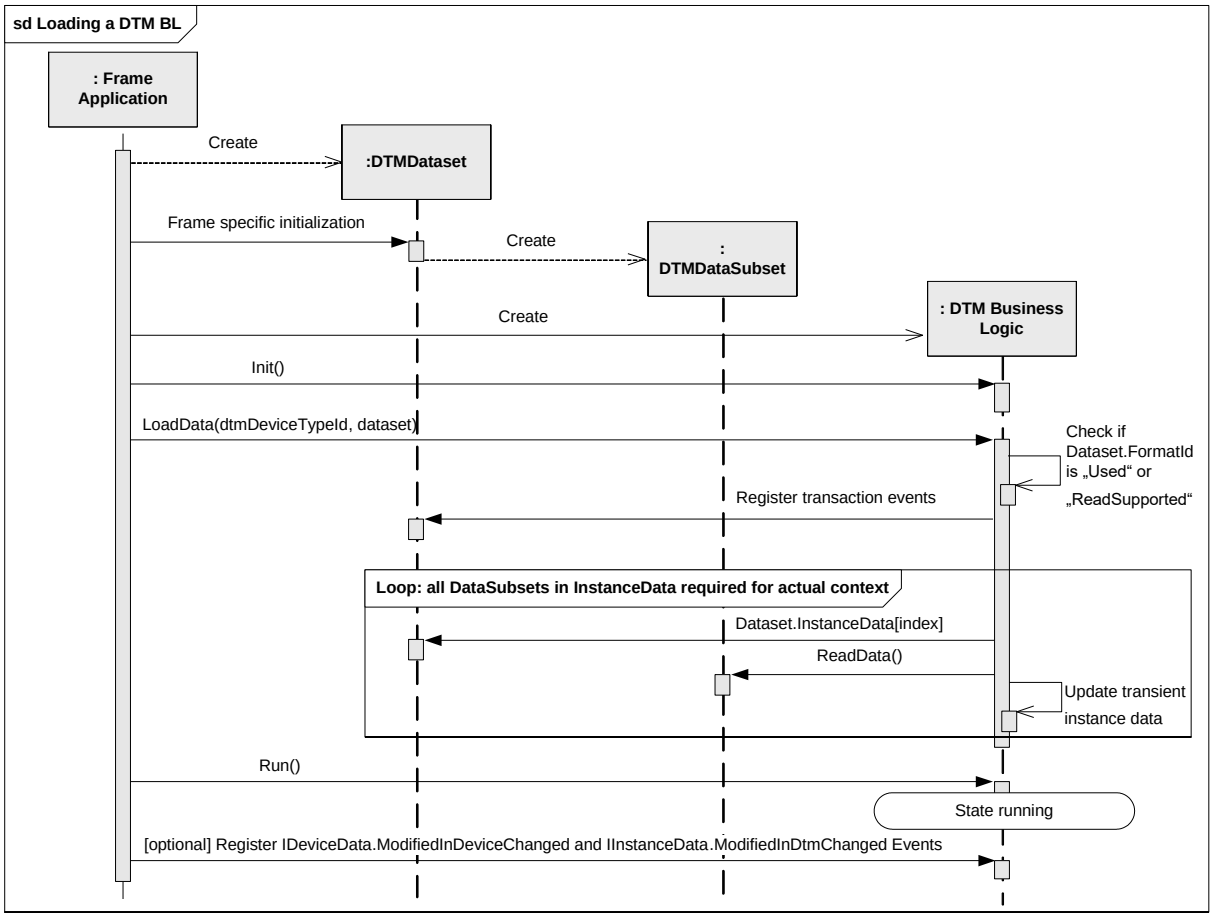
ICustomConfiguration.BeginGetAllDataInfo()
ICustomConfiguration.EndGetAllDataInfo()

Figure 135 — Configuration of user permissions

8.2.4 Loading a DTM BL

After creation of a DTM Business Logic instance for an existing DTMDataset the Frame Application shall call Init() and LoadData(), pass the identifier of the represented type (device, module, block) and the interface of the corresponding DTMDataset. The DTM checks the FormatId of the DTMDataset and reads the InstanceData from the DTMDataset to initialize its device data (see Figure 136).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IDtm3.Init()
IDtm3.LoadData()
IDataSubset.ReadData()
Event IDeviceData.ModifiedInDeviceChanged()
Event IInstanceData.ModifiedInDtmChanged()

Figure 136 — Loading a DTM BL

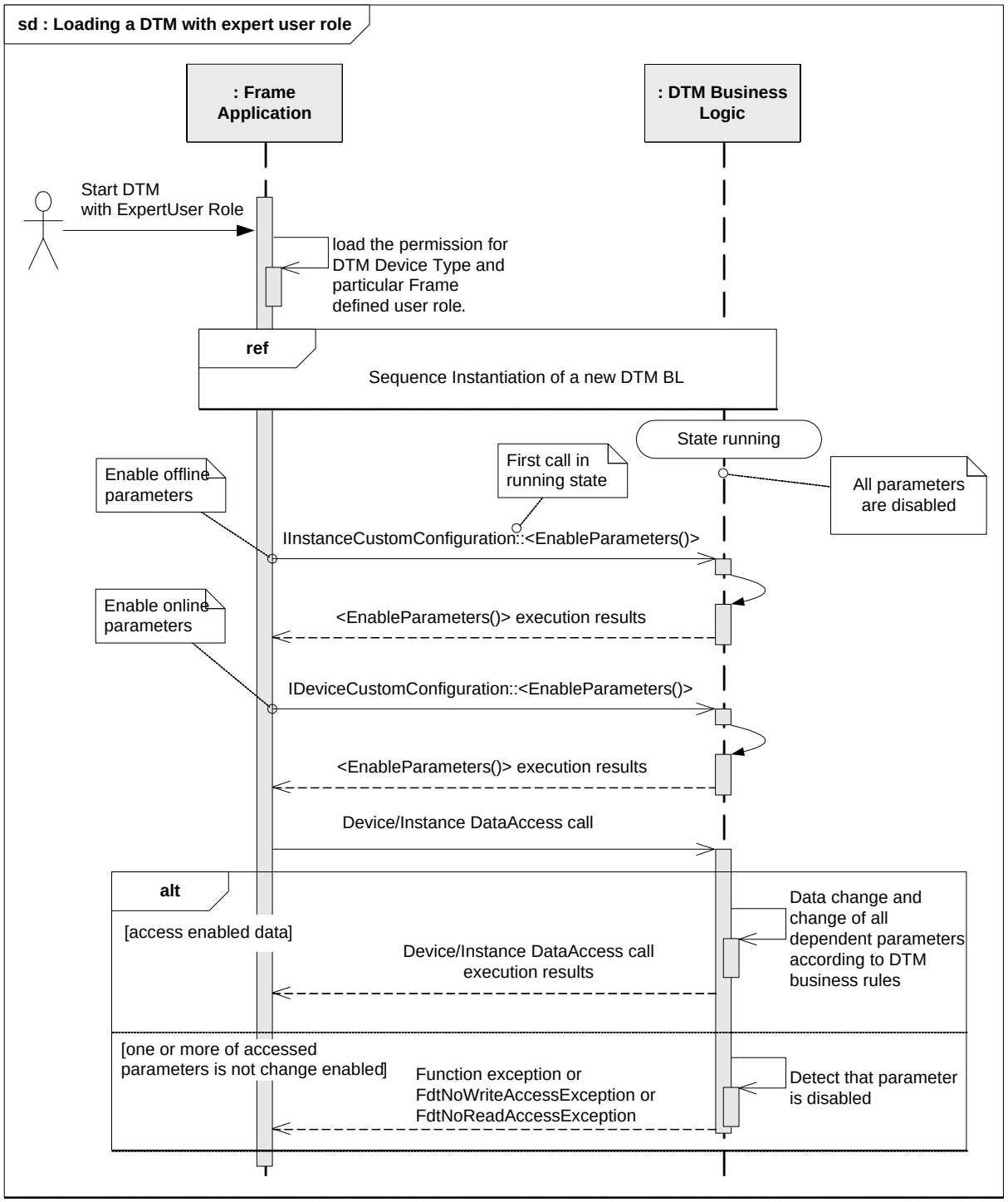
A DTM Business Logic shall read DTMDataSubsets of Datasets InstanceData on demand when the data are required according to a business function context. When transient data are not accessed any more, a DTM shall release the transient data and reload it from the Dataset if needed (see 8.3.1).

8.2.5 Loading a DTM with Expert user level

While loading the DTM with Expert user level, the Frame Application shall set the access rights using the ICustomConfiguration.<EnableParameters()>.

If the Frame Application does not invoke the ICustomConfiguration methods to grant permissions, the user will have restricted access as if the DTM is invoked with the Observer user level (see Figure 137).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

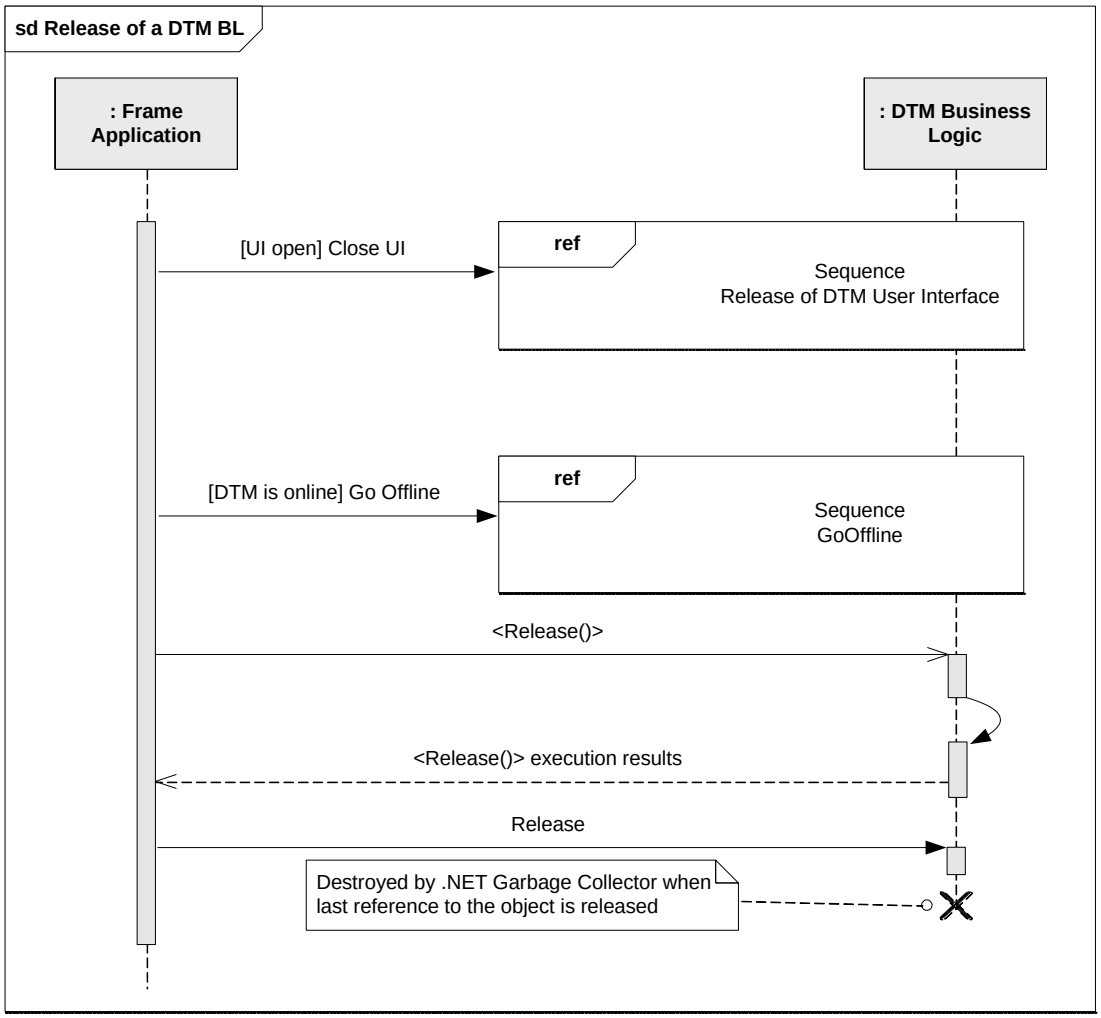
IInstanceCustomConfiguration.BeginEnableParameters() /
IInstanceCustomConfiguration.EndEnableParameters()
IDeviceCustomConfiguration.BeginEnableParameters() /
IDeviceCustomConfiguration.EndEnableParameters()

Figure 137 — Loading a DTM with Expert user level

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8.2.6 Release of a DTM BL

In order to release a DTM BL all ongoing activities need to be terminated (see Figure 138).



Used methods:
IDtm3.BeginRelease() / IDtm3.EndRelease()

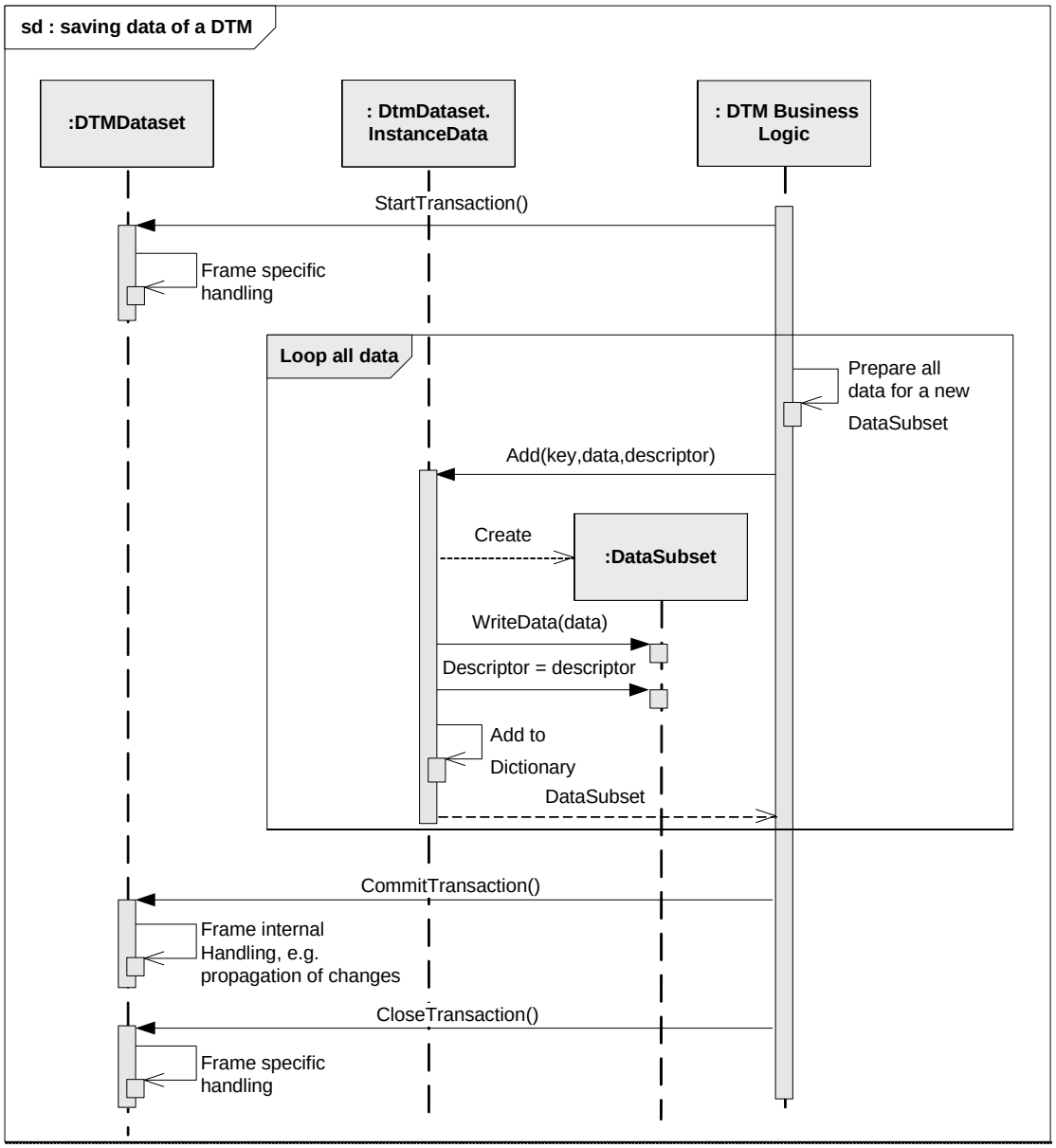
Figure 138 — Release of a DTM BL

8.3 Persistent storage of a DTM

8.3.1 Saving instance data of a DTM

The DTM instance saves its instance data on demand in a DataTransaction. The Frame Application can release a DTM only if no DataTransaction is active. If the Frame Application performs an action which requires that all data is committed (i.e. saving of a project file) then it shall check if there are no open transactions. If there are open transactions then the Frame Application should inform the user and list the DTMs which have open transactions and thereby may have uncommitted data (see Figure 139).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IDataSet.StartTransaction()
IDataSet.CommitTransaction()
IDataSet.CloseTransaction()
IDataSubsetDictionary.Add()

Figure 139 — Saving data of a DTM

8.3.2 Copy and versioning of a DTM instance

Saved datasets can be copied by a Frame Application from one device node to a different device node. The copied Dataset is loaded with LoadData() into instances of the corresponding device node.

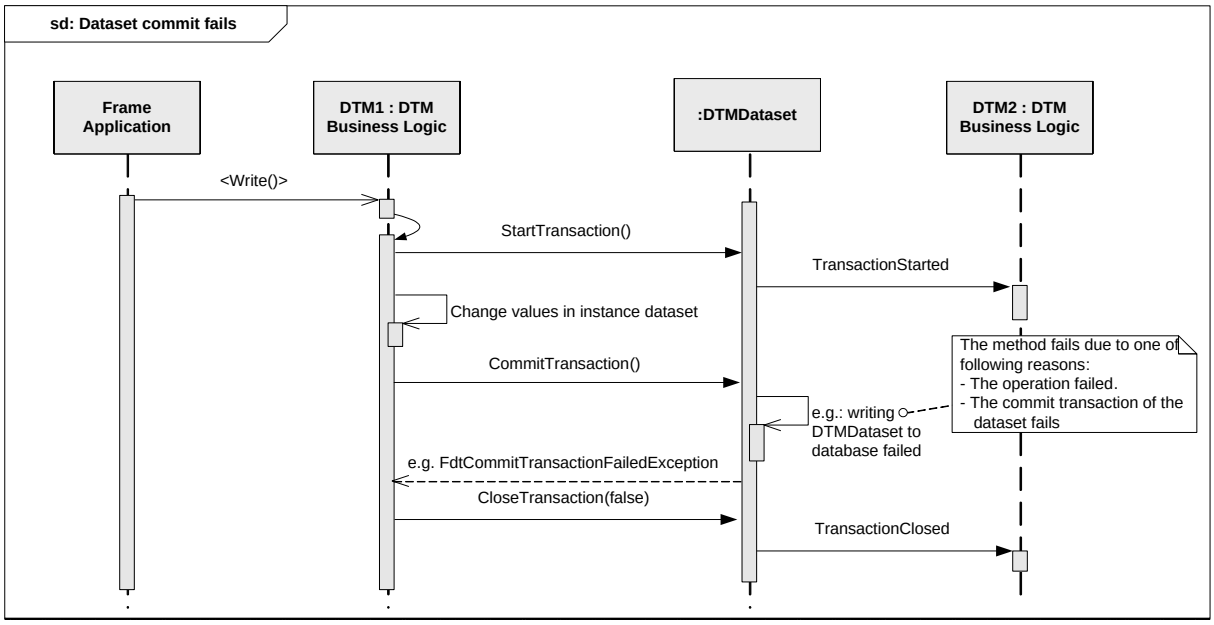
The Frame Application is responsible to handle the Frame-Application-specific versioning aspects and to manage the different instance datasets (e.g. fieldbus address and device tag) for a device.

8.3.3 Dataset commit failed

The following workflow describes the expected behavior if committing changes in the dataset fails. This exception is usually caused by a serious problem in the Frame Application. The

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Frame Application shall inform the user that the latest changes could not be saved and to release the DTM (see Figure 140).



Used methods:

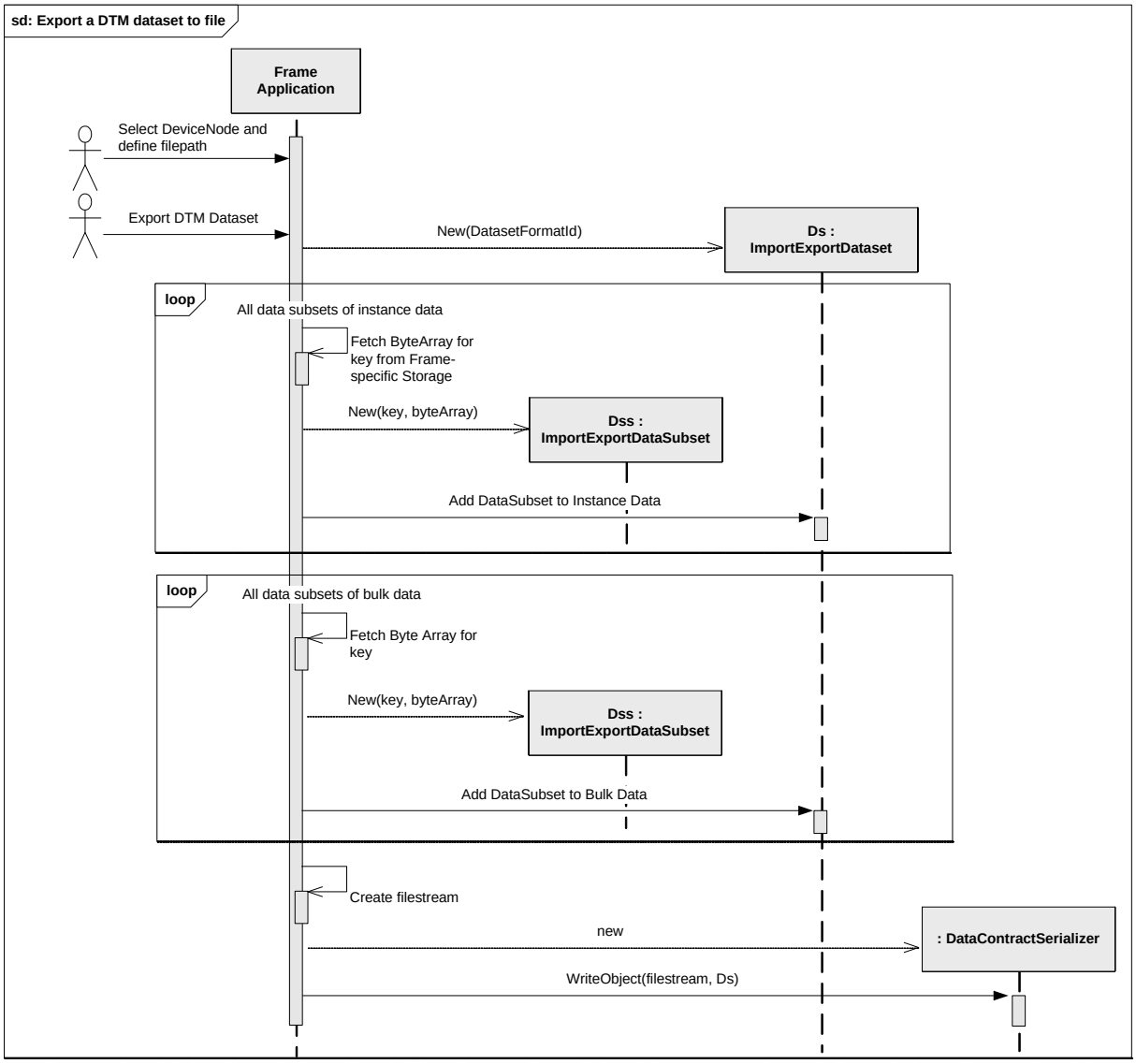
IDataSet.StartTransaction()
IDataSet.TransactionStarted()
IDataSet.CommitTransaction()
IDataSet.TransactionClosed()
IDataSet.CloseTransaction()
IDataSubsetDictionary.Add()

Figure 140 — Dataset commit failed

8.3.4 Export a DTM dataset to file

The following diagram (Figure 141) shows the use of datatypes for exporting the data of a DTM instance to a file.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

-

Figure 141 — Export a DTM dataset to file

8.4 Locking and DataTransactions in multi-user environments

8.4.1 General

Within a multi-user environment it is common, that more than one DTM instance has access to the same dataset. To synchronize DTMs which are started by several users on different PCs FDT provides a locking mechanism. Target for this event mechanism is that only one DTM has read/write access to the instance dataset and to the device data. All other DTMs have read access only.

For this reason a DTM shall lock its dataset with StartTransaction() only if required and only during modification of the data. After the data is committed and the data is not further under modification, the DTM shall unlock its dataset with CloseTransaction() immediately to enable concurrent access to the data by other DTM instances within a multi-user environment.

- The DTM shall start a DataTransaction before an activity is started, that may change the instance data (e.g. upload, IInstanceData.<Write()>) or the data in the device (e.g. OnlineParameterize). The DTM shall close the DataTransaction after the activity is finished.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

If instance data is changed, then the DTM shall save the Dataset before closing the DataTransaction. E.g.:

  - While a DTM WebUI is opened the DTM shall try to start a DataTransaction if write access is needed. If successful, all user input fields can be enabled. If the start of the DataTransaction failed, user input fields shall be disabled. After closing all DTM WebUIs in case of a locked Dataset the DTM should write modified DTMDataSubsets and commit the Dataset and close the DataTransaction after the Frame Application has saved the Dataset.

- A Frame Application shall return a negative result when a DTM calls StartTransaction while a second DTM has already an open DataTransaction. (The property LockResult.IsLocked will be set to false.)
- The Frame Application shall throw an exception if a DTM writes DTMDataSubsets while this DTM does not have an open DataTransaction.
- The DTM shall keep a DataTransaction open as short as possible. It is not allowed to set the lock for the whole time that a DTM is in states ‘running’ and ‘communicationAllowed”.
- If committing the dataset fails, then the transaction shall be closed without saving (CloseTransaction(false)) and the user shall be informed. It is recommended to stop working with the DTM.
- If closing the transaction fails, then the user shall be informed. It is recommended to stop working with the DTM.
- If a DTM receives the event TransactionCommitted(), it is mandatory to update the instance data from storage.

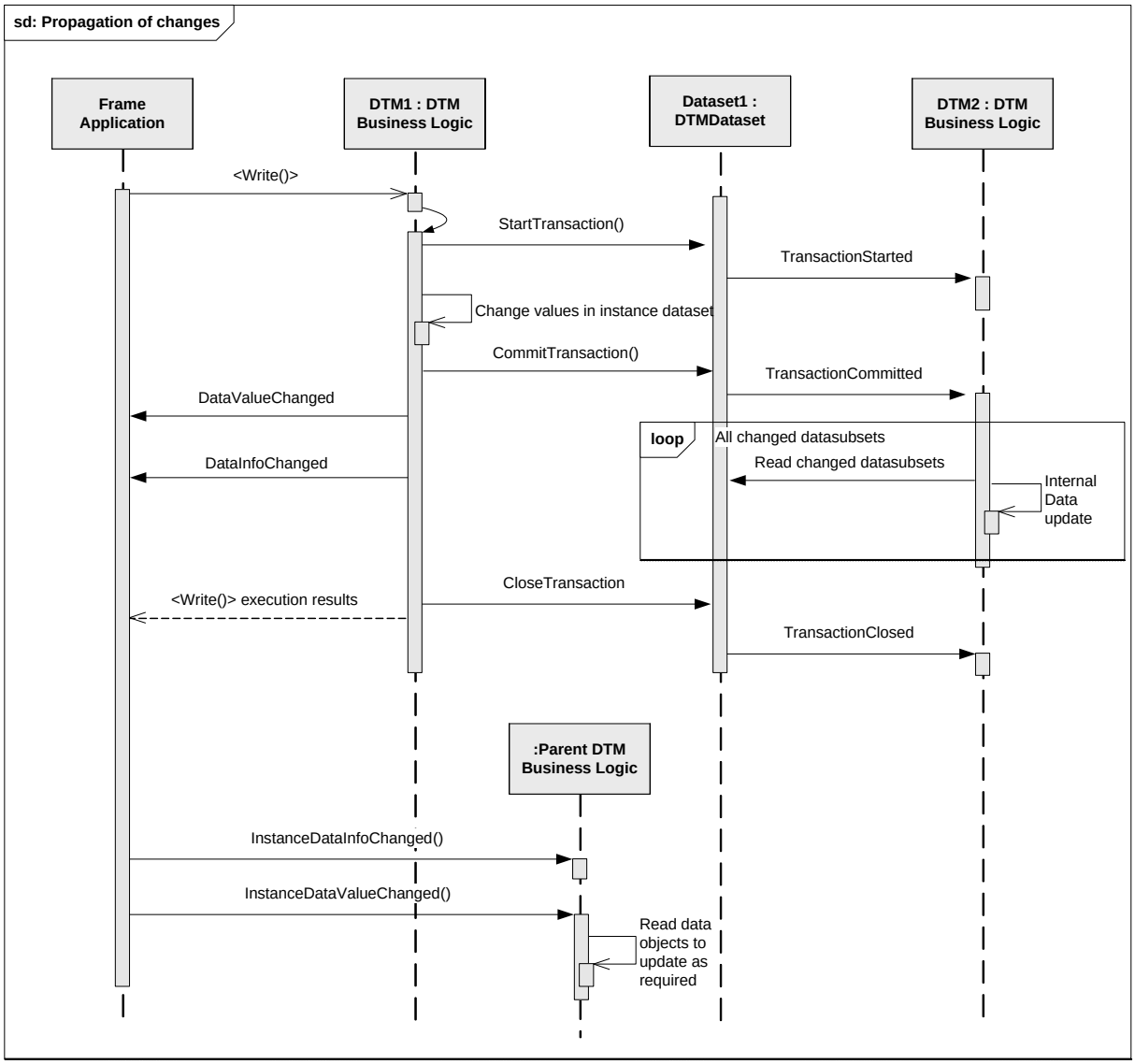
8.4.2 Propagation of changes

When multiple DTM instances are executed in a multi-user environment for the same device (see 4.8.2) and one DTM instance is changing the DataSet, the other DTM instance receive notifications indicating the process of change (TransactionStarted, TransactionCommitted, TransactionClosed).

Receiving the event TransactionCommitted indicates that the data in the persisted DataSet has been changed and that the DTM shall update the instance data from the storage.

The following sequence diagram (Figure 142) shows how changes in the instance dataset of one DTM instance (DTM1) are propagated to other DTM instances (DTM2) and to the Parent DTM.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IDataset.StartTransaction() / IDataset.CommitTransaction() / IDataset.CloseTransaction()
IDataset.TransactionStarted / IDataset.TransactionCommitted /IDataset.TransactionClosed
IInstanceData.BeginWrite() / IInstanceData.EndWrite()
Event IInstanceData.DataValueChanged()
Event IInstanceData.DataInfoChanged()
Event IChildDtmEvents.InstanceDataValueChanged()
Event IChildDtmEvents.InstanceDataInfoChanged()

Figure 142 — Propagation of changes

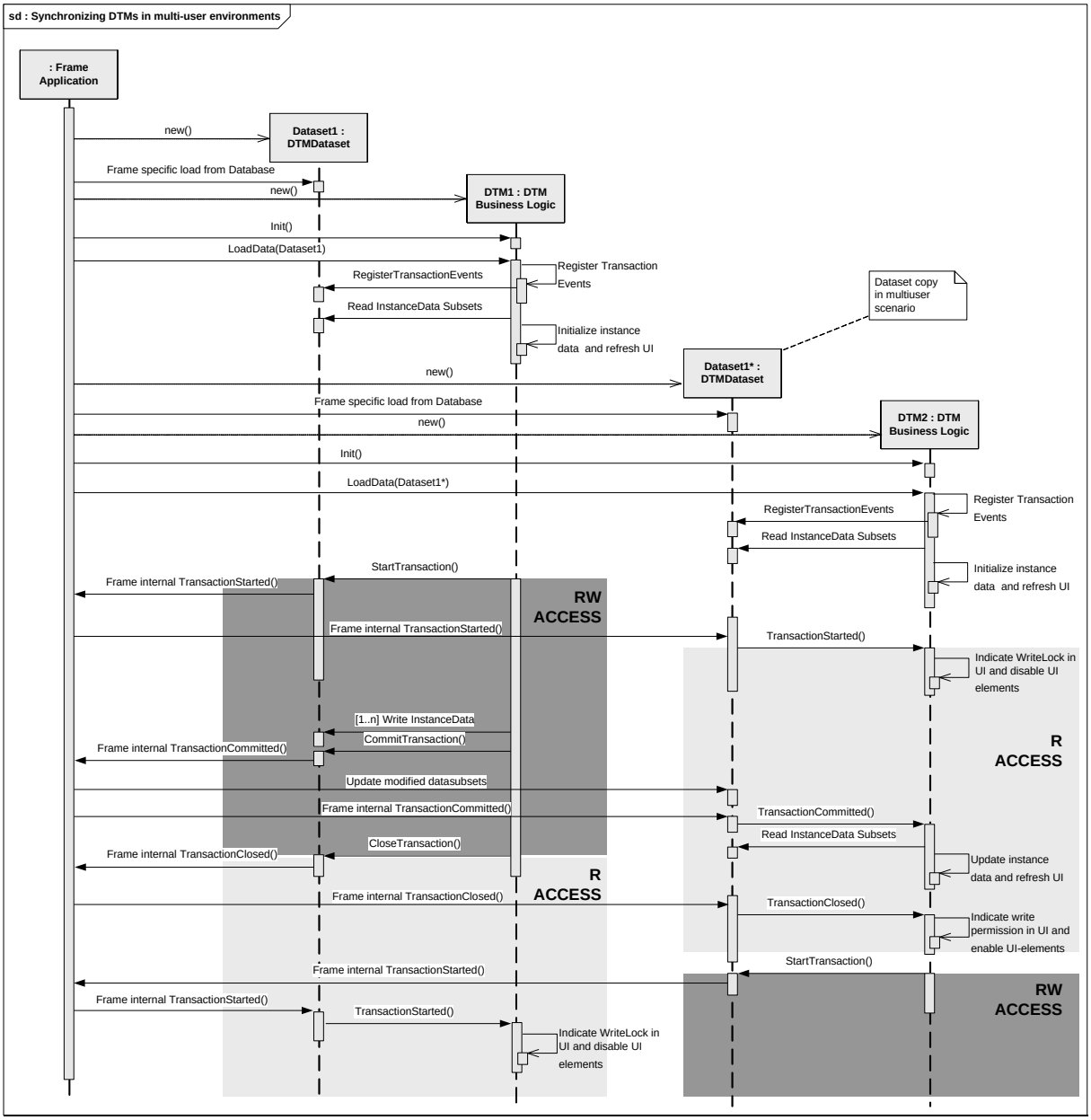
The figure above shows how a DTM instance ("DTM2") receives notifications on changes and how it updates its instance data, because the dataset was changed by a different DTM instance ("DTM1").

Note: For simplification, it is not shown here how "DTM2" reads already committed data while "DTM1" is still modifying data in an open transaction (refer to Figure 143, which shows this scenario).

8.4.3 Synchronizing DTMs in multi-user environments

The synchronization of DTMs is a mandatory feature to provide a better handling for the user within a multi-user environment (see Figure 143).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IDtm3.Init()
IDtm3.LoadData()
IDataset.CloseTransaction()
IDataset.CommitTransaction()
IDataset.StartTransaction()
IDataset.TransactionClosed()
IDataset.TransactionCommitted()
IDataset.TransactionStarted()

Figure 143 — Synchronizing DTMs in multi-user environments

The sequence diagram above describes an implementation example where a Frame Application provides a copy of the last committed DTMDataset (Dataset1*) for concurrently accessing DTM instances in multi-user scenarios. These instances cannot change the dataset at the same time (FA rejects StartTransaction()), but can read from the dataset last committed. How the Frame Application synchronizes the two instances of DtmDataset, is not in scope of FDT but specific to the Frame Application (shown as Frame Internal methods).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8.5 Execution of DTM Functions

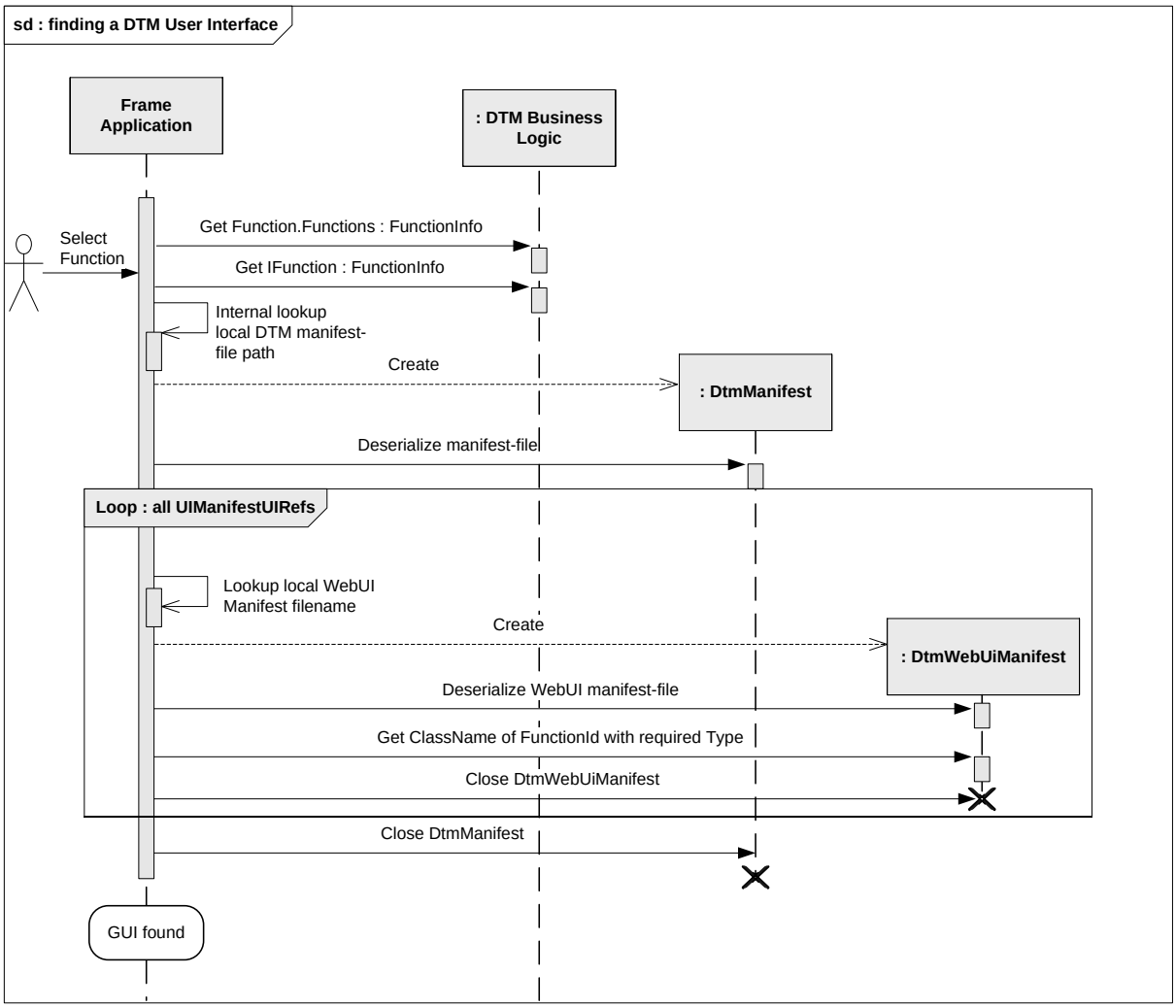
8.5.1 General

This specification defines different types of DTM WebUI (see 6.4).

The sequence diagrams in this section show the different handling of these user interface types.

8.5.2 Finding a DTM WebUI

The FunctionInfo property of IFunction interface provides access to user interfaces provided by a DTM. If a DTM provides user interfaces the FunctionInfo property contains a list of UiFunction objects. A UiFunction object represents a DTM WebUI function. The actual information about the object which implements this function is provided in a manifest file (see DtmManifest.UiManifestRefs description). The Frame Application shall use the property UiFunction.FunctionId to find the information in the manifest (see UiFunction description) (see Figure 144).



Used methods:

IDtm3.Functions
IFunction.Functioninfo

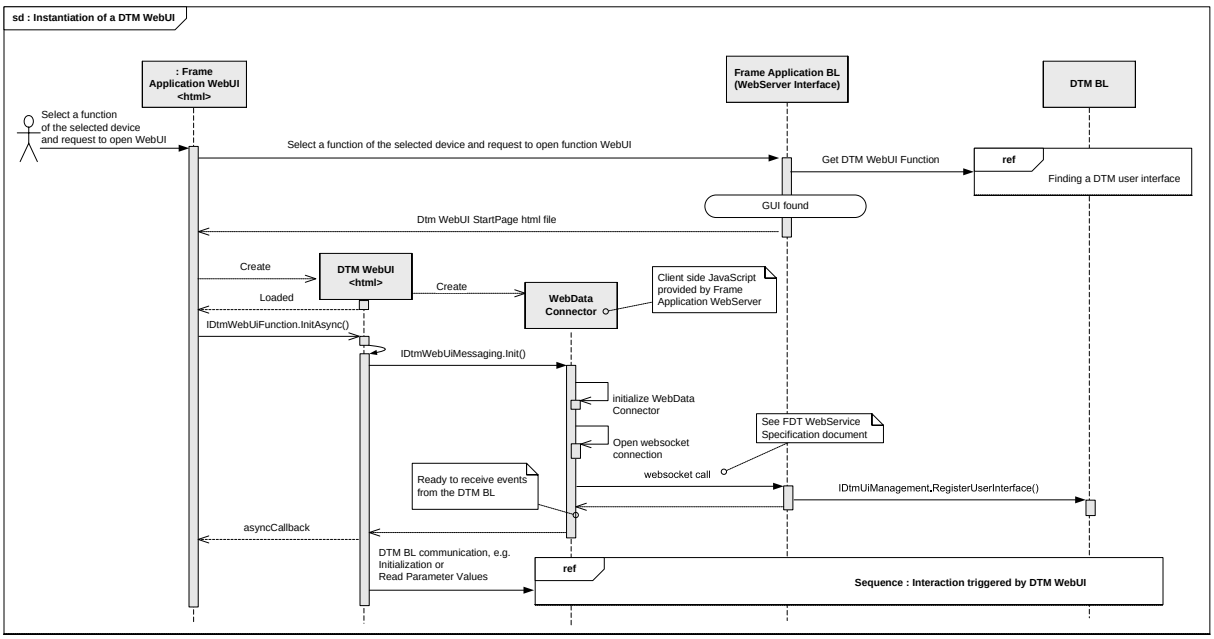
Figure 144 — Finding a DTM WebUI

8.5.3 Instantiation of a DTM WebUI

This sequence diagram outlines the opening of a DTM WebUI for a DTM function selected by the user (e.g. in a DTM-specific context menu) (see Figure 145). Different Frame Applications

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

may start the sequence differently, that is why the beginning of the sequence is described abstractly. The sequence may also be started by a different trigger (e.g. by a Frame Application function).



Used methods:

IFunction.FunctionInfo
IDtmWebUiFunction.initAsync()
IDtmWebUiMessaging.Init()
IDtmUiManagement.RegisterUserInterface()

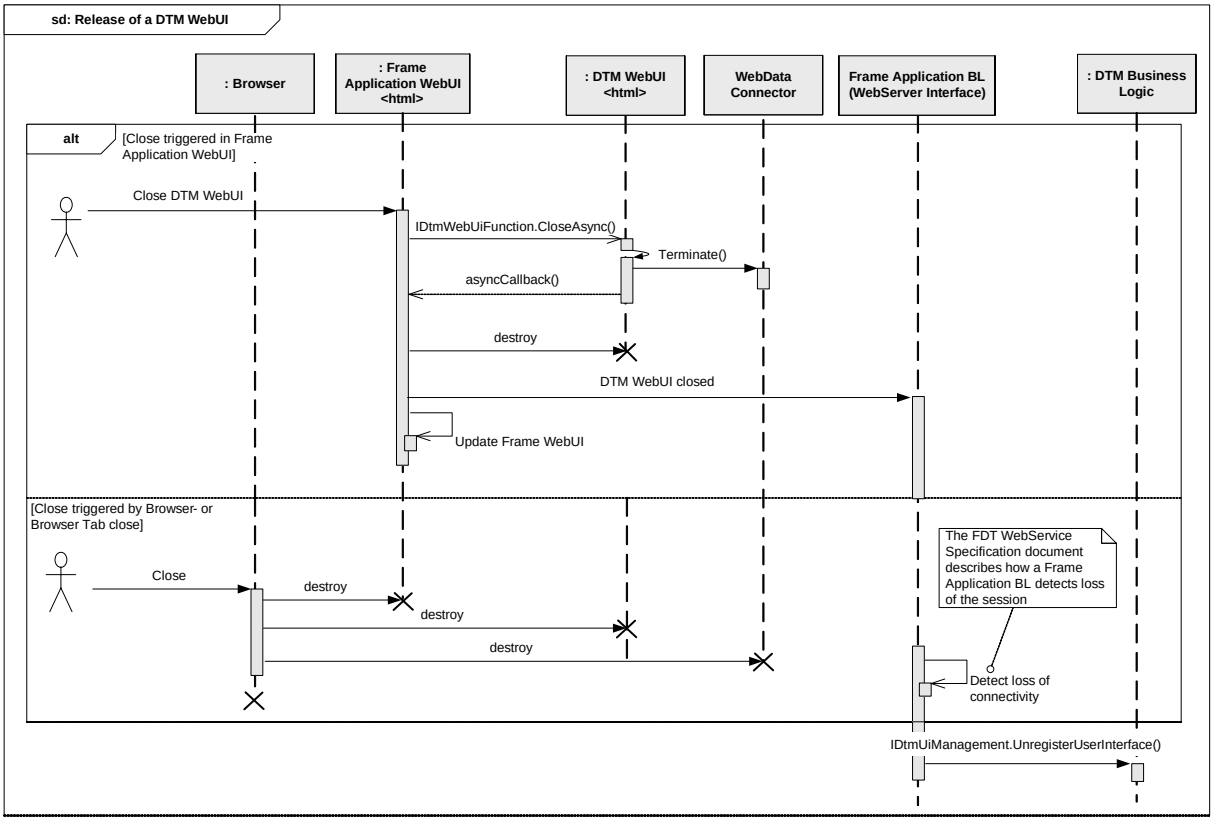
Figure 145 — Instantiation of a DTM WebUI

8.5.4 Release of a DTM WebUI

This sequence diagram outlines the closing of a DTM WebUI for a DTM function as a result of a request to the Frame Application or as result of an (unexpected) close of the browser (see Figure 146).

If the Frame Application releases a WebUI of a DTM, it has to prepare the release by sending a notification to the presentation object first. After receiving the call to closeAsync() the DTM WebUI may release its references to other components and may send DTM-specific releasing messages.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

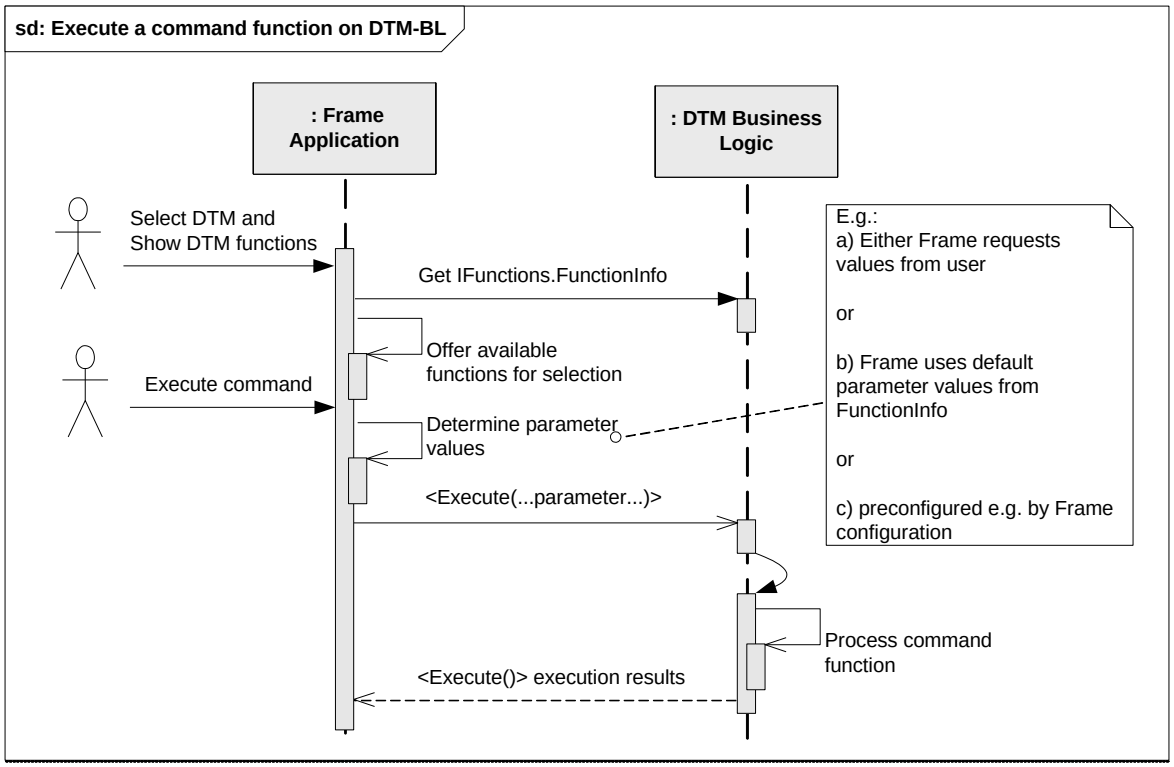
IDtmWebUiFunction.closeAsync()
IDtmUiManagement.UnregisterUserInterface()

Figure 146 — Release of a DTM WebUI

8.5.5 Execution of command functions

The execution of a command function on the DTM BL is started via the ICommandFunction interface. The execution of the command is triggered by BeginExecute() and EndExecute() (see Figure 147).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IFunction.FunctionInfo

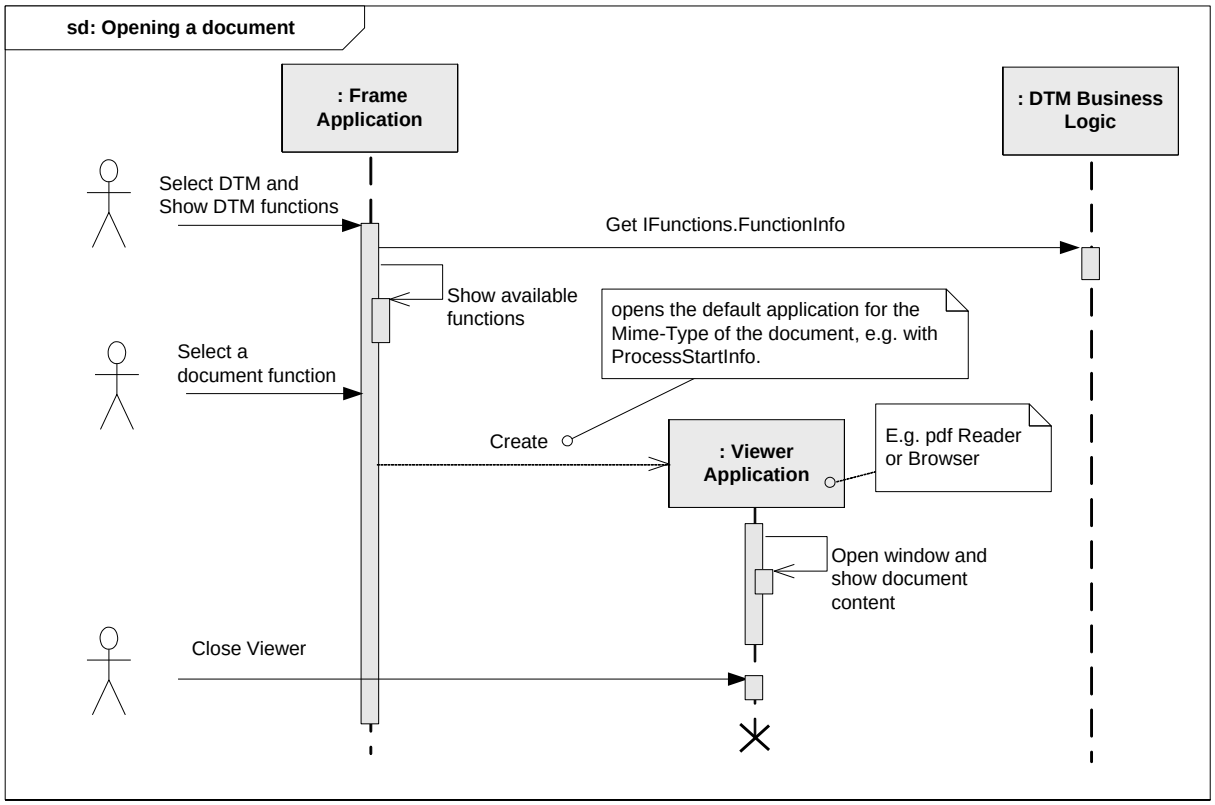
ICommandFunction.BeginExecute() / ICommandFunction.EndExecute()

Figure 147 — Execute a command function

8.5.6 Opening of documents

In order to open a document which is provided by a DTM, the Frame Application opens the default application for the MIME type of the document, for instance by calling the method ProcessStartInfo() (provided by the operating system) (see Figure 148).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



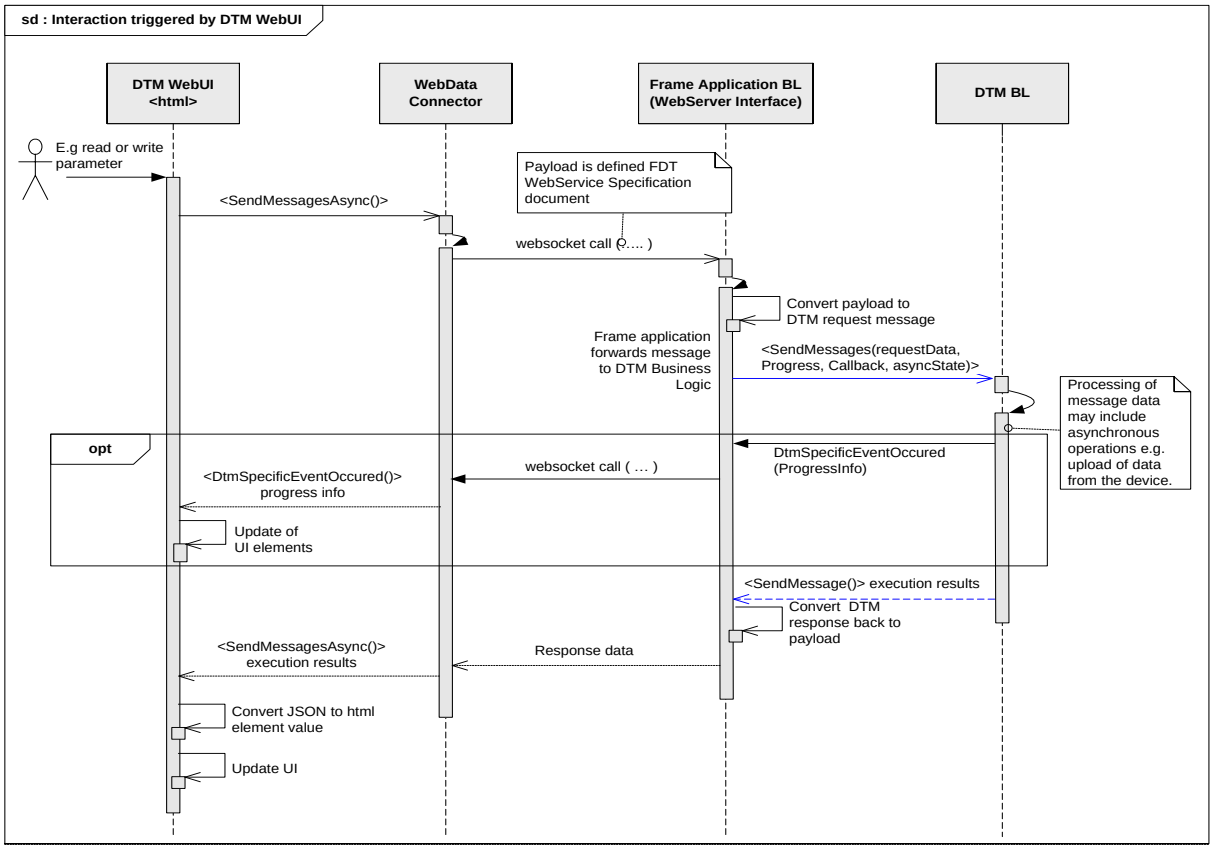
Used methods:
IFunction.FunctionInfo

Figure 148 — Opening a document

8.5.7 Interaction between DTM WebUI and DTM Business Logic

This sequence diagram outlines the interaction of a DTM WebUI with its Business Logic over the messaging interface provided by the Frame Application (see Figure 149).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IDtmWebUiMessaging.SendMessageAsync()
IDtmWebUiMessageEvents.DtmSpecificEventOccured()

Figure 149 — Interaction triggered by the DTM WebUI

Note: The callback SendMessageCallback are provided as argument of SendMessageAsync

In this scenario the DTM WebUI requests data from the DTM Business Logic (e.g. to read measured values from the device) by sending a DTM-specific request message(s) derived from the abstract DtmRequestMessage class.

The IDtmUiMessaging interface is implemented by the DTM Business Logic and the Frame Application. The reference to the Frame Application implemented interface shall be passed to a DTM WebUI with the IDtmWebUiFunction.initAsync() call. The Frame Application shall forward the messages between the DTM WebUI and the DTM Business Logic.

The DTM Business Logic evaluates the requests and creates corresponding response message(s) derived from the abstract DtmResponseMessage class. The response messages contain the requested data and are sent back by calling the Progress and Callback methods.

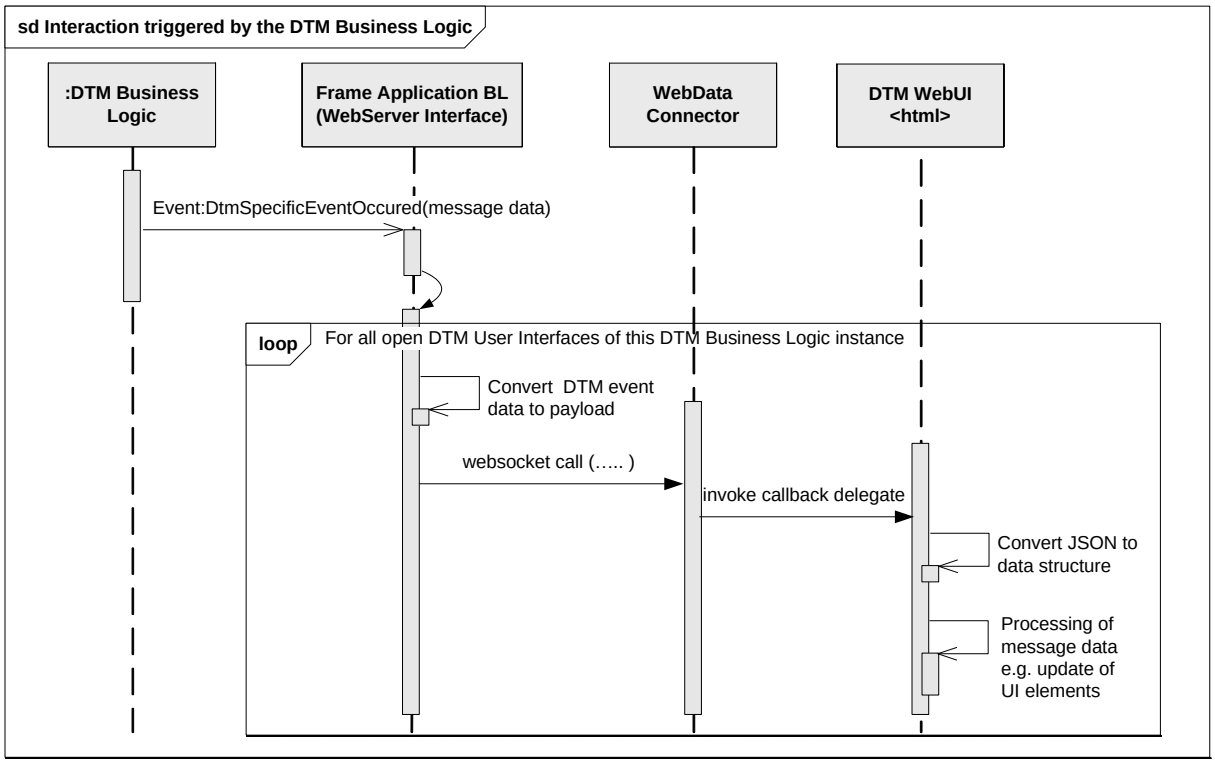
More detailed information can be found in descriptions of:

- IDtmUiMessaging
- DtmRequestMessage
- DtmResponseMessage

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8.5.8 Interaction between DTM Business Logic and DTM WebUI

In 8.5.8 it is described, how a DTM BL sends an event to all its instantiated DTM WebUIs. This sequence diagram outlines how the event also is distributed from a DTM BL to its currently instantiated DTM WebUI (see Figure 150).



Used methods:

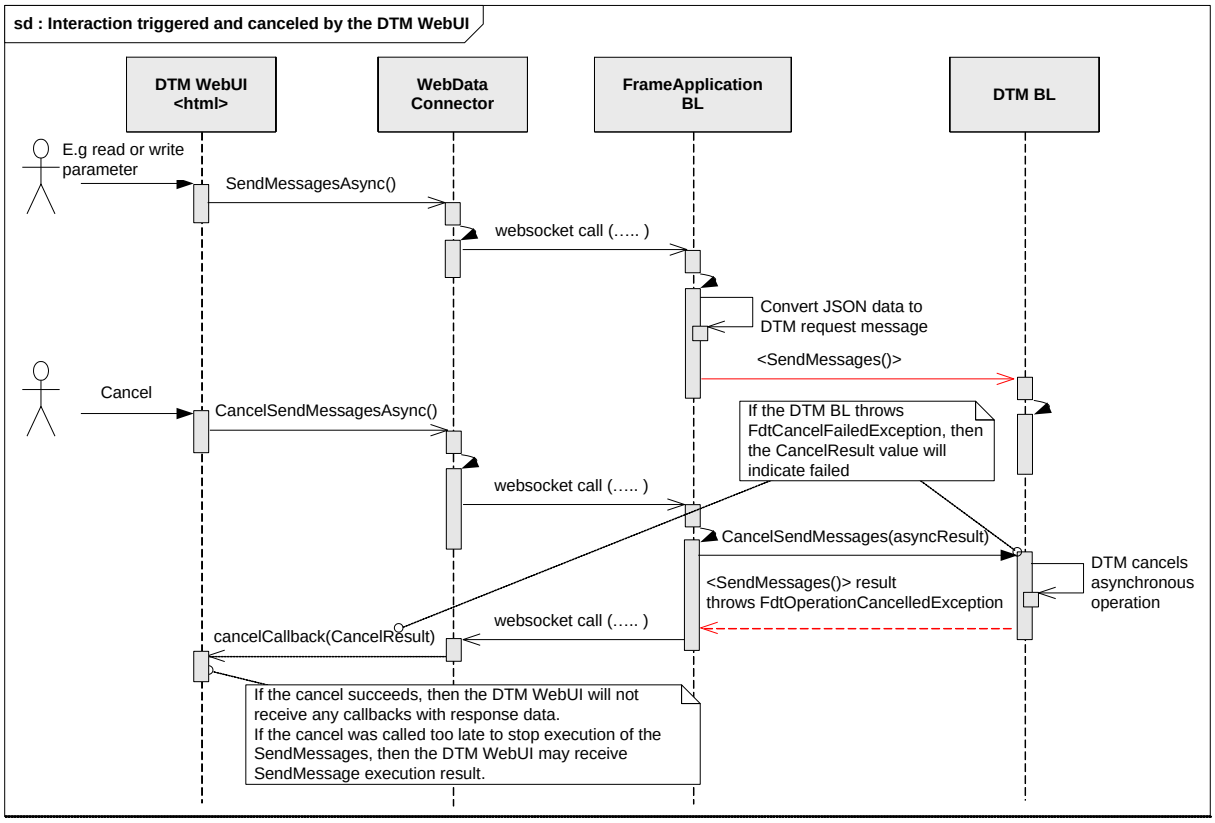
Event IDtmUiMessaging.DtmSpecificEventOccured()

Figure 150 — Interaction triggered between DTM BL and DTM WebUI

8.5.9 Interaction between DTM WebUI and DTM Business Logic with Cancel

This sequence diagram outlines the canceling of a pending user interface message on request of the user (see Figure 151).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IDtmWebUiMessaging.SendMessagesAsync()
IDtmUiMessaging.BeginSendMessages()
IDtmUiMessaging.EndSendMessages()
IDtmWebUiMessaging.CancelSendMessagesAsync()
IDtmUiMessaging.CancelSendMessages()

Used exceptions:

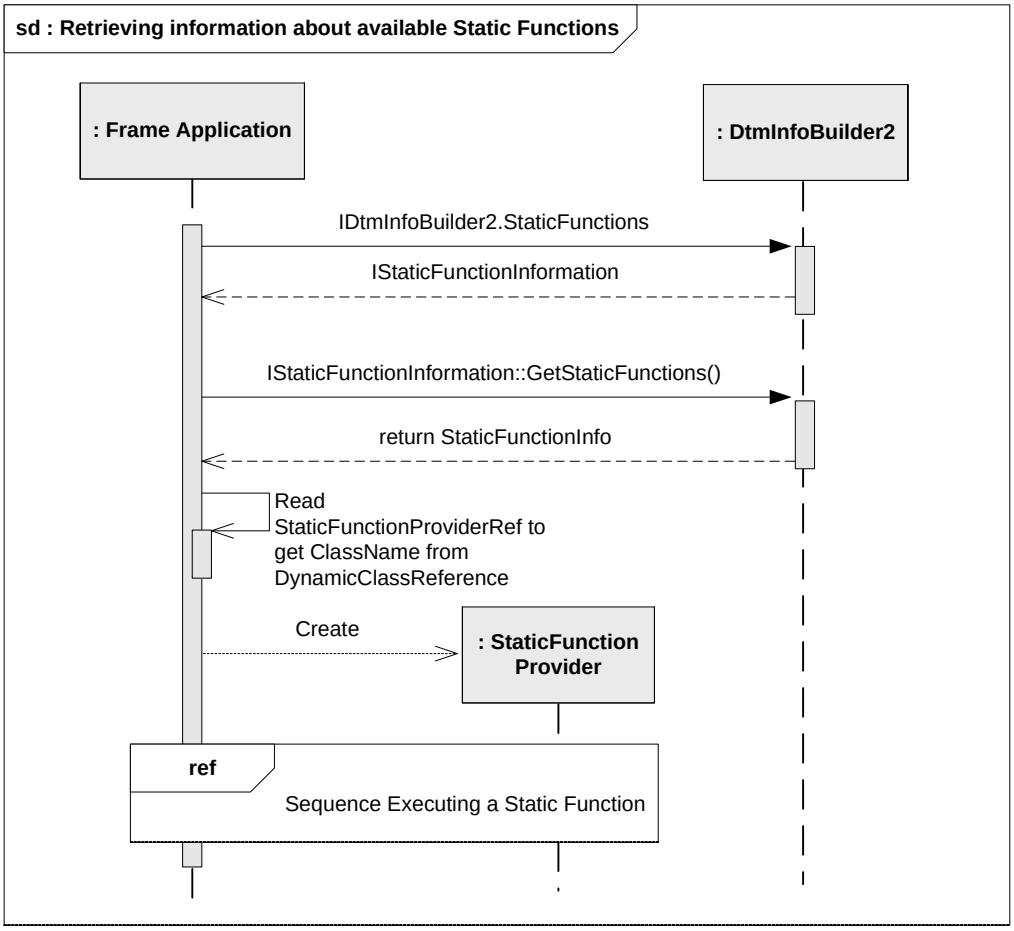
Fdt.FdtOperationCancelledException

Figure 151 — Interaction triggered and canceled by the DTM WebUI

8.5.10 Retrieving information about available Static Functions

In order to use a Static Function for a specific device, the Frame Application retrieves the information about available Static Functions from the corresponding DtmInfoBuilder instance (see Figure 152).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:
IDtmInfoBuilder2.StaticFunctions
IStaticFunctionInformation.GetStaticFunctions()

Figure 152 — Retrieving information about available Static Functions

Figure 153 shows the example for StaticFunctionInfo data, which was retrieved from a DTM.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

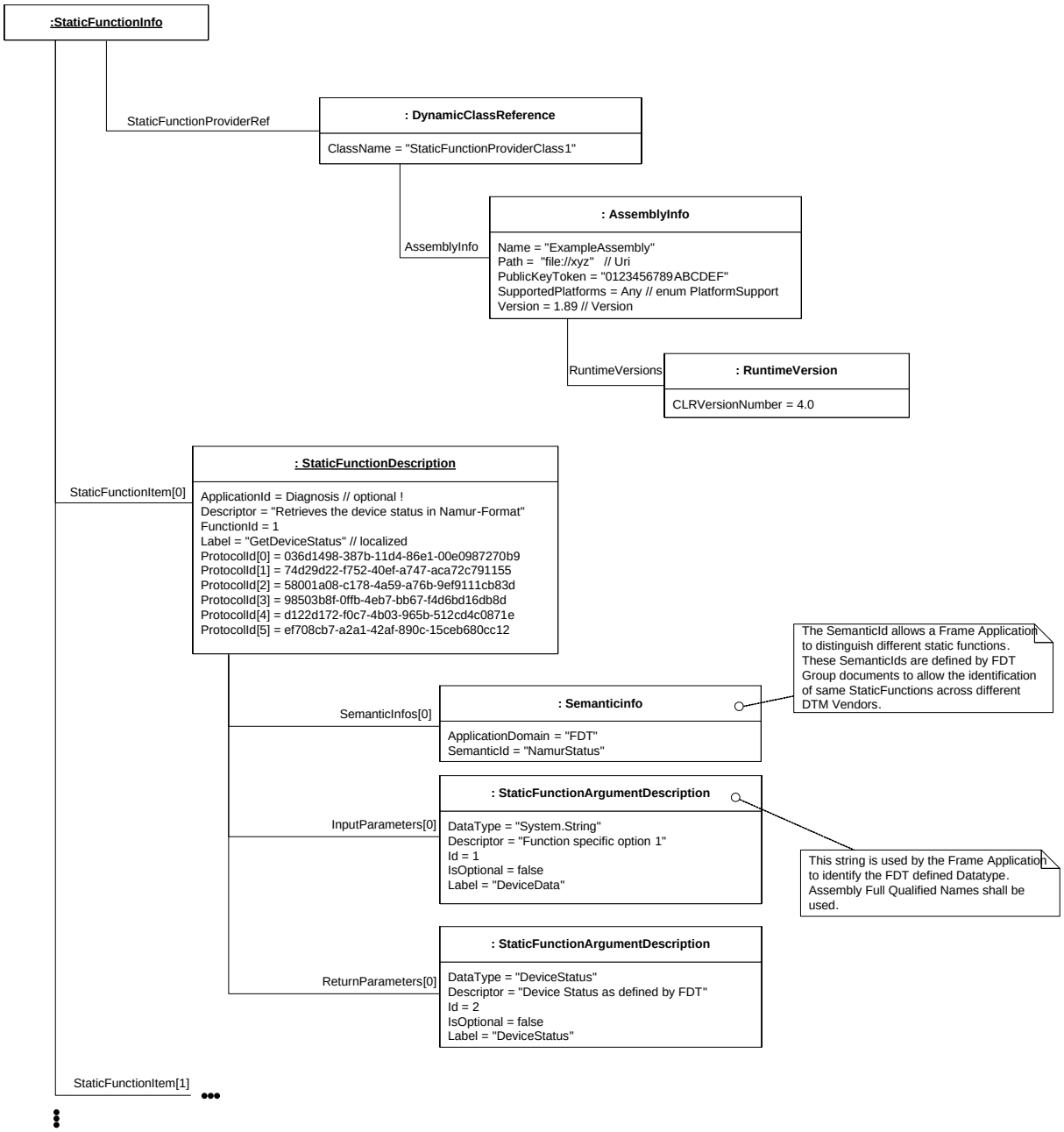
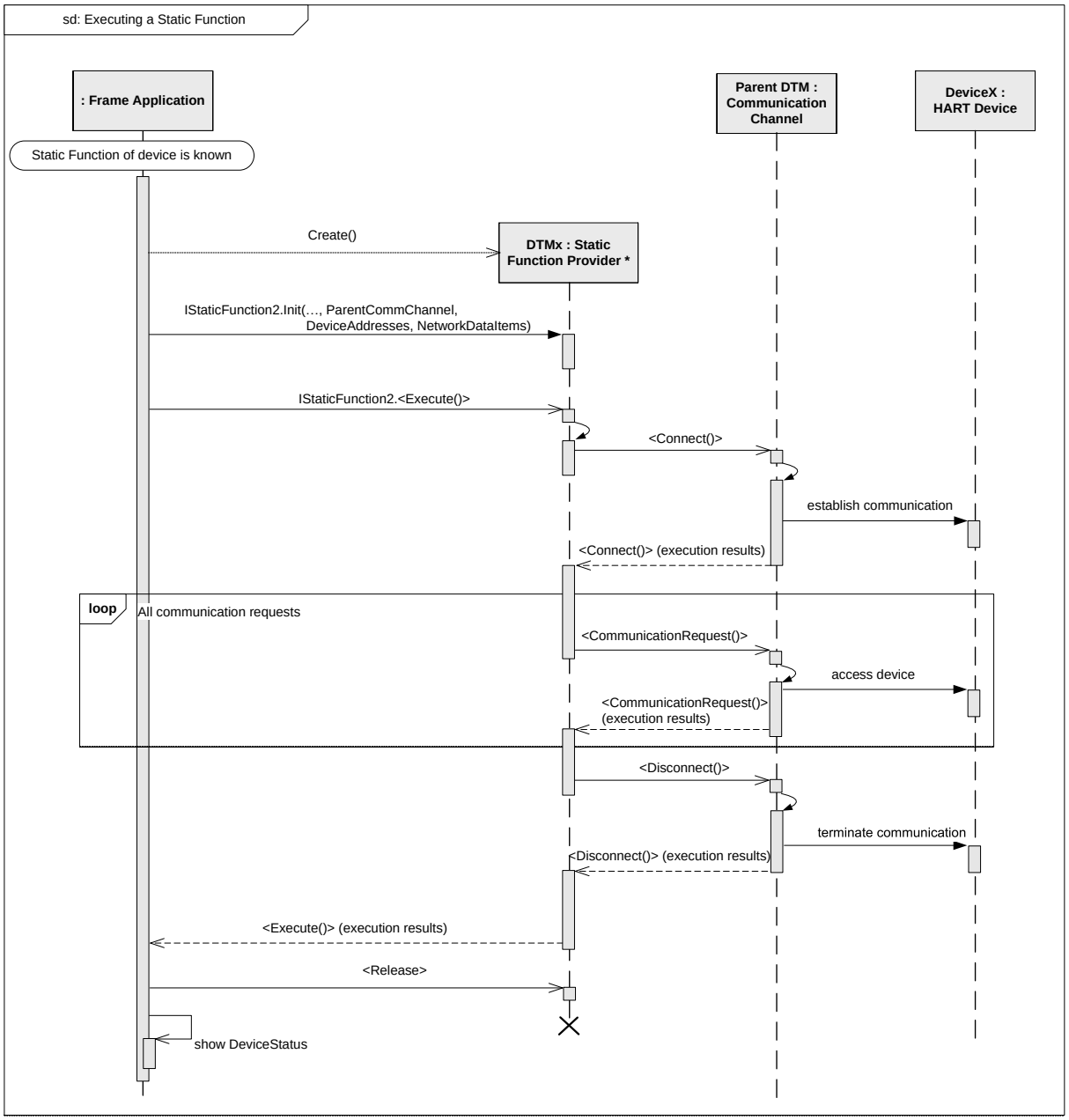


Figure 153 — Example: Information about available Static Functions

8.5.11 Executing a Static Function

After retrieving the information regarding the available Static Functions, the Frame Application may provide triggers for execution of the Static Functions to the user (e.g. in a menu) or may use internal triggers to execute a Static Function (see Figure 154).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

ICommunication.BeginConnect()
ICommunication.EndConnect()
ICommunication.BeginCommunicationRequest()
ICommunication.EndCommunicationRequest()
ICommunication.BeginDisconnect()
ICommunication.EndDisconnect()
IStaticFunction2.Init()
IStaticFunction2.BeginExecute()
IStaticFunction2.EndExecute()
IStaticFunction2.BeginRelease()
IStaticFunction2.EndRelease()

Figure 154 — Executing a Static Function

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8.6 DTM communication

8.6.1 General

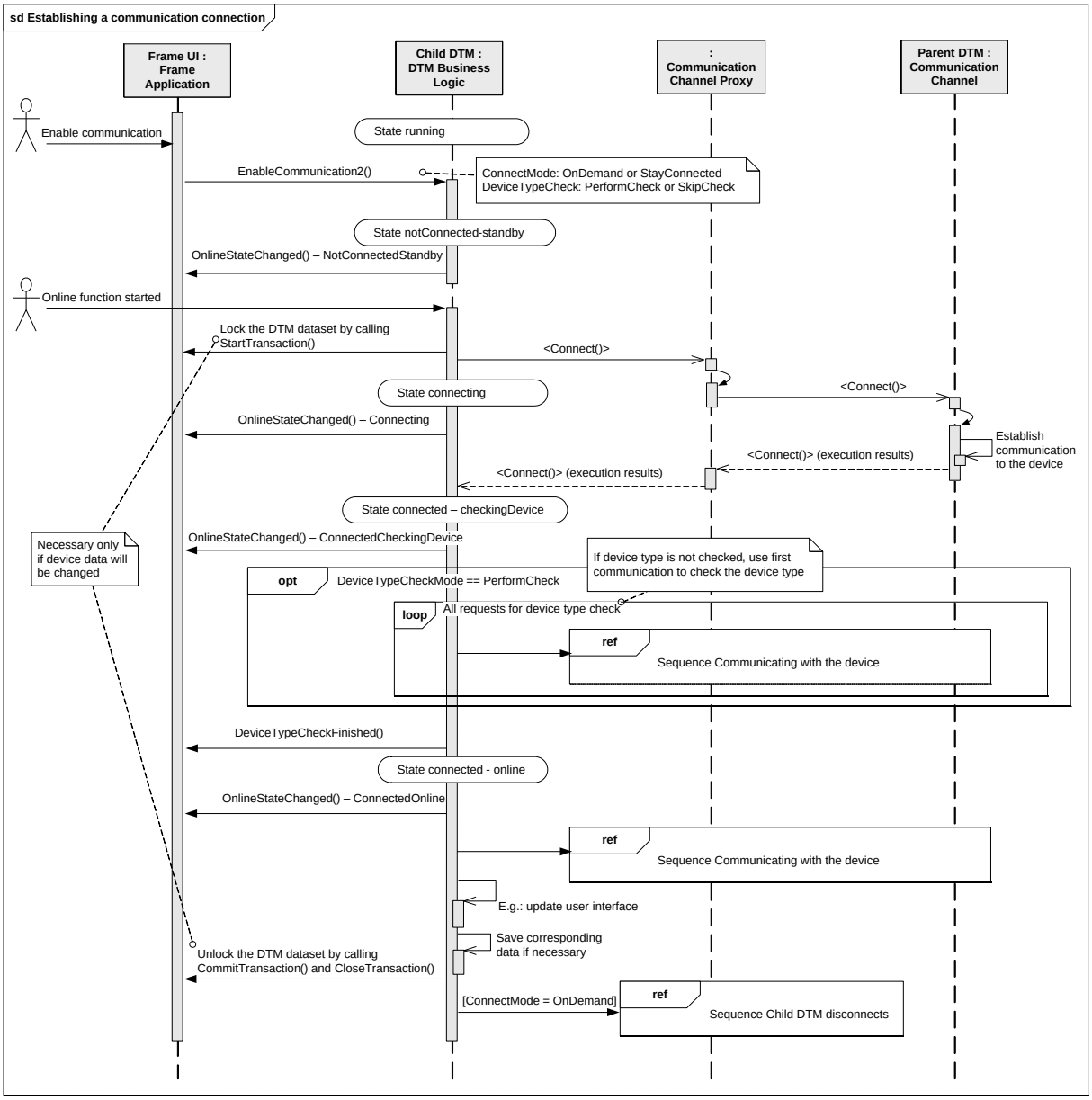
Each communication connection for a DTM is established as a point-to-point connection. This subclause describes the field communication related workflows. Communication Channels implement the interface ICommunicationChannel. The interface ICommunication can be accessed by the ICommunicationChannel property “Communication” and provides services for fieldbus connection and communication requests.

In order to ensure that only the Frame Application can modify the sub-topology of a Communication Channel, DTMs cannot directly access the ICommunicationChannel interface of the parent channel. Instead the Frame Application provides a proxy for the channel implementing the ICommunicationChannelProxy interface. This proxy provides access to all Communication Channel interfaces except the interface for sub-topology management. The proxy redirects all method calls to the Communication Channel.

8.6.2 Establishing a communication connection

The following sequence diagram describes the calling sequence of a DTM when connecting to the field device (see Figure 155).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IDtm3.EnableCommunication2()
ICommunication.BeginConnect()
ICommunication.EndConnect()
Event IDtm3.OnlineStateChanged()

Figure 155 — Establishing a communication connection

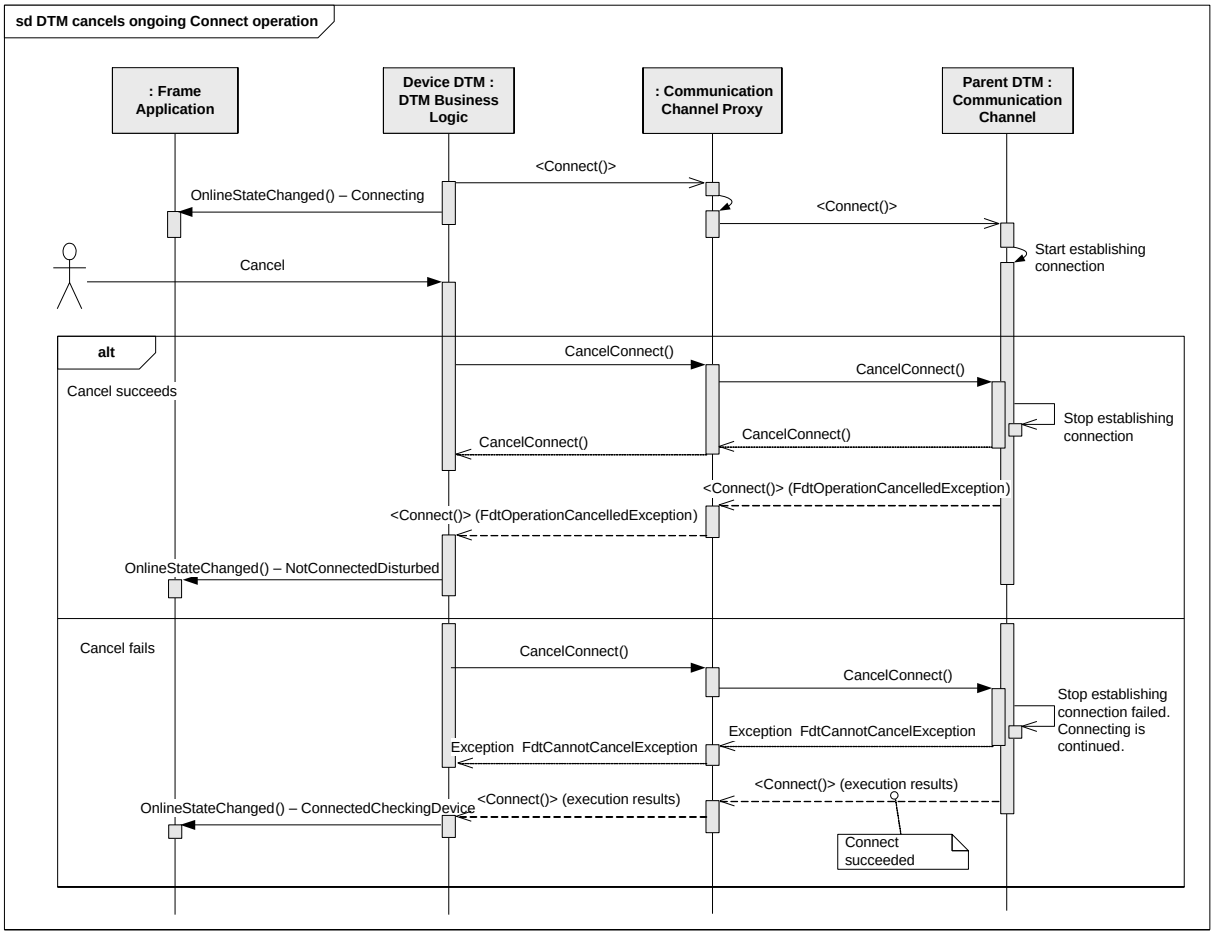
Note: Online functions which affect the device data or the instance data require a locked DtmDataset. Prolonged locks shall be avoided to support multi-user Frame Applications. Thus, it is a DTM-specific decision to balance between the granularity of online operations and the drawback of prolonged locks.

8.6.3 Cancel establishment of communication connection

This workflow describes how an ongoing connect request is canceled (see Figure 156).

If the connect action cannot be canceled, the call of the method CancelConnect() throws an exception.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

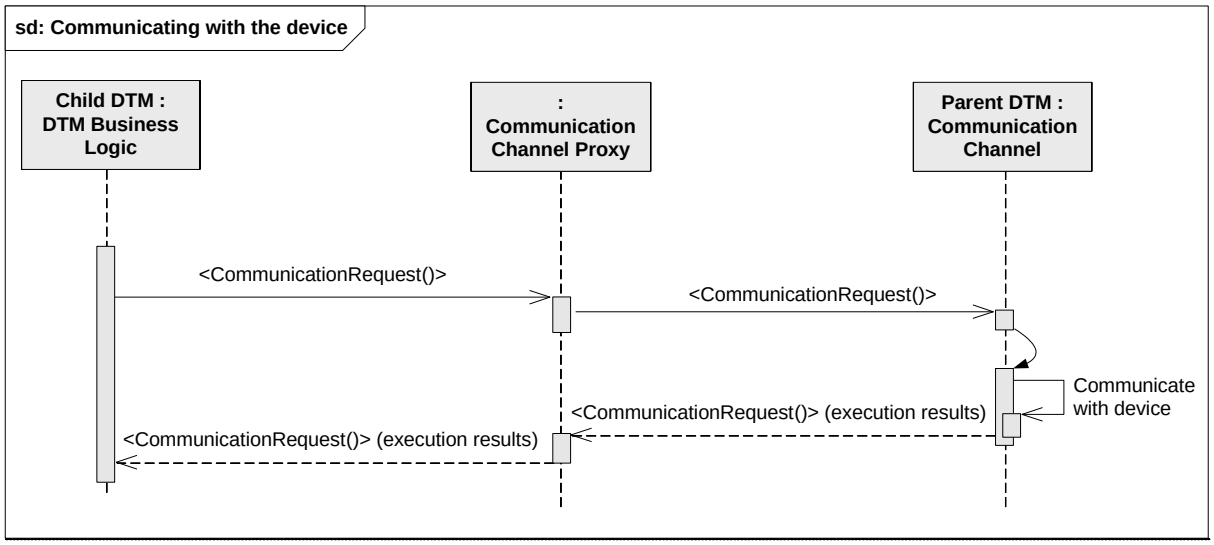
ICommunication.BeginConnect()
ICommunication.CancelConnect()
ICommunication.EndConnect()
Event IDtm3.OnlineStateChanged()

Figure 156 — DTM cancels ongoing Connect operation

8.6.4 Communicating with the device

The following sequence diagram explains the Device DTM communication with the device using a Communication Channel (see Figure 157).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:
ICommunication.BeginCommunicationRequest()
ICommunication.EndCommunicationRequest()

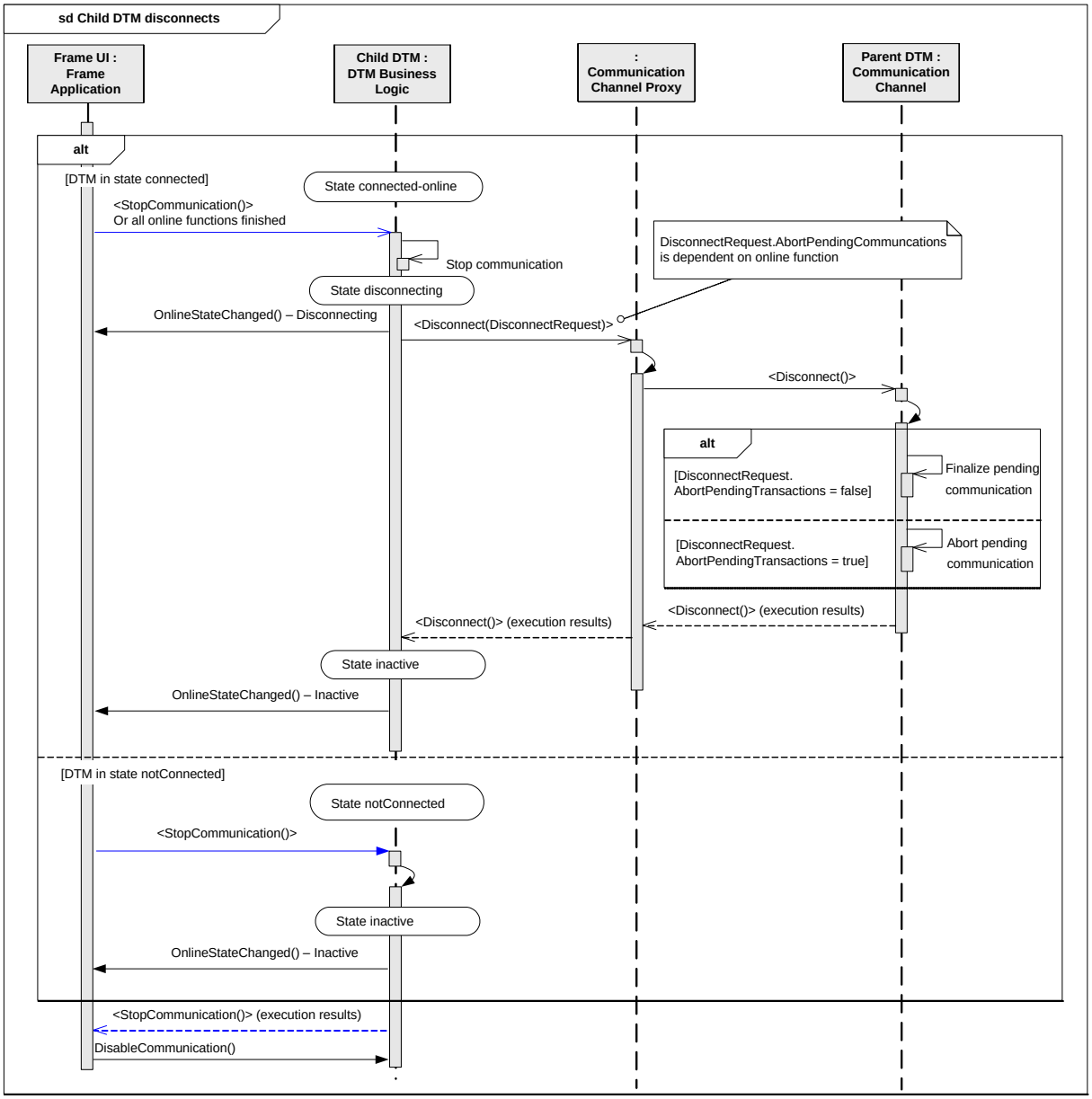
Figure 157 — Communicating with the device

8.6.5 Frame Application or Child DTM disconnect a device

Figure 158 shows the flow of messages, when a Frame Application sets a DTM offline.

It depends on the Child DTM, whether pending communication requests are finalized or aborted.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

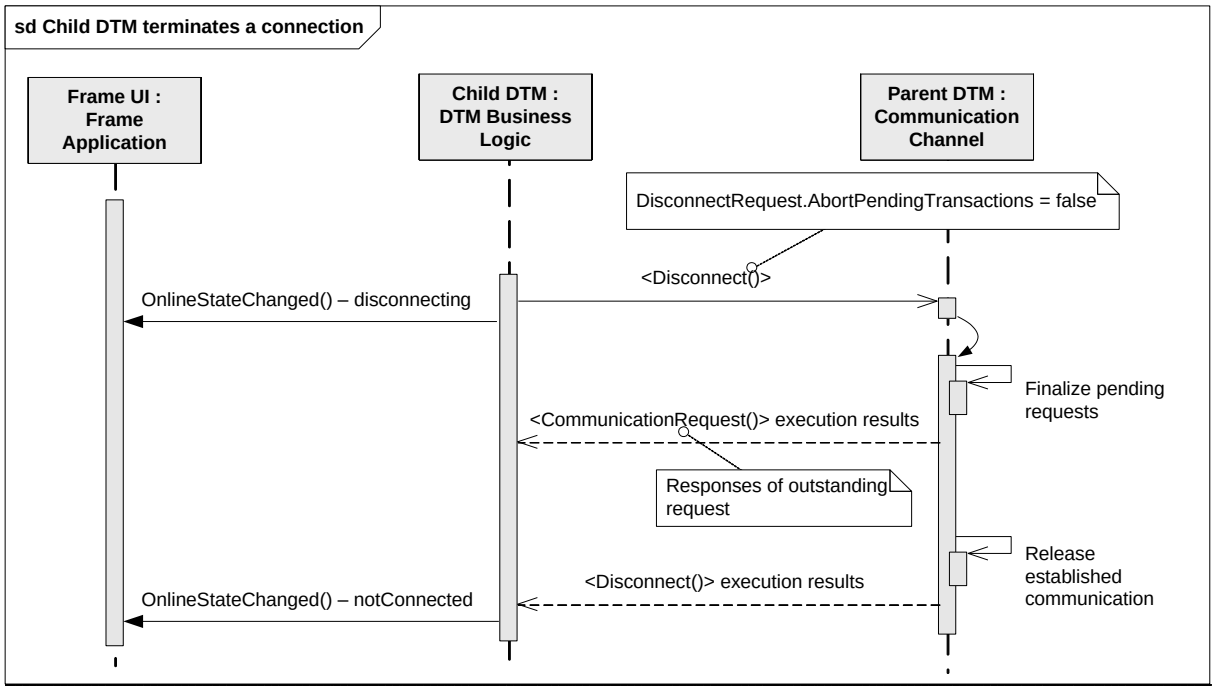
IDtm3.BeginStopCommunication()
IDtm3.EndStopCommunication()
Event IDtm3.OnlineStateChanged()
ICommunication.BeginDisconnect()
ICommunication.EndDisconnect()
IDtm3.DisableCommunication()

Figure 158 — Child DTM disconnects

8.6.6 Terminating a communication connection

The following sequence diagram (see Figure 159) shows how a communication connection is terminated by a Child DTM.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

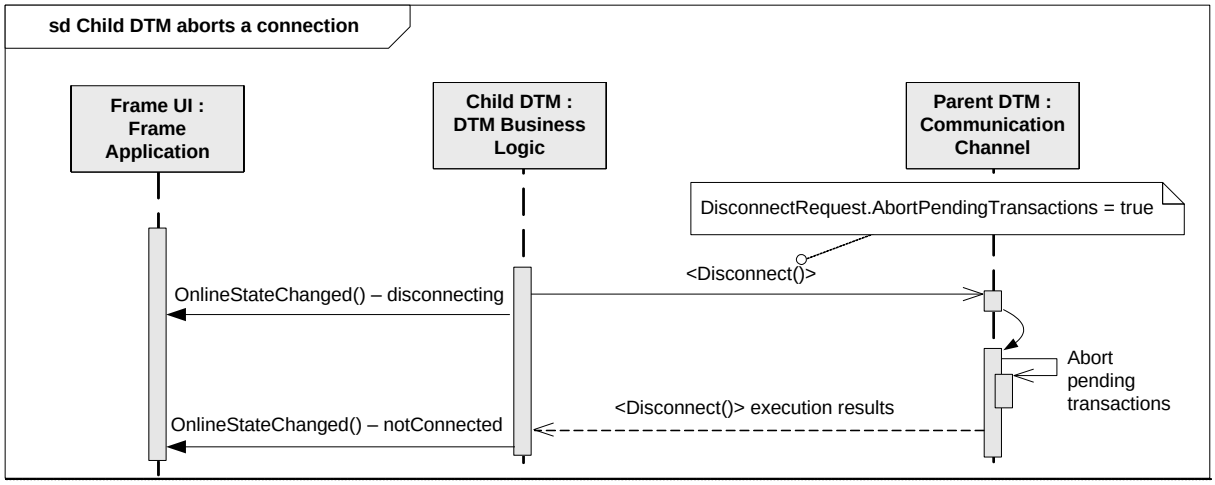
ICommunication.BeginDisconnect()
ICommunication.EndDisconnect()

Figure 159 — Child DTM terminates a connection

In case of a <Disconnect()> with argument AbortPendingTransactions set to 'false', the Communication Channel executes all outstanding communication requests. The Child DTM will receive responses with the respective communication data.

8.6.7 DTM aborts communication connection

This sequence describes the abort of a communication link to a device without expecting any further communication response (see Figure 160).



Used methods:

ICommunication.BeginDisconnect()
ICommunication.EndDisconnect()
Event OnlineStateChanged()

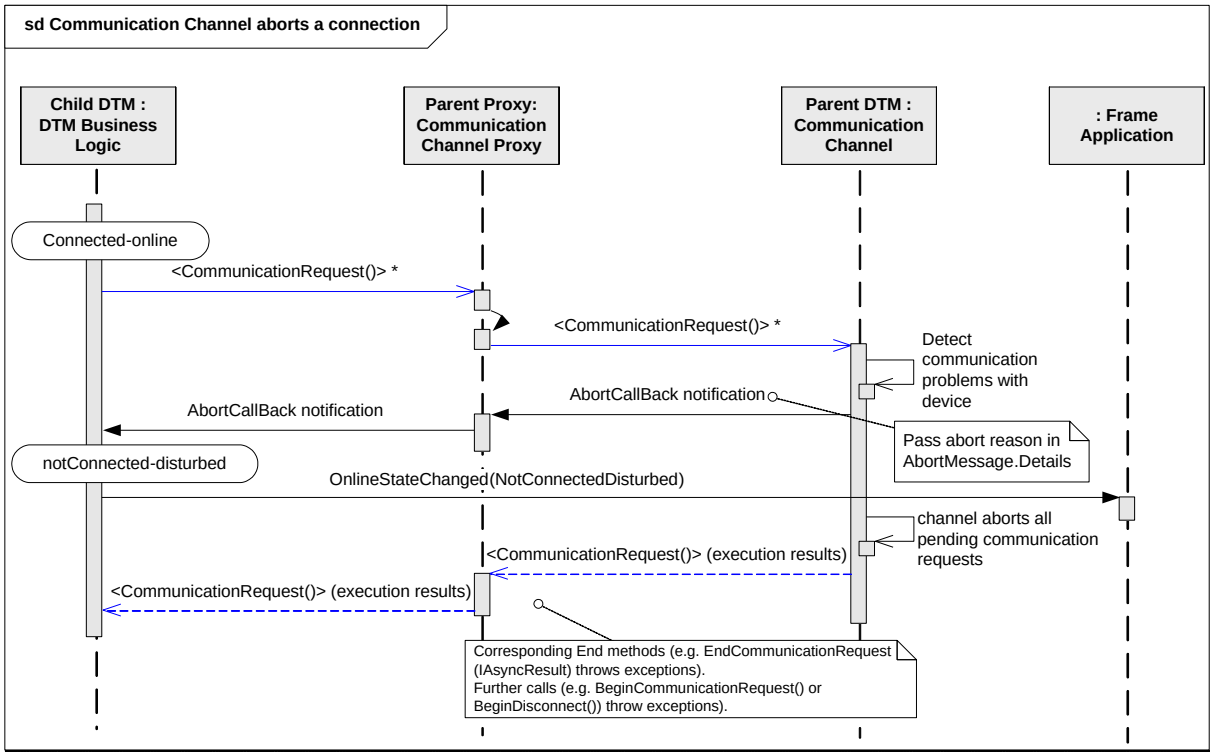
Figure 160 — Child DTM aborts a connection

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

In case of a <Disconnect()> with argument AbortPendingTransactions set to 'true', the Communication Channel cancels all outstanding communication requests. The Child DTM will receive responses with the information that the communication request was not executed.

8.6.8 Communication Channel aborts communication connection

This sequence (Figure 161) describes how a Communication Channel aborts an active communication connection to a device.



Used methods:

ICommunication.BeginCommunicationRequest()
ICommunication.EndCommunicationRequest()

Used events:

AbortCallBack delegate
Event OnlineStateChanged()

Figure 161 — Communication Channel aborts a connection

8.7 Nested communication

8.7.1 General

This subclause describes communication related to devices with gateway functionality like remote I/O devices. Nested communication is used to establish the connection to a device on a sub-system.

The example in Figure 162 shows how a Device DTM communicates to a field device which is connected to a Communication Channel of a Gateway DTM, which in turn is connected to a Communication Channel of a Communication DTM. Since the Device DTM represents a HART field device, it is communicating based on HART protocol. The Gateway DTM represents a PROFIBUS/HART gateway (e.g. a Remote IO), that is why the Gateway DTM is communicating to the gateway based on PROFIBUS protocol. The Communication DTM represents the fieldbus interface, the DTM accesses the driver of the fieldbus interface. (This example will also be used in other subsections of 8.7).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

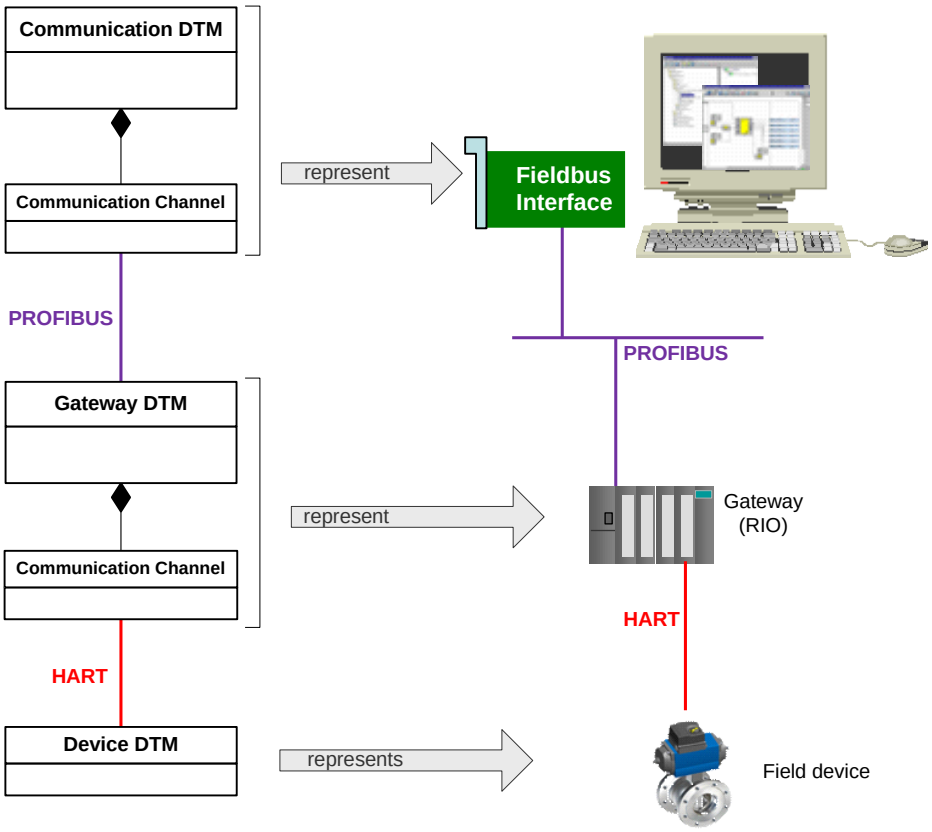


Figure 162 — Example: Nested communication behavior

Gateway DTMs (e.g. for a remote I/O) have to provide one or more Communication Channels that are used by other DTMs.

The requirement is that a DTM shall not need to know anything about the communication hierarchy. Nevertheless, the structure of the sub-system is well known to Frame Application and by the Gateway DTM.

The functionality for address management is always provided by the Frame Application or by the Parent DTM. Therefore each DTM has to allow setting the network parameters like 'tag' and 'BusInformation' according to the communication protocol (see also: INetworkData, NetworkDataInfo, AddressInfo).

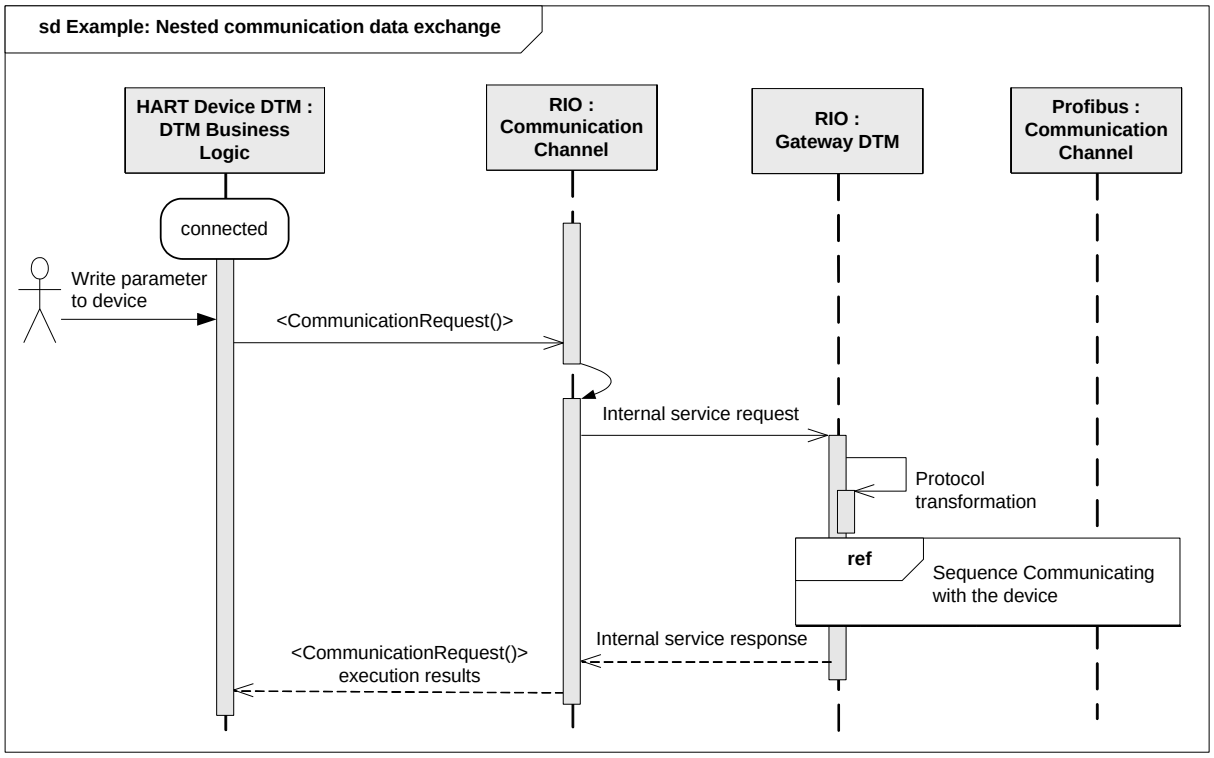
8.7.2 Communication request for a nested connection

The sequence in Figure 163 shows an example how the HART Device DTM from Figure 162 communicates to its field device. The internal communication of the Gateway DTM and the communication to the PROFIBUS Communication Channel are transparent to the Device DTM.

To write a parameter to the device, the HART Child DTM calls BeginCommunicationRequest() at the Communication Channel. The HART request is wrapped in the remote I/O channel to a PROFIBUS communication message sent to the parent PROFIBUS Communication DTM.

The corresponding response is provided by the PROFIBUS Parent Communication DTM. After extracting the HART response, the remote I/O Gateway DTM sends the response via the Communication Channel to the HART Device DTM.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:
ICommunication.BeginCommunicationRequest()
ICommunication.EndCommunicationRequest()

Figure 163 — Example: Nested communication data exchange

8.7.3 Propagation of errors for a nested connection

In a nested communication hierarchy there may be several sources for communication errors. Considering the example from Figure 162, possible sources are:

- Field device responds to communication requests with errors (e.g. wire break)
- Gateway device (RIO) has communication problems (e.g. field device does not communicate) and responds with errors
- Gateway device (RIO) has internal problems (e.g. module failure) and responds with errors
- Fieldbus interface has communication problems (e.g. gateway device does not communicate) and responds with errors
- Fieldbus interface has internal problems (e.g. not configured) and responds with errors

If errors occur during execution of communication requests, the errors have to be propagated back to the origin of the communication request (see 4.11.1.2).

In order to support fixing the problem, the DTM representing the component where the error occurred shall inform the user about the source of error within the CommunicationError. This helps to avoid a situation, where the user receives several error reports (e.g. if gateway device detects, that the field device does not respond, the Gateway DTM will produce a user message and the Device DTM will produce a user message).

If an intermediate component receives such a communication error, it shall in turn generate a communication error, provide own additional information and shall pack the received communication error as inner communication error into the generated communication error (similar to exceptions/inner exceptions).

If the origin of communication request receives such a communication error, it shall inform the user with a user message that includes the information from the inner communication errors.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8.8 Topology planning

8.8.1 General

The Frame Application is responsible to generate and manage the topology.

The requirement is that a DTM shall not need to know anything about the communication hierarchy. Nevertheless, the structure of the whole topology is well known to a Frame Application.

Chapter 8.8.2 describes how a Frame Application creates a topology. The example in chapter 8.8.5 shows how a Gateway DTM generates a sub-topology.

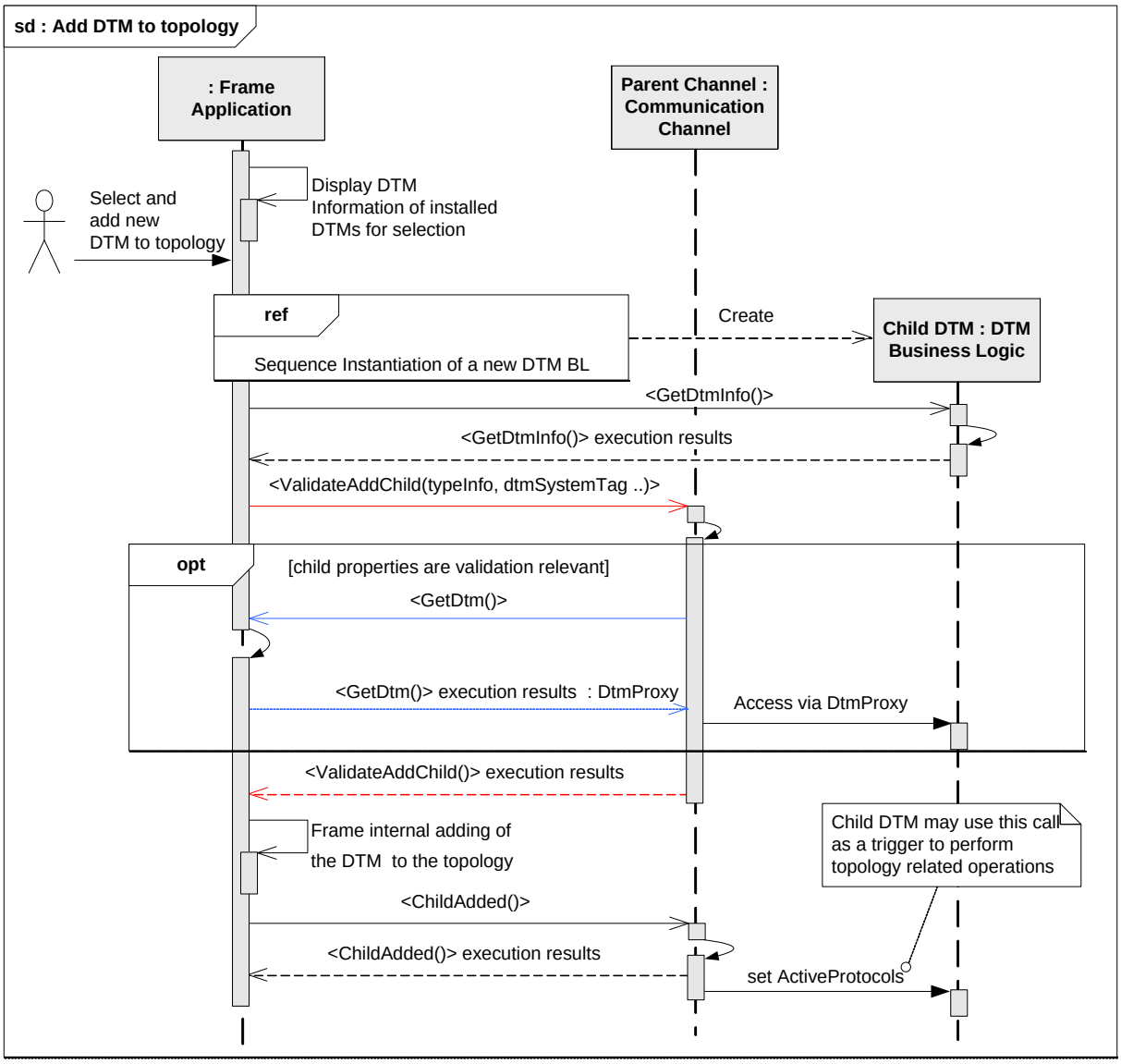
8.8.2 Adding a DTM to the topology

If a DTM is added to the topology, a validation is executed whether the DTM fits into the topology.

This validation is executed by the Communication Channel to which the new Child DTM is added. During the validation the Communication Channel may access the Child DTM.

Since the Child DTM at this point is not yet part of the topology, the Child DTM does not yet have a Parent DTM and may not access the Parent DTM (see Figure 164).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

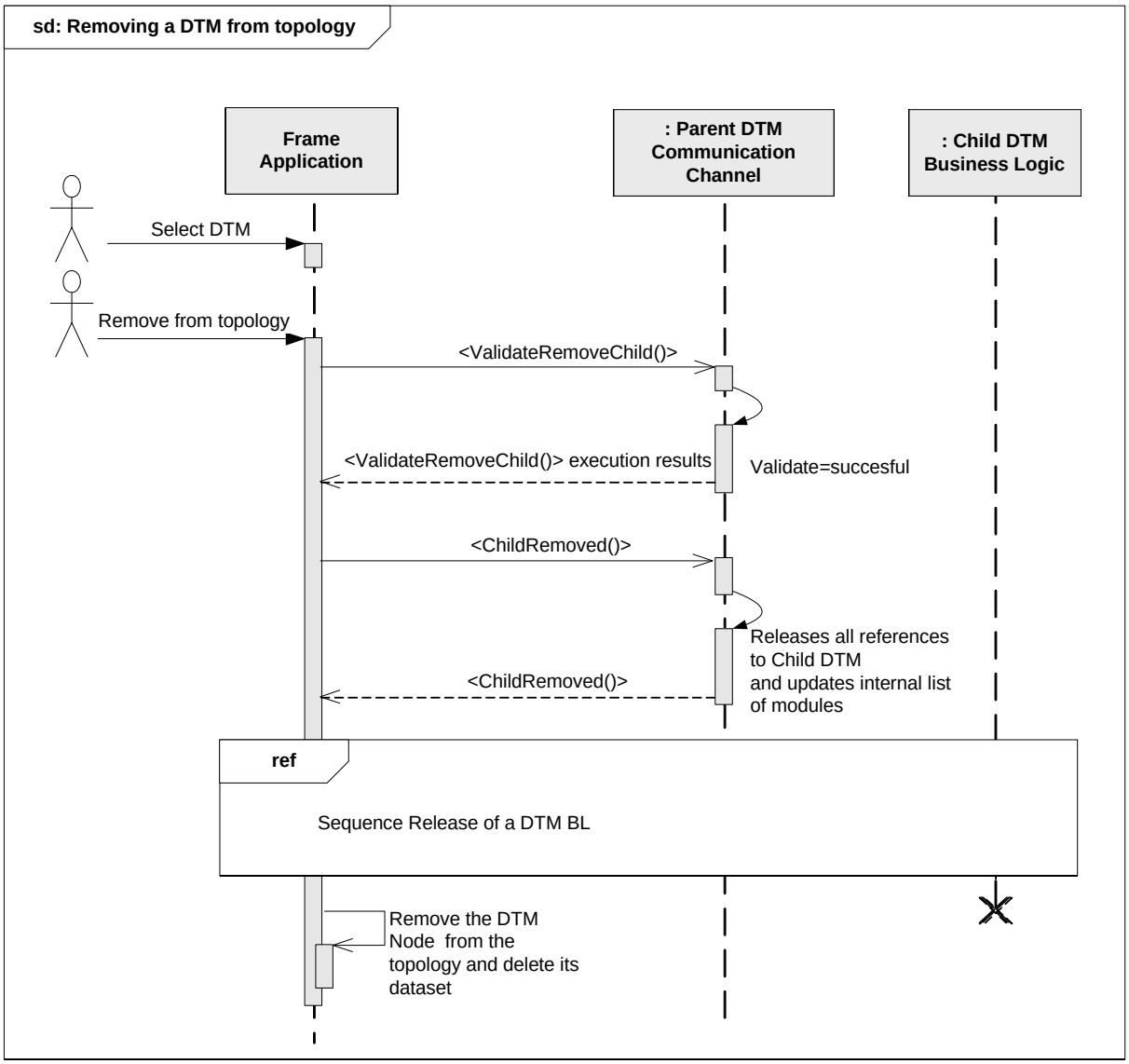
IDtmInformation.BeginGetInfo()
IDtmInformation.EndGetInfo()
ISubTopology.BeginValidateAddChild() / ISubTopology.EndValidateAddChild()
ISubTopology.BeginChildAdded() / ISubTopology.EndChildAdded()
INetworkData.ActiveProtocols

Figure 164 — Add DTM to topology

8.8.3 Removing a DTM from topology

Figure 165 shows how a DTM is removed from a topology. Before the Frame Application removes the device node and its dataset from the topology, the Parent DTM shall validate the removal, release all references to the Child DTM and update the internal list of modules.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

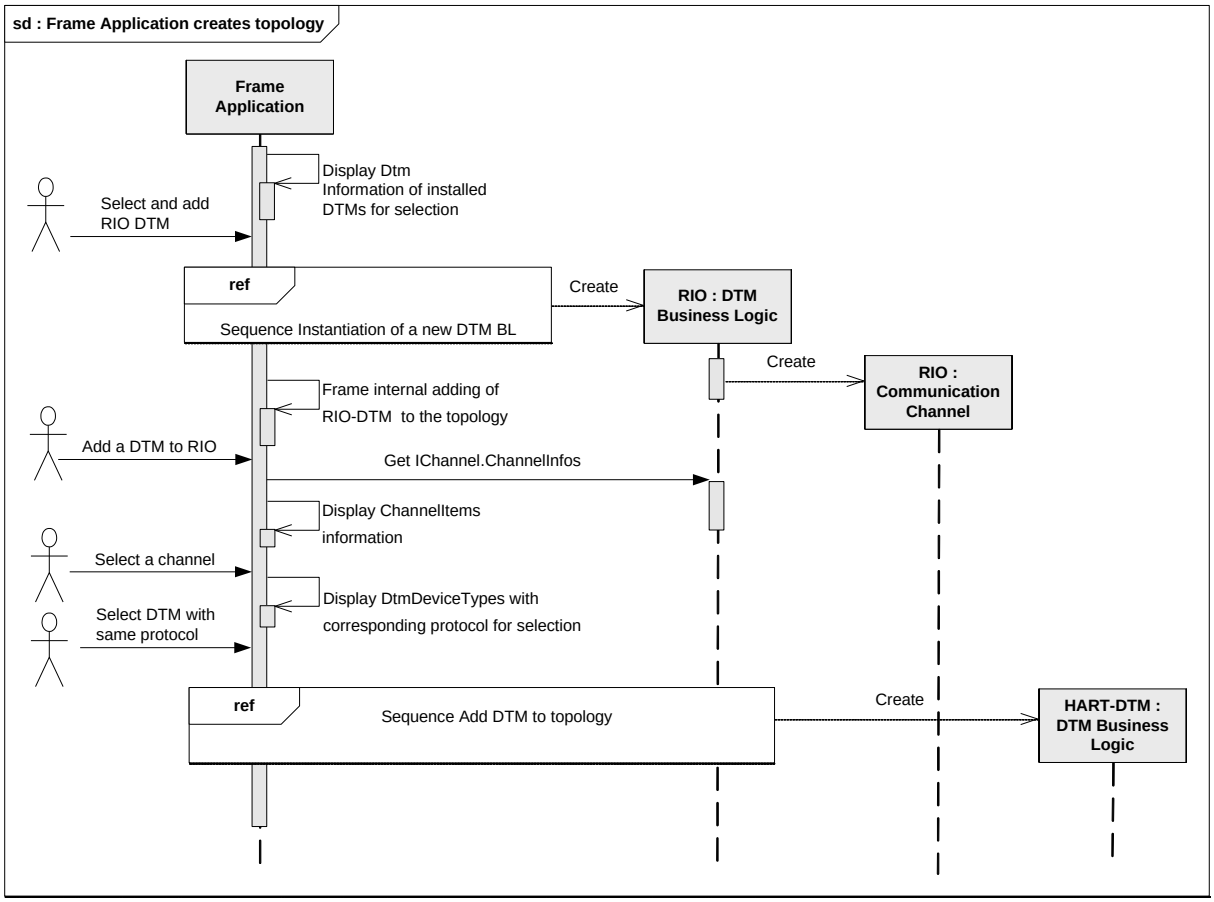
ISubTopology.BeginValidateRemoveChild()
ISubTopology.EndValidateRemoveChild()
ISubTopology.BeginChildRemoved()
ISubTopology.EndChildRemoved()

Figure 165 — Removing a DTM from topology

8.8.4 Frame Application creates topology

The following sequence diagram (Figure 166) shows an example workflow how a Frame Application initially adds a Gateway DTM (for a remote IO) to the topology and afterwards adds a Device DTM (for a HART device).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

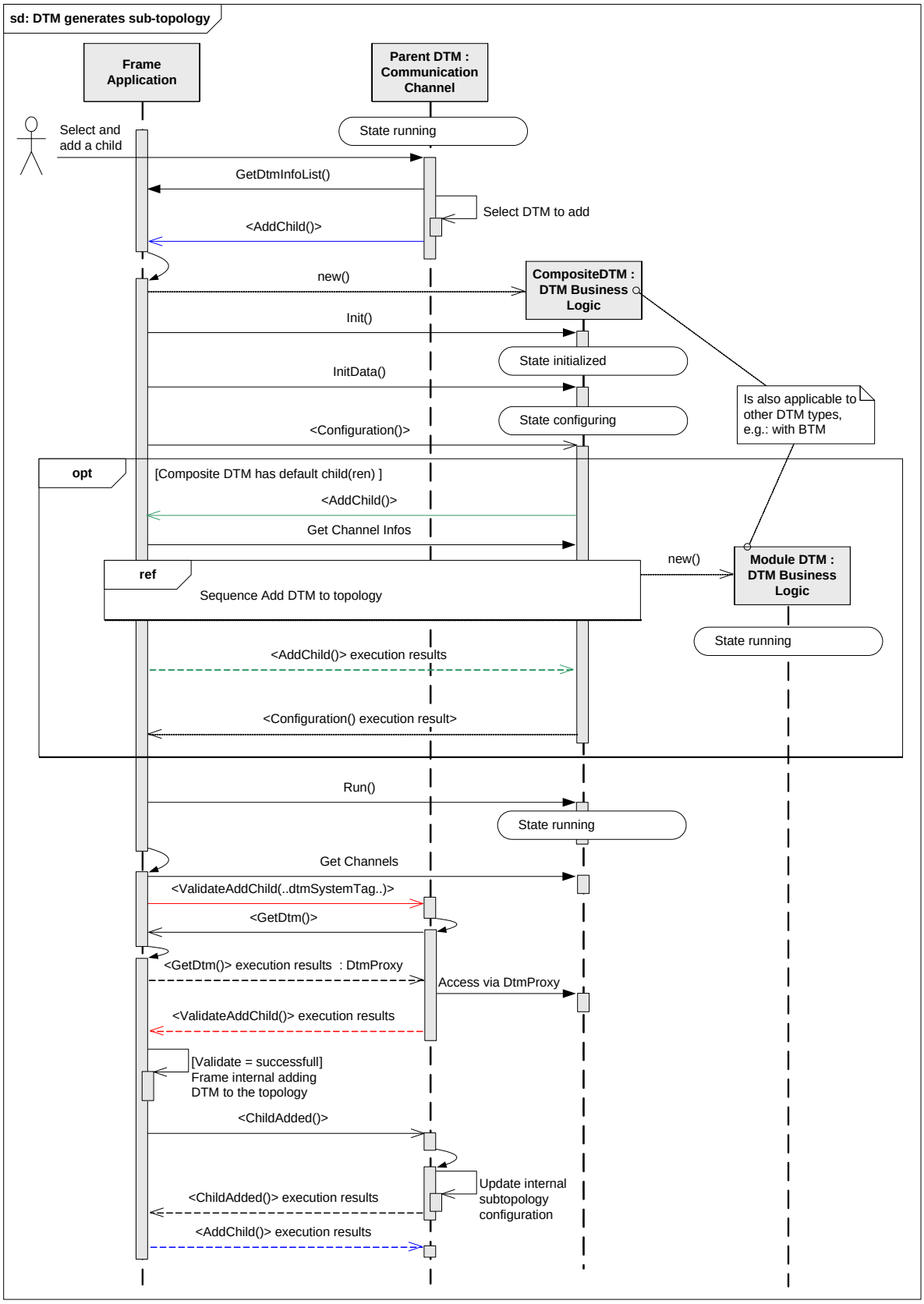
IChannels.ChannelInfos

Figure 166 — Frame Application creates topology

8.8.5 DTM generates sub-topology

This sequence diagram shows the generation of the sub-topology triggered by a DTM (see Figure 167).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

ITopology.GetDtmInfoList()
ITopology.BeginAddChild() / ITopology.EndAddChild()
ISubTopology.BeginValidateAddChild() / ISubTopology.EndValidateAddChild()
ISubTopology.BeginChildAdded() / ISubTopology.EndChildAdded()

Figure 167 — DTM generates sub-topology

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

The same sequence can be used for adding Module DTMs to a Composite Device DTM and for adding BTMs to Device DTMs.

8.8.6 Physical Layer and DataLinkLayer

IEC 61158-2 defines a wide range of possible physical media that is used by different fieldbus protocols. Many fieldbus protocols support different physical media. For example, HART supports wired (4-20 mA) and wireless connections, while PROFIBUS supports RS485, manchester-coded bus powered (MBP) and optical media. Even if a field device supports the same fieldbus protocol as a communication component (e.g. fieldbus interface or gateway), communication may be impossible, because device and communication component support different physical media. In such cases the use of media converters or gateway devices is required.

In order to avoid such incompatibility during offline planning of a physical topology, a Frame Application should use the physical layer information, which is exposed in the property Port.PhysicalLayers.

On the other hand, different protocols may share the same physical layer (e.g. Ethernet based protocols). If a physical layer is shared between protocols, it depends additionally on the IEC 61158-2 Data Link Layer, whether a physical connection is feasible or not.

In order to facilitate such checks, a Frame Application should use the data link layer information, which is exposed by the property Port.DataLinkLayers.

For comparison of the supported physical layer and data link layer, the properties PhysicalLayer and DataLinkLayer are used.

The following rules apply for Frame Applications managing the physical topology:

- If PhysicalLayer values do not match and DataLinkLayer values do not match, the Frame Application shall reject the new connection.
- If PhysicalLayer values match but DataLinkLayer values do not match, the Frame Application may reject the new connection.
- If PhysicalLayer values do not match but DataLinkLayer values do match, the Frame Application may issue a warning and accept the new connection, since the planned physical topology might contain transparent media converters, which are not part of the physical topology in the Frame Application.
- If both layers match, the Frame Application shall accept the new connection.

See Annex J for examples of PhysicalLayer values.

8.9 Instantiation, configuration, move and release of Child DTMs

8.9.1 General

The following workflows describe interactions between Parent DTM and Child DTMs. Such interactions may occur for instance between:

- Composite Device DTM and related Module DTMs
- Device DTM and related Block DTMs
- Gateway DTM and related Device DTMs

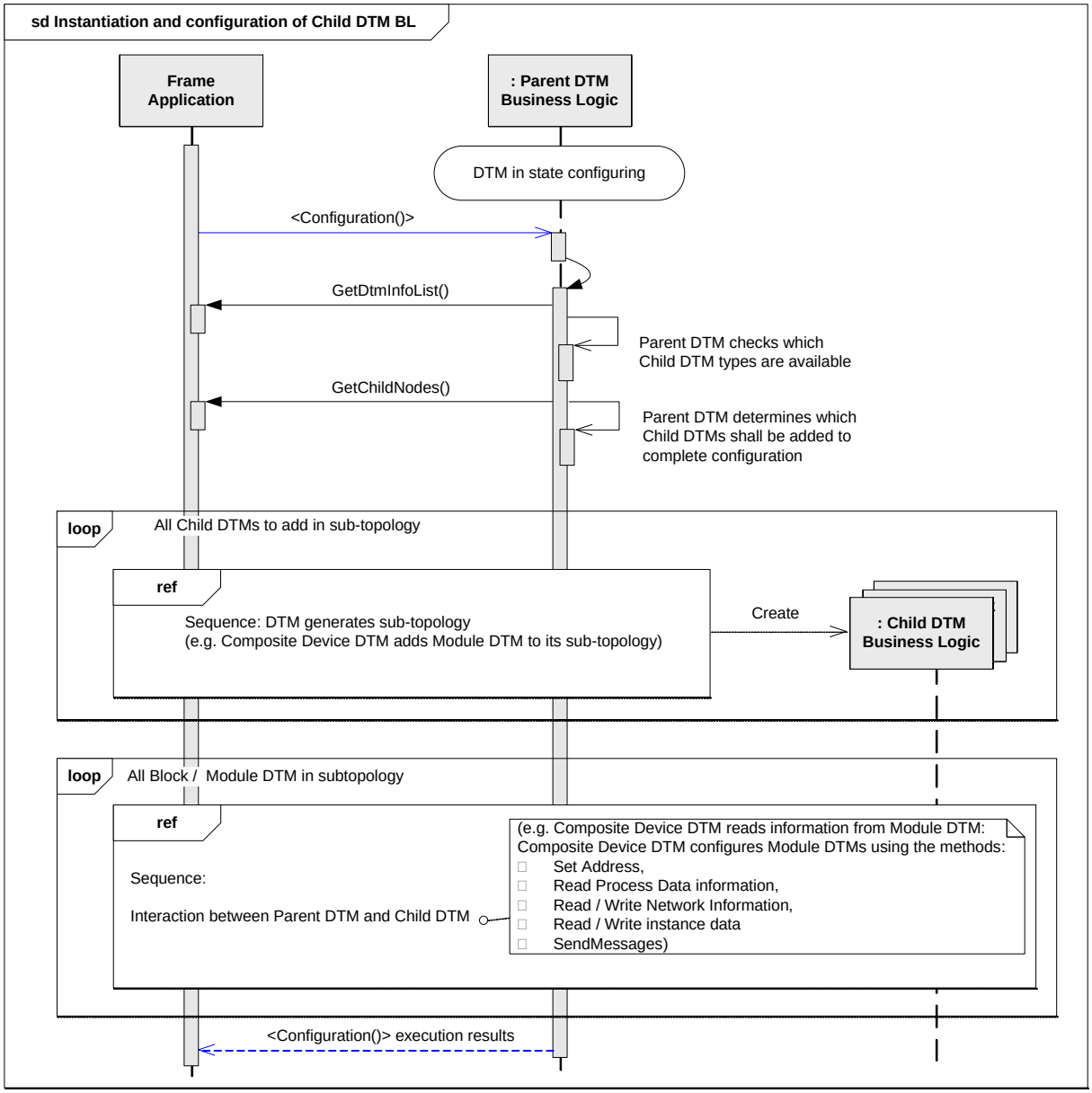
8.9.2 Instantiation and configuration of Child DTM BL

The diagram in Figure 168 shows how a Parent DTM can create and configure its sub-topology. In order to enable configuration of a sub-topology, the Parent DTM has to implement the

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

<Configuration()> method. The Frame Application shall call <Configuration()> when the DTM is in state 'configuring'.

A Parent DTM shall add Child DTMs only to its own channels.



Used methods:

IDtm3.BeginConfiguration()
IDtm3.EndConfiguration()
ITopology.GetDtmInfoList()
ITopology.GetChildNodes()

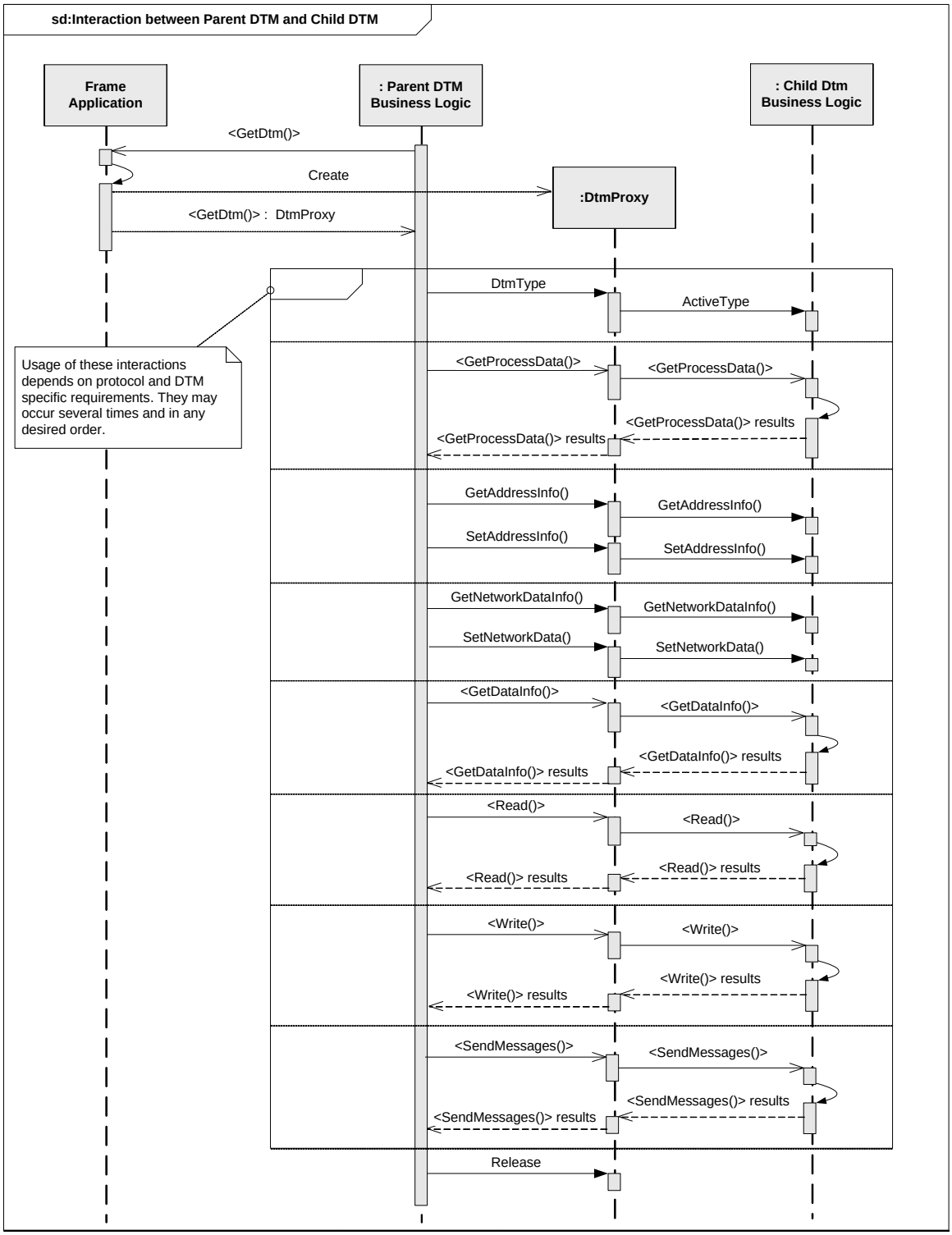
Figure 168 — Instantiation and configuration of Child DTM BL

8.9.3 Interaction between Parent DTM and Child DTM

Figure 169 shows how a Parent DTM can exchange data with its Child DTM.

For interaction between DTMs only the interfaces shall be used which are provided by IDtmProxy.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

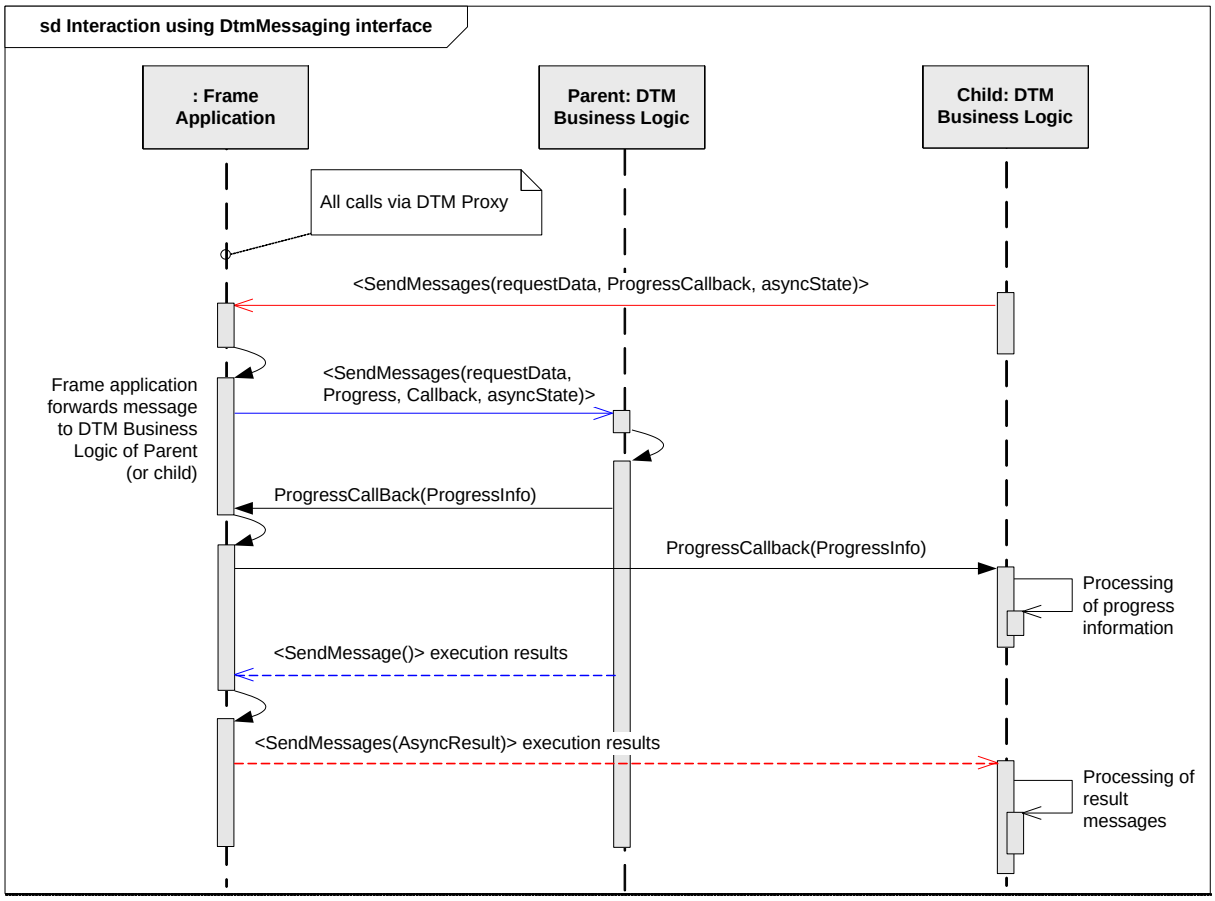
ITopology.BeginGetDtm() / ITopology.EndGetDtm()
IDtmProxy.Dispose()
IDtmProxy.DtmType
IDtm3.ActiveType
IProcessData.BeginGetProcessData() / IProcessData.EndGetProcessData()
INetworkData.GetAddressInfo() / INetworkData.SetAddressInfo()
INetworkData.GetNetworkDataInfo() / INetworkData.SetNetworkData()
IInstanceData.BeginGetDataInfo() / IInstanceData.EndGetDataInfo()
IInstanceData.BeginRead() / IInstanceData.EndRead()
IInstanceData.BeginWrite() / IInstanceData.EndWrite()

Figure 169 — Interaction between Parent DTM and Child DTM

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8.9.4 Interaction between Parent DTM and Child DTM using IDtmMessaging

This sequence diagram outlines the interaction between two DTMs using the IDtmMessaging interface (see Figure 170).



Used methods:

IDtmMessaging.BeginSendMessages()
IDtmMessaging.EndSendMessages()

Figure 170 — Interaction using IDtmMessaging

In this scenario the DTM Business Logic of a Child DTM sends a list of proprietary messages to its Parent DTM. The Frame Application provides access to the IDtmMessaging by means of the IDtmProxy. It shall forward the messages to the corresponding DTM.

More detailed information can be found in descriptions of:

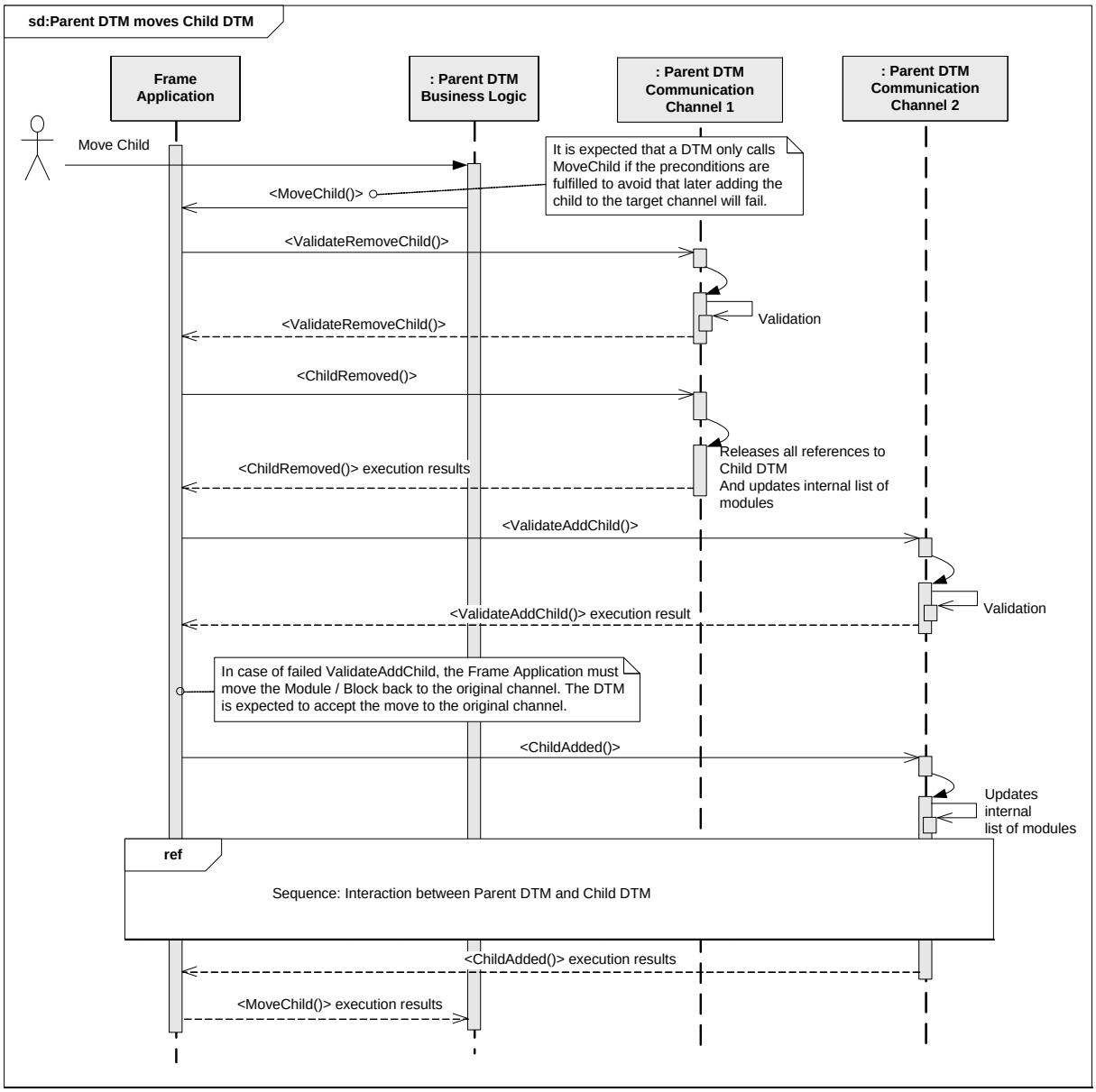
- IDtmMessaging
- DtmRequestMessage
- DtmResponseMessage

8.9.5 Parent DTM moves a Child DTM

Figure 171 shows how a Parent DTM can move one of its Child DTMs from one channel to another channel.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

A Parent DTM shall move Child DTMs only between its own channels.



Used methods:

ITopology.BeginMoveChild() / ITopology.EndMoveChild()
ISubTopology.BeginValidateAddChild() / ISubTopology.EndValidateAddChild()
ISubTopology.BeginChildAdded() / ISubTopology.EndChildAdded()
ISubTopology.BeginValidateRemoveChild() / ISubTopology.EndValidateRemoveChild()
ISubTopology.BeginChildRemoved() / ISubTopology.EndChildRemoved()

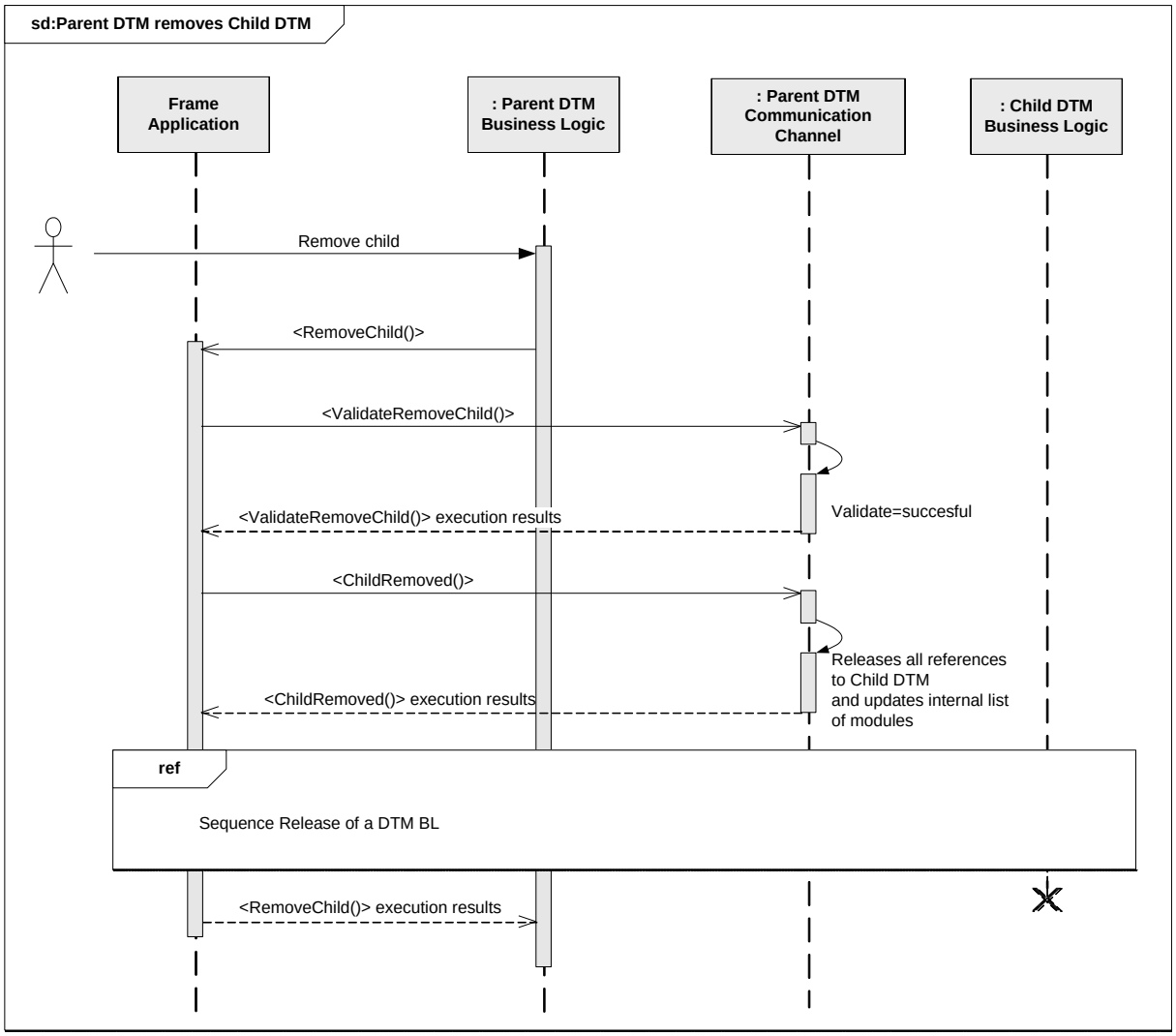
Figure 171 — Parent DTM moves a Child DTM

8.9.6 Parent DTM removes Child DTM

Figure 172 shows how a Parent DTM can remove one of its Child DTM

A Parent DTM can remove only its own Child DTMs.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

ITopology.BeginRemoveChild() / ITopology.EndRemoveChild()
ISubTopology.BeginValidateRemoveChild() / ISubTopology.EndValidateRemoveChild()
ISubTopology.BeginChildRemoved() / ISubTopology.EndChildRemoved()

Figure 172 — Parent DTM removes Child DTM

8.10 Topology scan

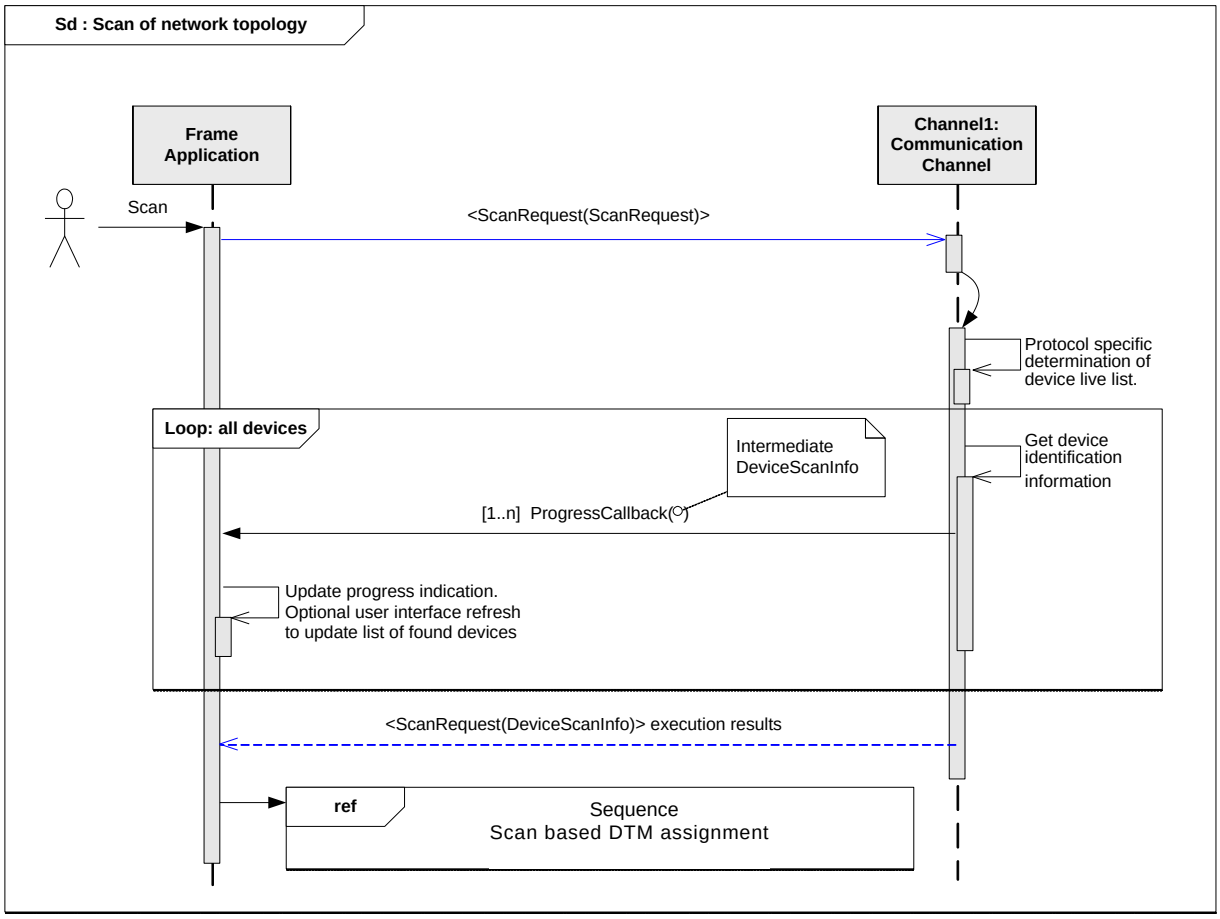
8.10.1 General

For a description of the general mechanism see IEC 62453-2 chapter 6.2.

8.10.2 Scan of network topology

The following workflow describes, how a Frame Application can request a list of connected devices and their protocol-specific device identification information from a Communication Channel (see Figure 173).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IScanning.BeginScanRequest()
IScanning.EndScanRequest()
ProgressCallback

Figure 173 — Scan of network topology

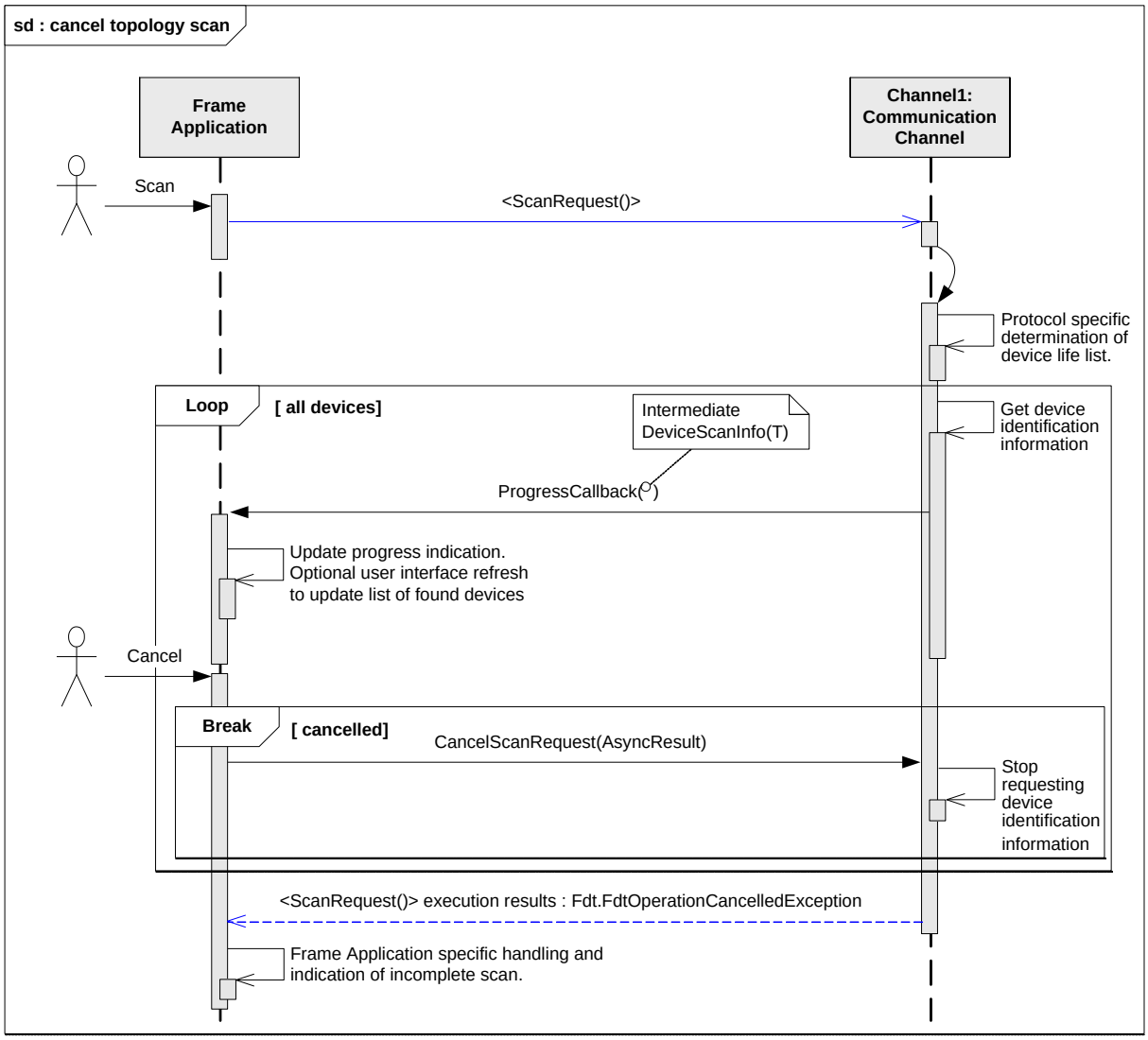
The final result data for the scan received with the IScanning.EndScanRequest() contains a list of DeviceScanInfo objects where each object contains information about a single device found on the bus. If the order of devices is relevant for the protocol of the Communication Channel, the order of objects in the final result list shall match the order of the devices on the bus. Contrary to the final result, the order of devices in the intermediate results may depend from the scanning algorithm and may differ from the final result.

For information on how a protocol-specific DeviceScanInfo(T) can be transformed into a protocol-independent DeviceScanInfo, please refer to the datatype definition (see Annex B).

8.10.3 Cancel topology scan

Scanning a sub-topology may take some time. The FDT methods are designed to be called asynchronously. If a Frame Application calls the scan methods asynchronously, CancelScanRequest() may be called to cancel an ongoing scanning operation in the Communication Channel. The following sequence shows the related flow of events (see Figure 174).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

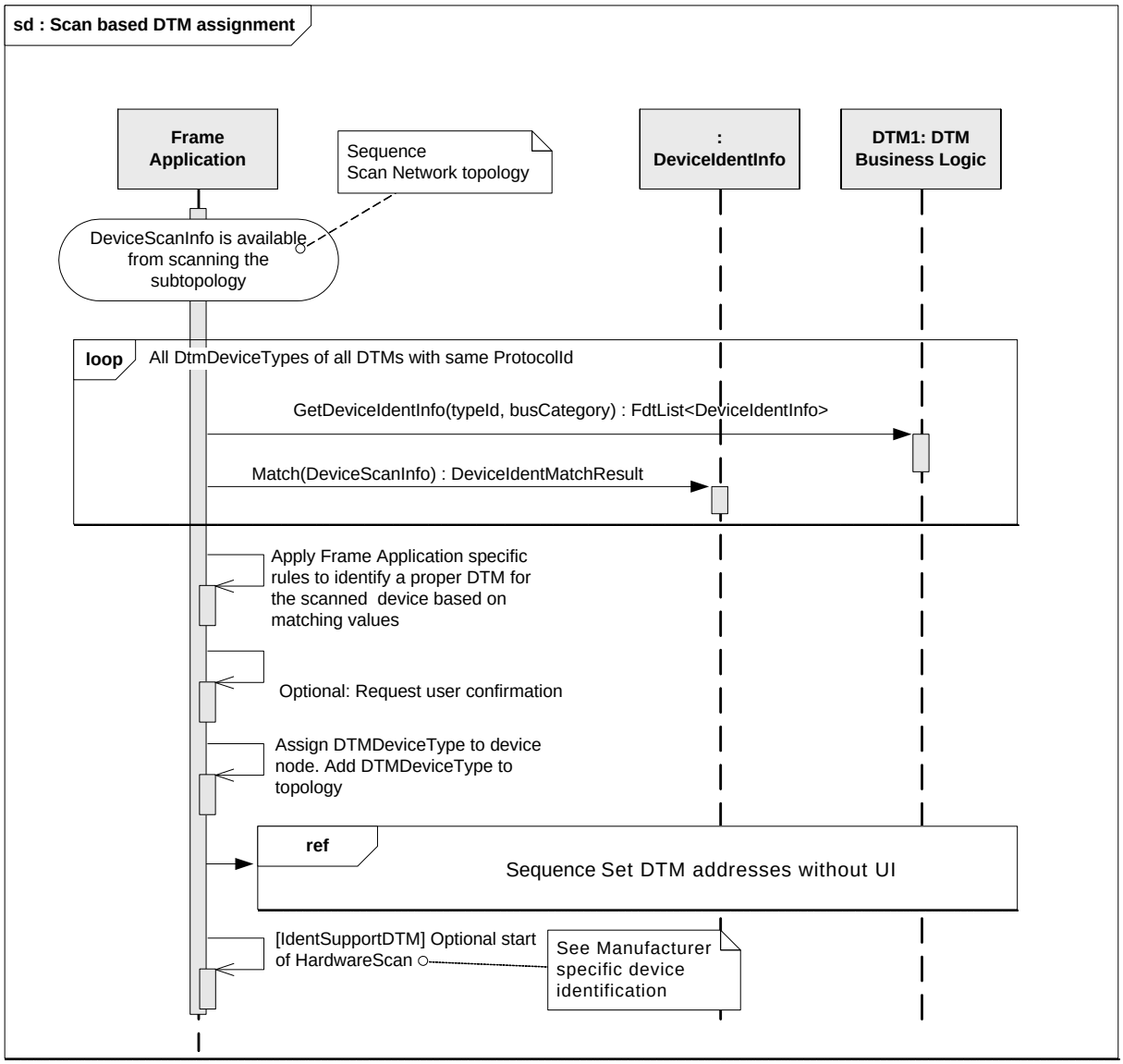
IScanning.BeginScanRequest()
IScanning.CancelScanRequest()
Callback ScanProgress

Figure 174 — Cancel topology scan

8.10.4 Scan based DTM assignment

A Frame Application may use the scanned life list to find appropriate DTMDeviceTypes and create a sub-topology accordingly. The following sequence chart describes the related flow of events (see Figure 175).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

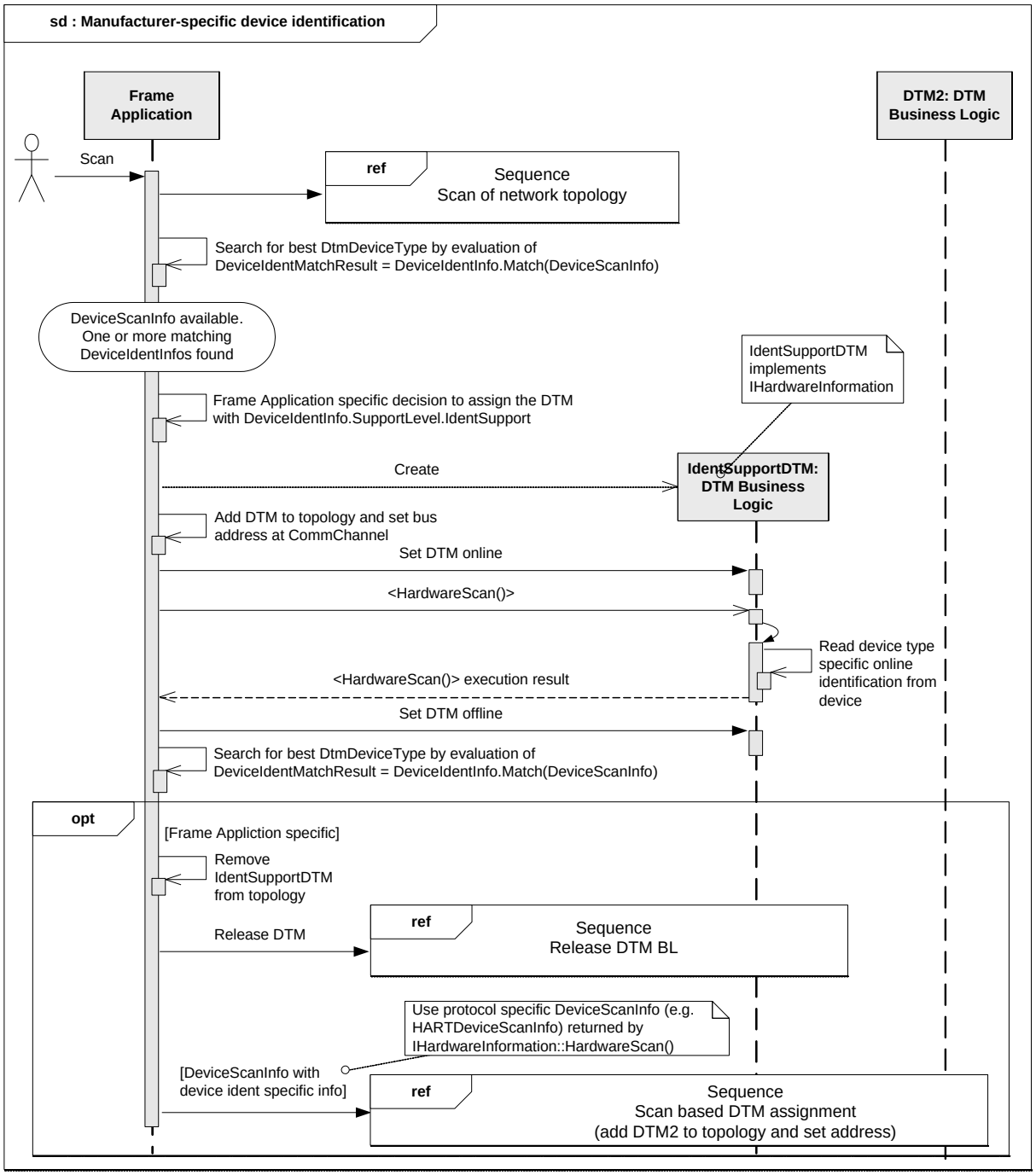
IDtmInformation.GetDeviceIdentInfo()
DeviceIdentValue<T>.Match()

Figure 175 — Scan based DTM assignment

8.10.5 Manufacturer-specific device identification

In this scenario a Frame Application scans an existing fieldbus network and uses DTM implementing IHardwareInformation interface to identify devices for which manufacturer-specific operation shall be performed (see Figure 176).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IHardwareInformation.BeginHardwareScan()
IHardwareInformation.CancelHardwareScan()
IHardwareInformation.EndHardwareScan()
DeviceScanInfo

Figure 176 — Manufacturer-specific device identification

For more information on manufacturer-specific device identification refer to section 4.13.4.

8.11 Configuration of communication networks

8.11.1 Configuration of a fieldbus master

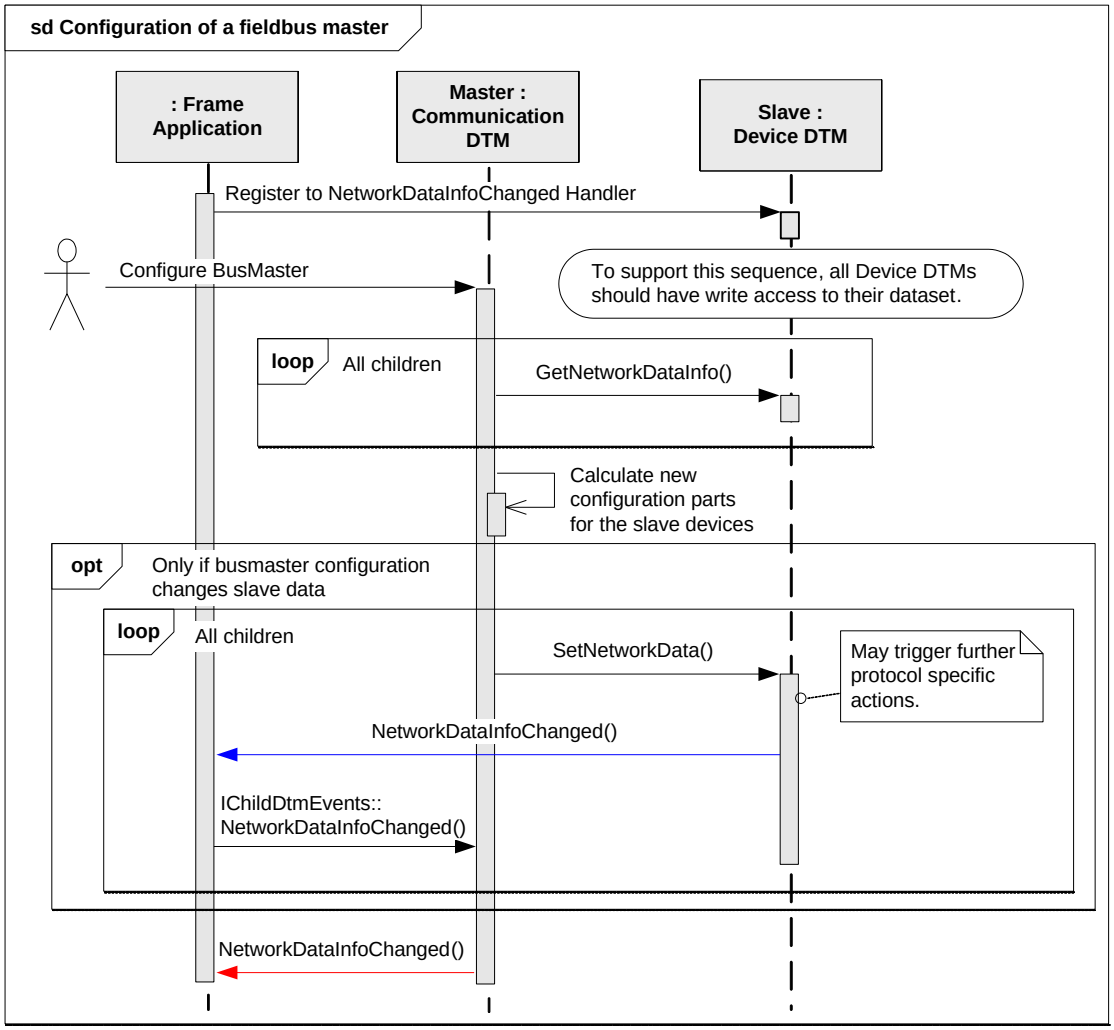
Device-specific bus parameters are needed to configure the fieldbus master or communication scheduler. To retrieve these parameters an interaction between DTMs and a master configuration tool (e.g. provided by Master Communication DTM or by Frame Application) is

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

required. Bus-specific data information is provided by Device DTMs in NetworkDataInfo and contains the device-specific bus information according to the fieldbus protocol specification (see FDT Protocol Annex specifications for protocol-specific definitions).

When NetworkDataInfo is available from all Slave DTMs, the master configuration tool can commission the fieldbus (see Figure 177). For that purpose, it uses protocol-specific master configuration information from each network participant and calculates the bus parameters of the corresponding master device.

The master configuration can be provided by the DTM (Figure 177) representing the bus master hardware or by a bus-master-specific Frame Application.



Used methods:

INetworkData.GetNetworkDataInfo()
INetworkData.SetNetworkData()
Event IChildDtmEvents.NetworkDataInfoChanged()

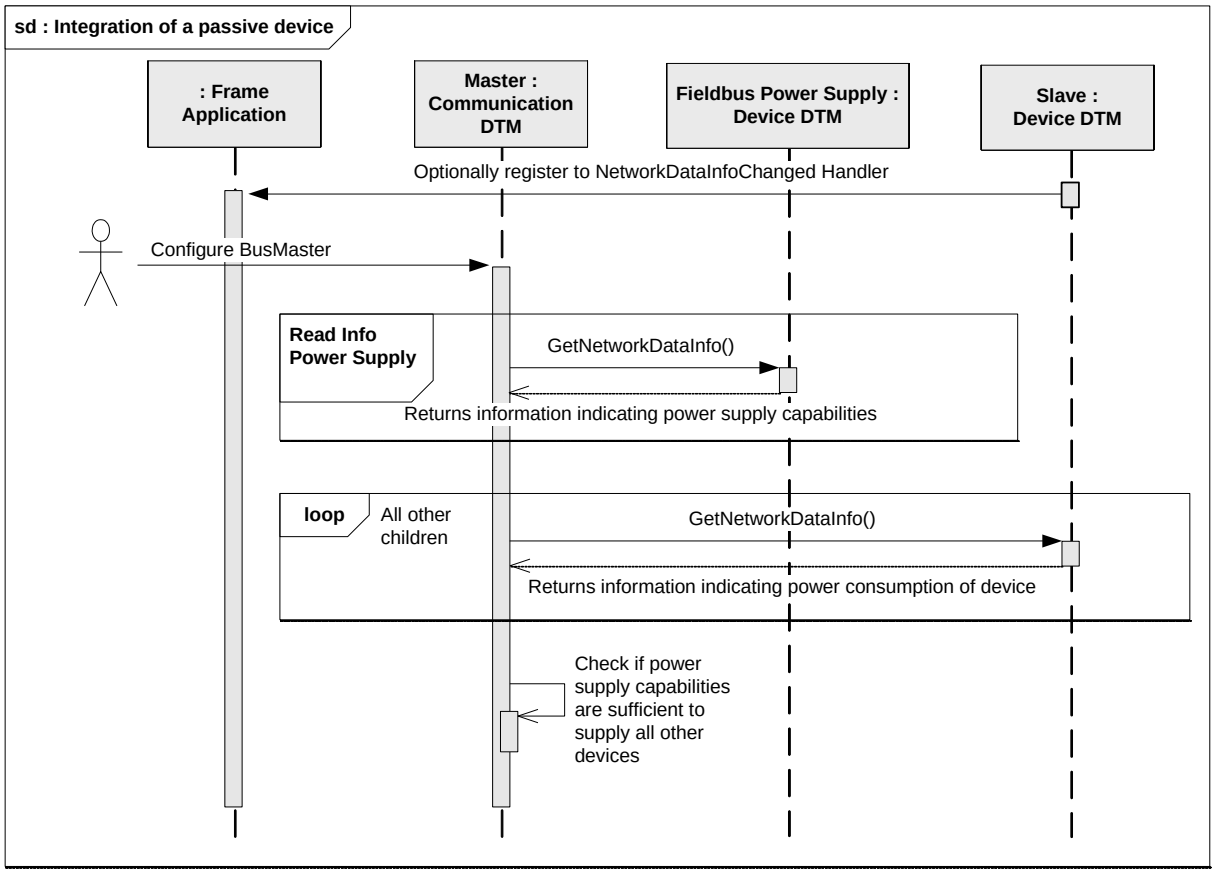
Figure 177 — Configuration of a fieldbus master

The transfer of the network information to the network (master device and/or field devices) is protocol specific or product specific. For description of protocol-specific rules please refer to the respective FDT Protocol Annex.

8.11.2 Integration of a passive device

This section shows the sequence when integrating information for a passive device as part of network configuration (see Figure 178).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

INetworkData.GetNetworkDataInfo()

Figure 178 — Integration of a passive device

After retrieving the NetworkDataInfo from the Device DTM for the fieldbus power supply and for the field devices, it is possible to compare the power consumption of the field devices with the power provided by the fieldbus power supply. If the consumption exceeds the provided power, the user should be informed.

8.12 Using IO information

8.12.1 Assignment of symbolic name to process data

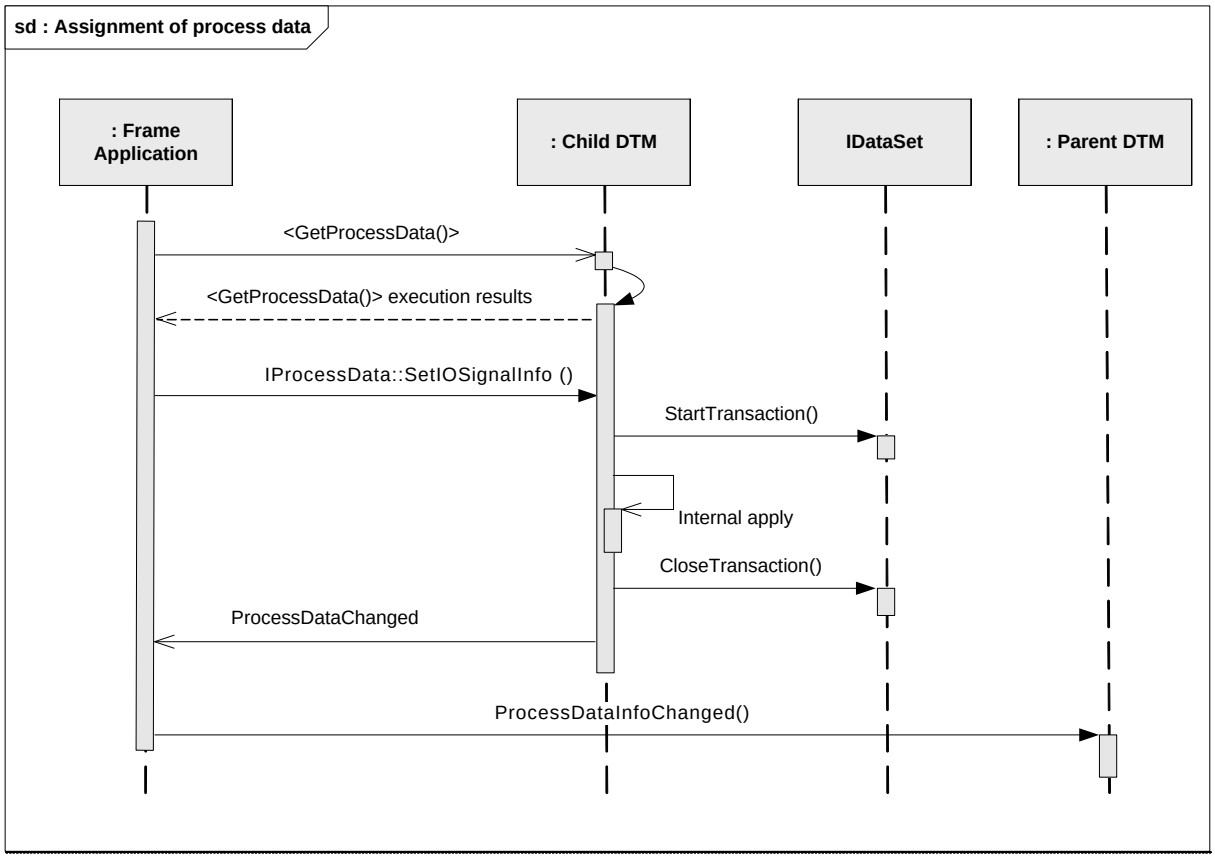
Figure 179 shows an example workflow how a PLC Tool Frame Application assigns an IO Signal defined by IProcessData to a variable used for PLC programming.

Note: The same mechanism is used for assignment of variables in DCS tools. This process may be referred to as “DCS channel assignment”.

The Frame Application first fetches a list of available process data (IO signals) from the DTM. It can then offer the user to assign a symbolic name to each of the IO signals contained in the list of process data. (See IEC 61131-3(ed2.0), section 2.4.3.1 Type assignment)

The symbolic variable name defined in the PLC program is stored in the property “FrameApplicationTag” of IOSignalInfo. If an IO Signal is used by the Frame Application (in a PLC program or otherwise), then this shall also be indicated by the property “IsLocked” of IOSignalInfo. Setting of the FrameApplicationTag and IsLocked is done using the method SetIOSignalInfo().

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:
IProcessData.BeginGetProcessData()
IProcessData.EndGetProcessData()
IProcessData.SetIOSignalInfo()
IDataset.StartTransaction()
IDataset.CloseTransaction()
Event IProcessData.ProcessDataChanged()
Event IChildDtmEvents.ProcessDataInfoChanged()

Figure 179 — Assignment of process data

A Frame Application shall set only the FrameApplicationTag for IO signals provided by a DTM directly using IProcessData. If a DTM provides IOSignals for Child DTMs (see 4.4.4) then the Frame Application shall set the respective properties at the Child DTMs, but not at the Parent DTMs.

Alternatively the interface IProcessImage can be used if it is provided by the corresponding Parent DTM (see section 8.12.4) to change IOSignals properties for Child DTMs.

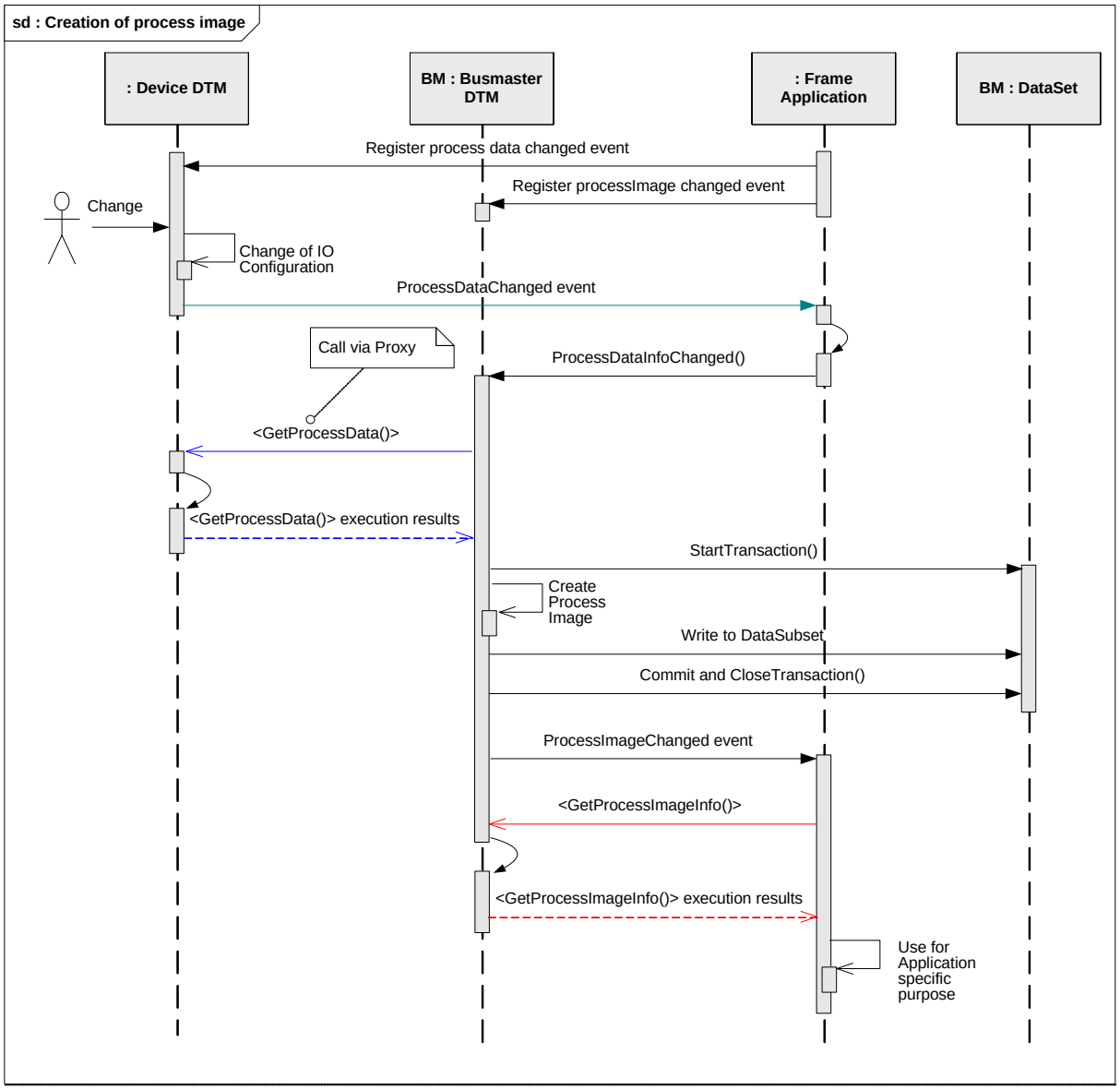
Assignment of process data to a PLC variable using the interface IProcessData is protocol specific. Protocol independent assignment can be done using interface IProcessImage.

8.12.2 Creation of Process Image

This sequence shows the creation and publishing of the process image by a DTM representing a busmaster (see Figure 180). This sequence diagram shows no validation of changes. Validation is described in section 8.12.3.

If the user changes the IO Configuration e.g. on a DTM-specific interface of a Device DTM the Frame Application receives a notification about this change. The notification is then forwarded to the Busmaster DTM. Since the notification contains the IDs of accessible data which is changed, the Busmaster DTM can examine the changes. Depending on the kind of changes the Busmaster DTM might fetch the process data of the Device DTM and cache this information.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

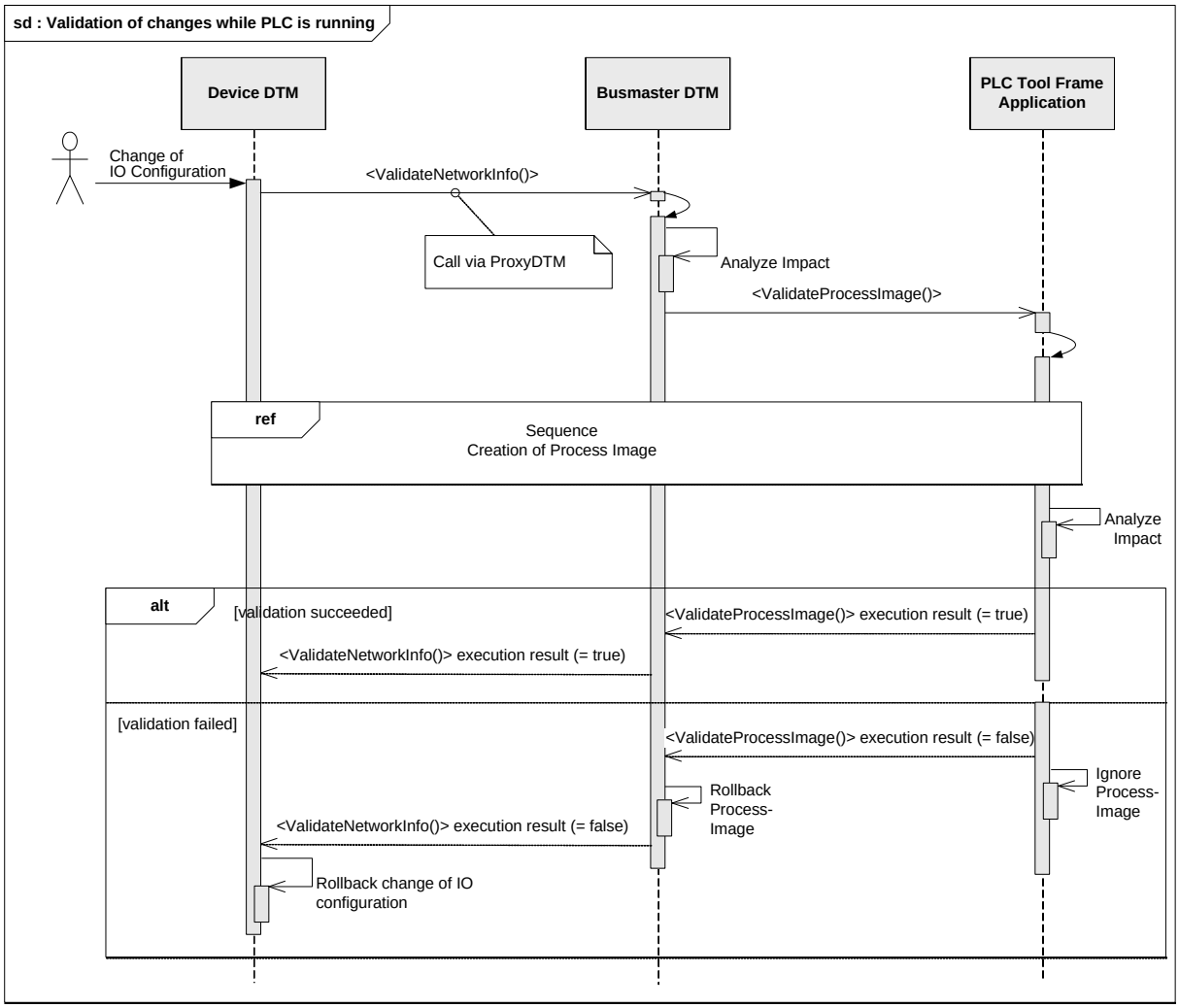
IProcessData.BeginGetProcessData()
IProcessData.EndGetProcessData()
Event IProcessData.ProcessDataChanged()
IProcessImage.BeginGetProcessImageInfo()
IProcessImage.EndGetProcessImageInfo()
IProcessImage.EndGetProcessImageInfo()
Event IChildDtmEvents.ProcessDataInfoChanged()
IDataset.StartTransaction()
IDataset.CloseTransaction()

Figure 180 — Creation of process image

8.12.3 Validation of changes in process image while PLC is running

The following sequence diagram (Figure 181) shows the validations which can be done in case a PLC tool Frame Application supports changes of the configuration while the PLC is running.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

INetworkInfoValidation.BeginValidateNetworkInfo()
INetworkInfoValidation.EndValidateNetworkInfo()
IProcessImageValidation.BeginValidateProcessImage()
IProcessImageValidation.EndValidateProcessImage()

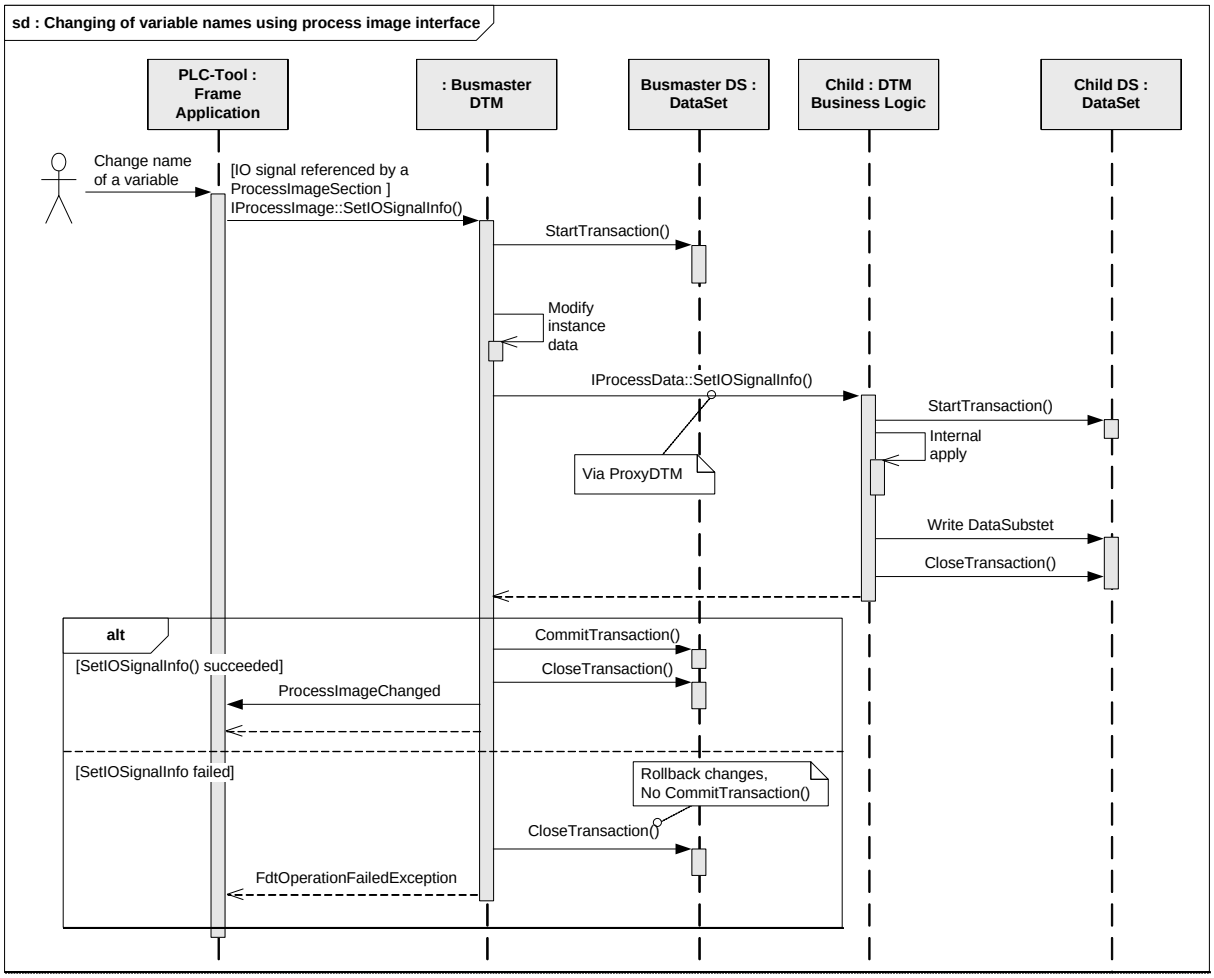
Figure 181 — Validation of changes while PLC is running

8.12.4 Changing of variable names using process image interface

Figure 182 shows how a PLC Tool Frame Application can change the names of variables using the Process Image interface.

The DTM shall also forward the call to corresponding Child DTMs by calling SetIOSignalInfo on the Process Data interface of the Child DTM.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IProcessImage.SetIOSignalInfo()
IProcessData.SetIOSignalInfo()
IDataset.StartTransaction()
IDataset.CommitTransaction()
IDataset.CloseTransaction()

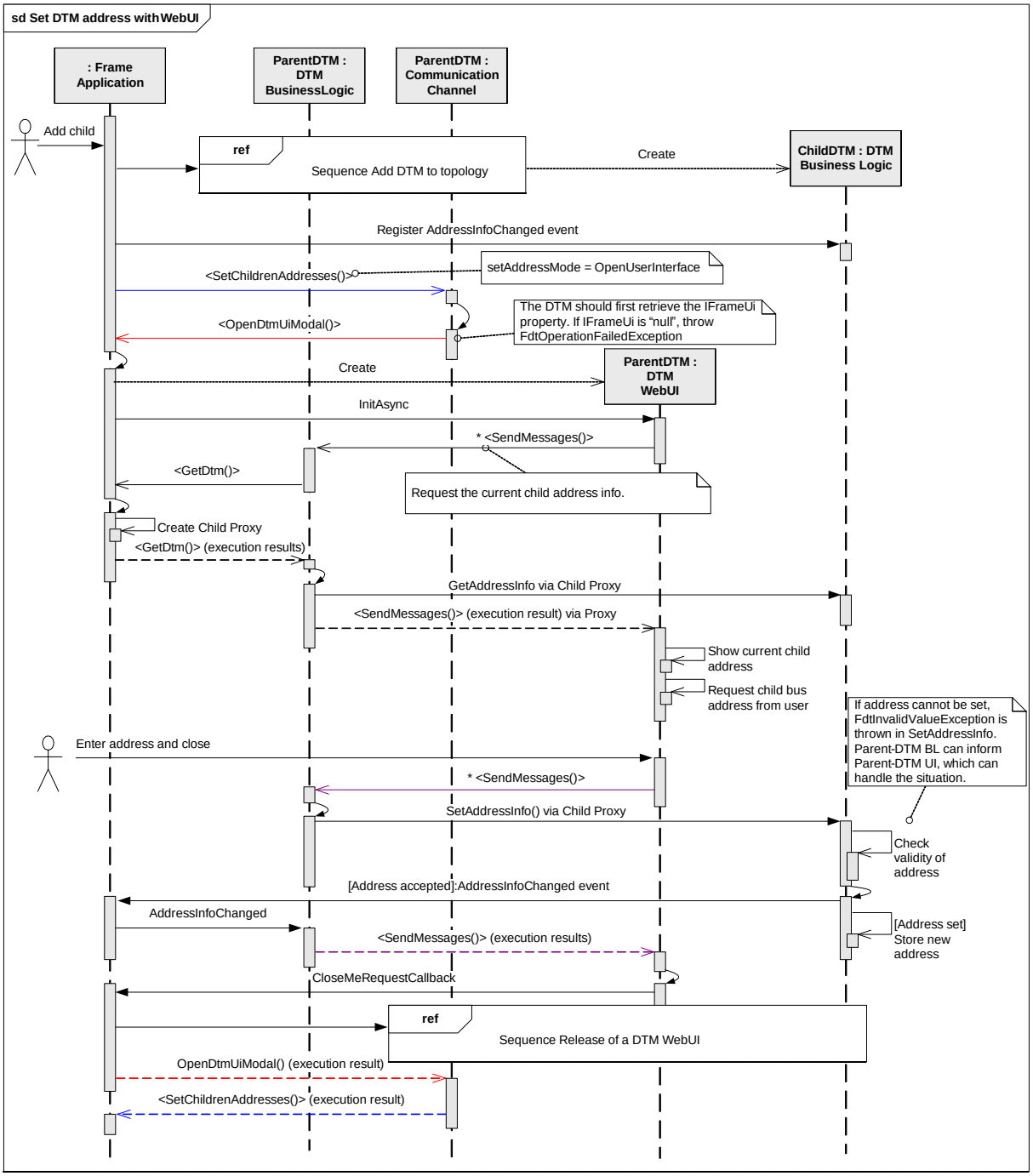
Figure 182 — Changing of variable names using process image interface

8.13 Managing addresses

8.13.1 Set DTM address with user interface

In this scenario the Frame Application requests setting child device addresses at the parent Communication Channel (e.g.: bus master DTMs). This sequence is started (see Figure 183) for example when a new DTM is added to the topology. A similar sequence can be applied if a Frame Application offers changing the address of a DTM manually.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

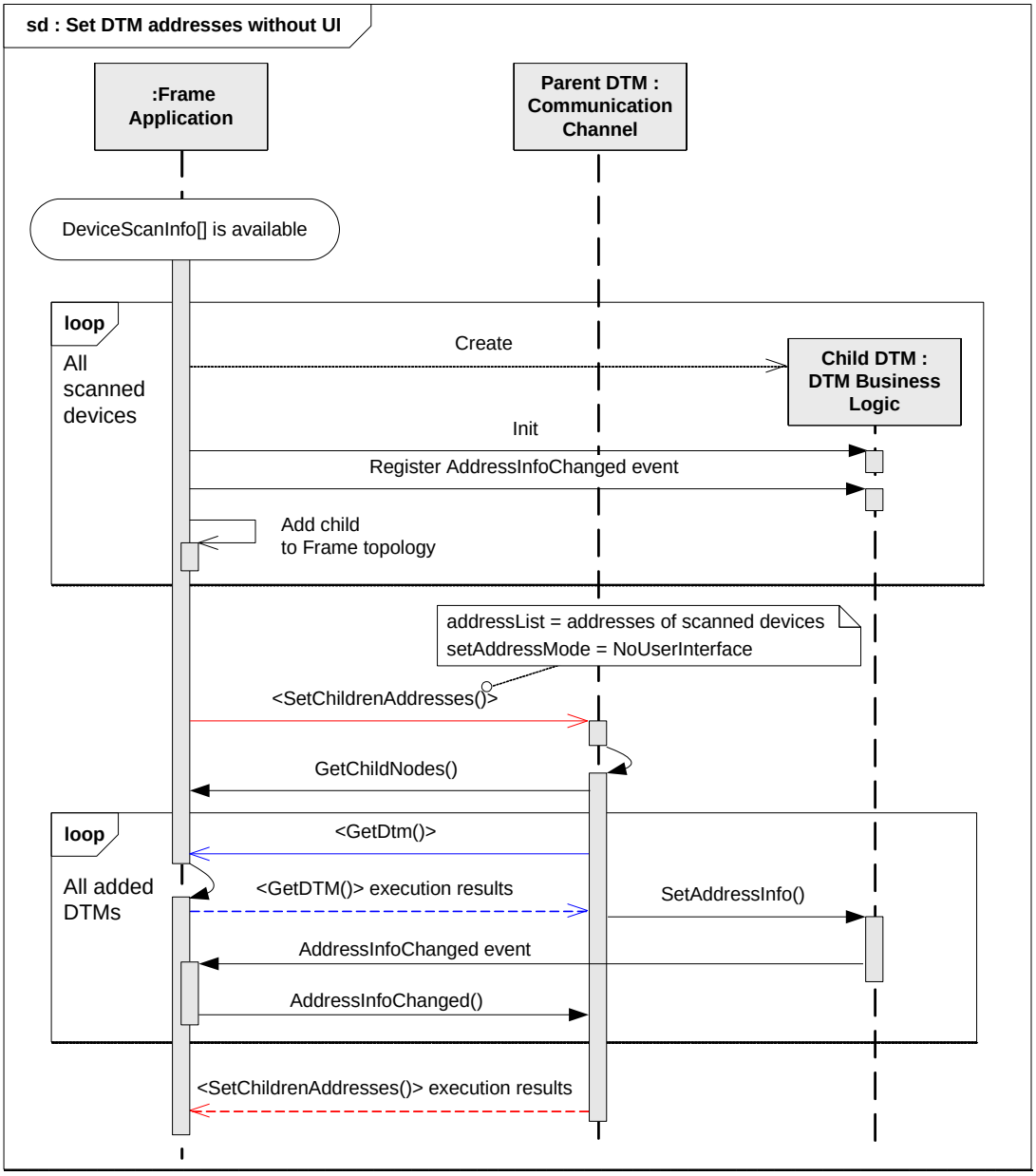
ISubTopology.BeginSetChildrenAddresses() / ISubTopology.EndSetChildrenAddresses()
IFrameUi.BeginOpenDtmUiModal() / IFrameUi.EndOpenDtmUiModal()
IDtmWebUiFunction.initAsync()
IDtmUiMessaging.BeginSendMessages() / IDtmUiMessaging.EndSendMessages
CloseMeRequestHandler
ITopology.BeginGetDtm() / ITopology.EndGetDtm()
INetworkData.GetAddressInfo() / INetworkData.SetAddressInfo()
Event INetworkData.AddressInfoChanged()
Event IChildDtmEvents.AddressInfoChanged()

Figure 183 — Set DTM address with WebUI

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8.13.2 Set DTM addresses without user interface

The following example shows the sequence of setting Child DTM addresses after scanning and DTM assignment. The Frame Application requests at a Communication Channel to set a number of known device addresses at Child DTMs (see Figure 184).



Used methods:

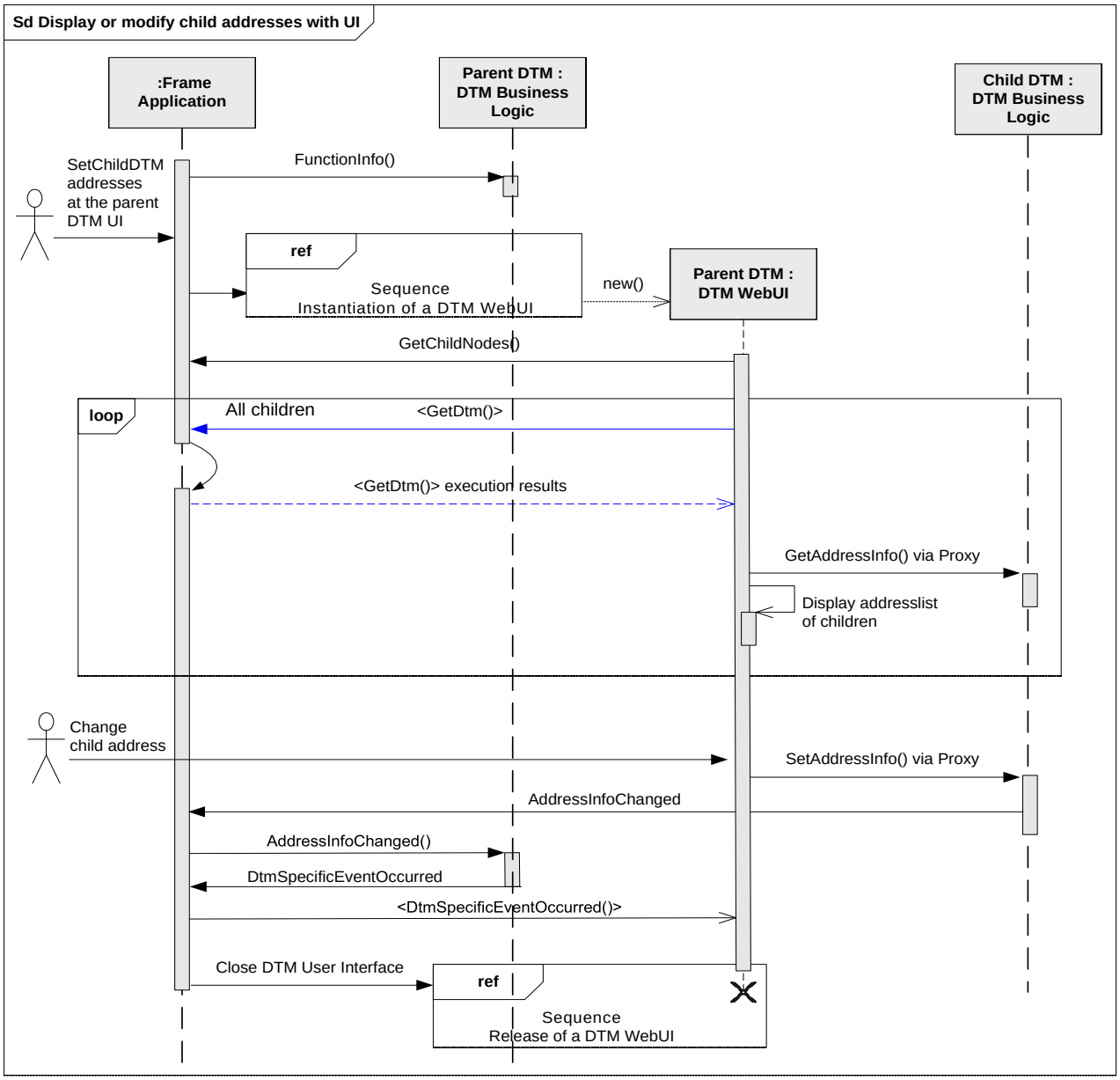
ISubTopology.BeginSetChildrenAddresses() / ISubTopology.EndSetChildrenAddresses()
ITopology.GetChildNodes()
ITopology.BeginGetDtm() / ITopology.EndGetDtm()
INetworkData.SetAddressInfo()
Event INetworkData.AddressInfoChanged()
Event IChildDtmEvents.AddressInfoChanged()

Figure 184 — Set DTM addresses without UI

8.13.3 Display or modify addresses of all Child DTMs with user interface

In this scenario Frame Application requests to display or modify all Child DTM addresses at a Parent DTM. This sequence (see Figure 185) for example is started when a user selects the corresponding menu entry in context of a Communication DTM or a Gateway DTM.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IFunction.Functioninfo
ITopology.GetChildNodes()
ITopology.BeginGetDtm() / ITopology.EndGetDtm()
INetworkData.GetAddressInfo()
INetworkData.SetAddressInfo()
Event INetworkData.AddressInfoChanged()
Event IChildDtmEvents.AddressInfoChanged()
Event IDtmUiMessaging.DtmSpecificEventOccured()

Figure 185 — Display or modify child addresses with UI

8.14 Device-initiated data transfer

Some protocols support data transfer services which are initiated by the device and not by the DTM. A Communication Channel supports this by providing the ISubscription interface. For an example of device initiated data transfer see Figure 186.

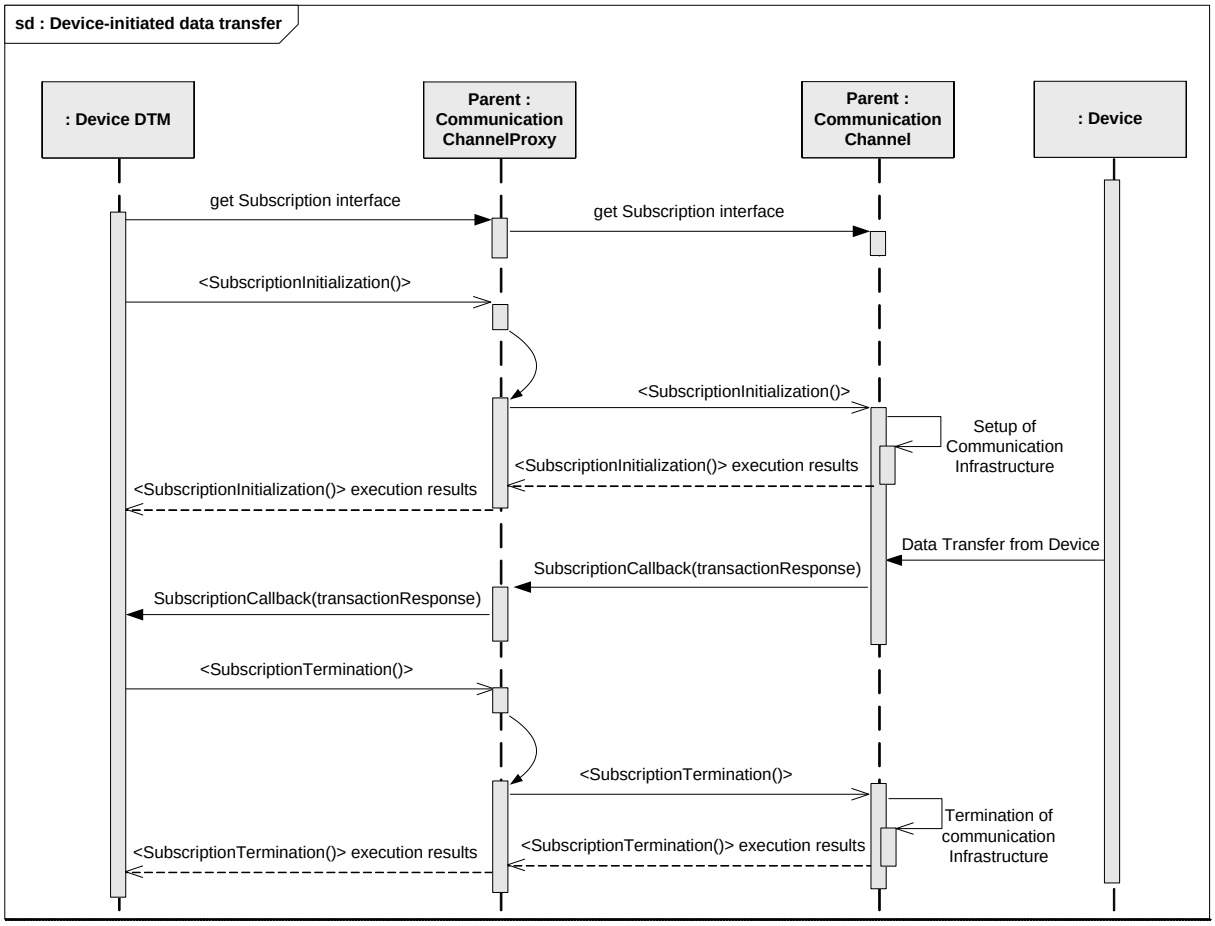
A Child DTM requests the ISubscription instance from the Communication Channel of the Parent DTM to access the subscription services.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

The infrastructure (e.g. filter, service queue) for such services is initiated by a protocol-specific request of the DTM to initialize the subscription.

The device initiated data transfer is transported by multiple invocations of the SubscriptionCallback() of the DTM with protocol-specific communication responses as arguments.

The infrastructure for these services is terminated by a protocol-specific request of the DTM to terminate the subscription.



Used methods:

ICommunicationChannelProxy.Subscription()
ICommunicationChannel.Subscription()
ISubscription.BeginSubscriptionInitialization() / ISubscription.EndSubscriptionInitialization()
Fdt.Communication.SubscriptionCallback()
ISubscription.BeginSubscriptionTermination() / ISubscription.EndSubscriptionTermination()

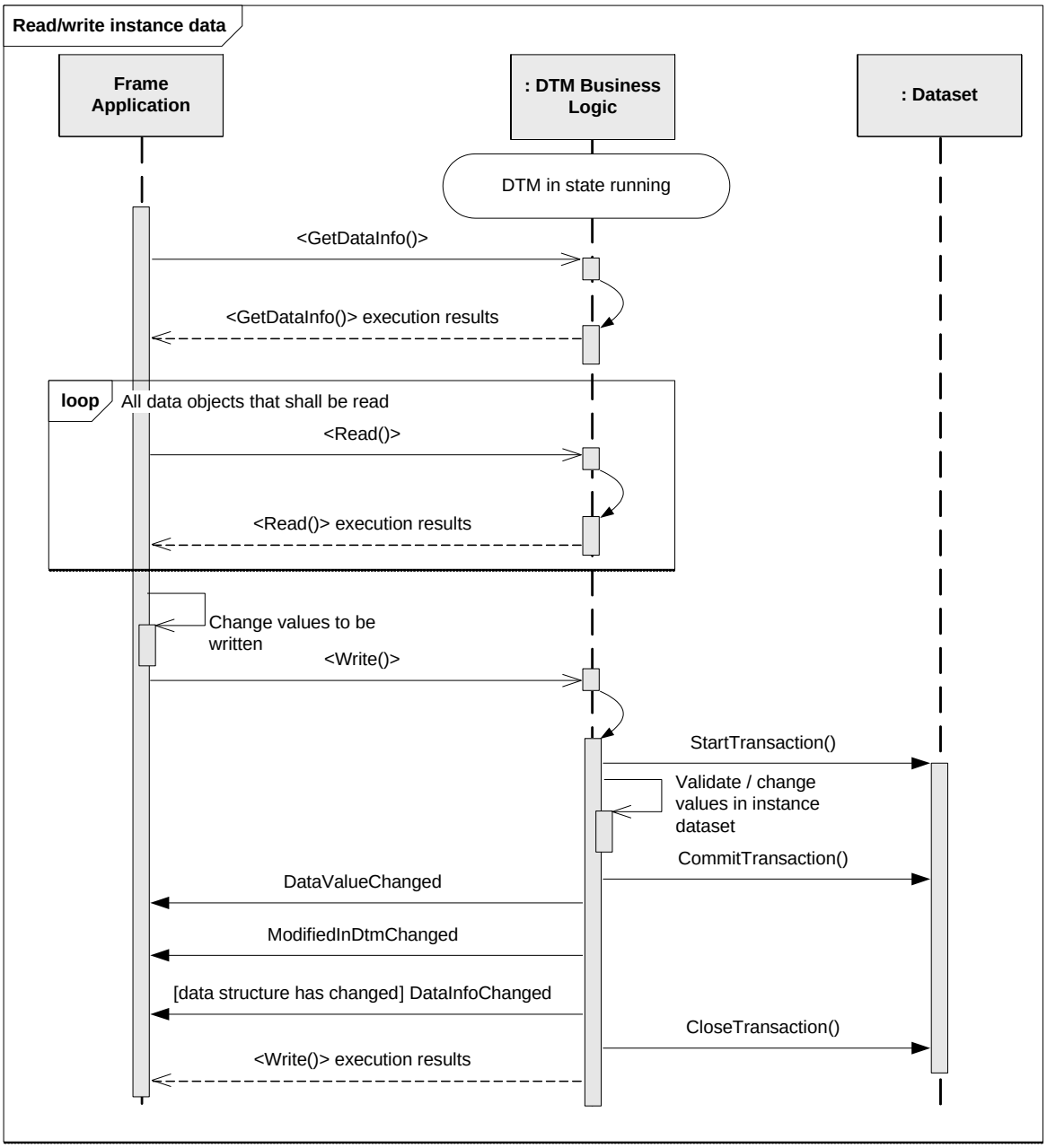
Figure 186 — Device-initiated data transfer

8.15 Reading and writing data

8.15.1 Read/write instance data

The following sequence diagram (Figure 187) shows how instance data is read from / written to the instance dataset using IInstanceData interface.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

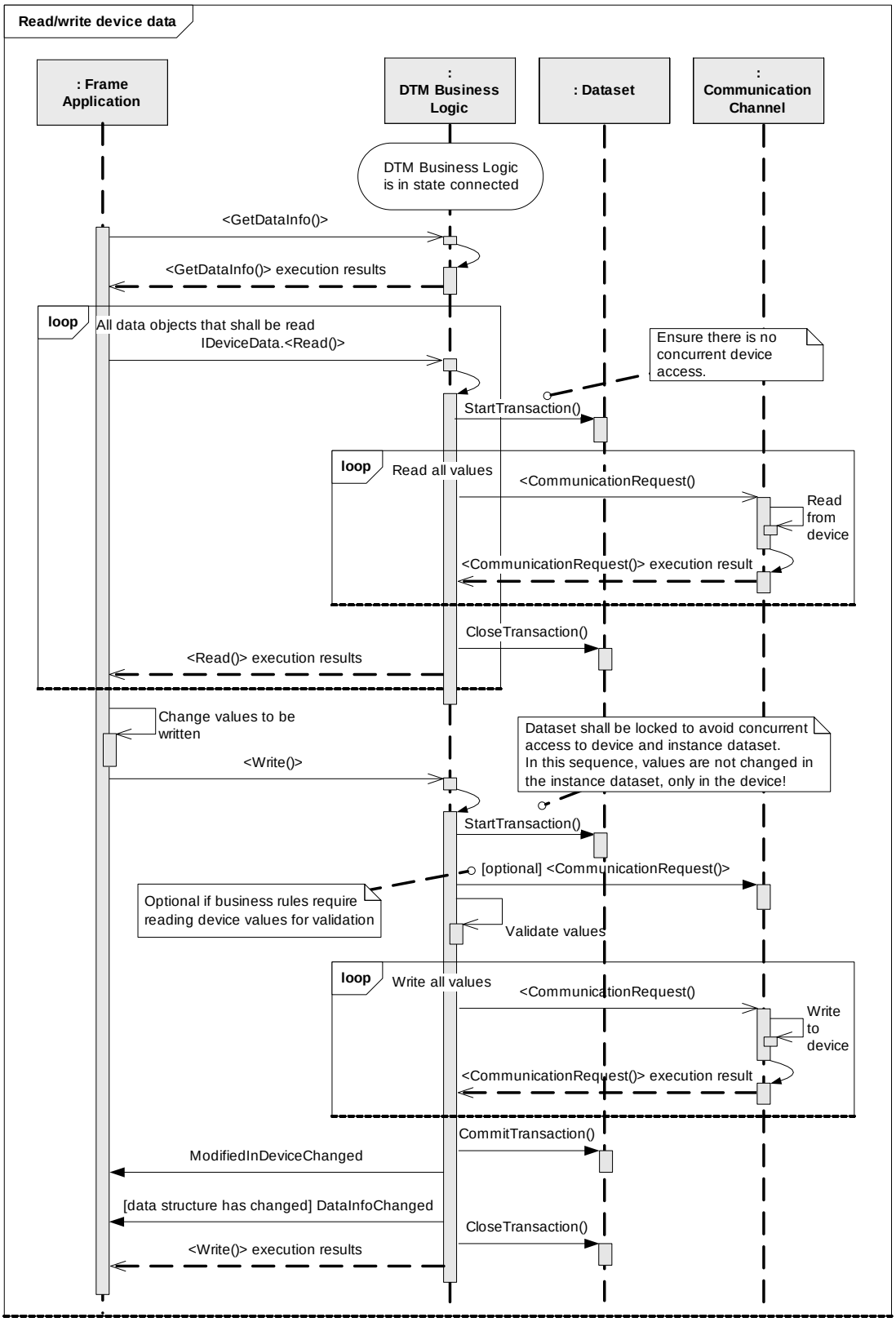
IInstanceData.GetDataInfo()
IDataset.StartTransaction() / IDataset.CommitTransaction() / IDataset.CloseTransaction()
IDataset.TransactionStarted / IDataset.TransactionCommitted /IDataset.TransactionClosed
IInstanceData.BeginRead() / IInstanceData.EndRead()
IInstanceData.BeginWrite() / IInstanceData.EndWrite()
Event IInstanceData.DataValueChanged()
Event IInstanceData.DataInfoChanged()

Figure 187 — Read/write instance data

8.15.2 Read/write device data

The following sequence diagram (Figure 188) shows how device data is read from / written to the device using IDeviceData interface.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IInstanceData.GetDataInfo()
IDataset.StartTransaction() / IDataset.CommitTransaction() / IDataset.CloseTransaction()
IDataset.TransactionStarted / IDataset.TransactionCommitted /IDataset.TransactionClosed
IDeviceData.BeginRead() / IDeviceData.EndRead()
IDeviceData.BeginWrite() / IDeviceData.EndWrite()
Event IDeviceData.ModifiedInDeviceChanged()

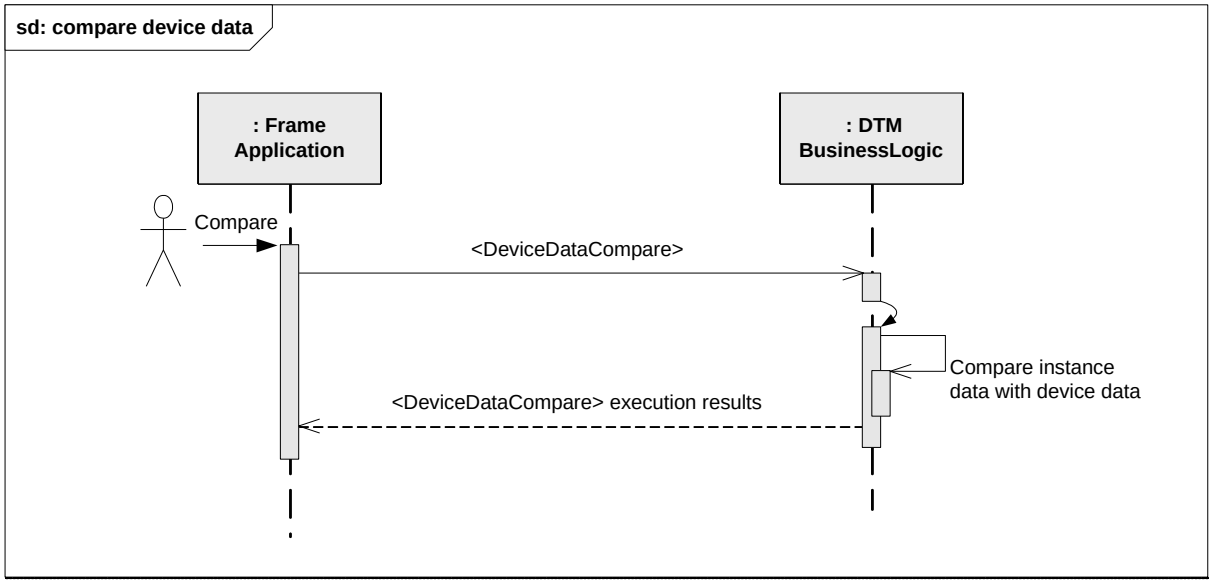
Figure 188 — Read/write device data

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8.16 Comparing data

8.16.1 Comparing device dataset and instance dataset

In order to compare the data of a DTM instance with the data of the respective device, the action <DeviceDataCompare()> (defined 5.15.2) is executed (see Figure 189).



Used methods:
IComparison.BeginDeviceDataCompare() / IComparison.EndDeviceDataCompare()

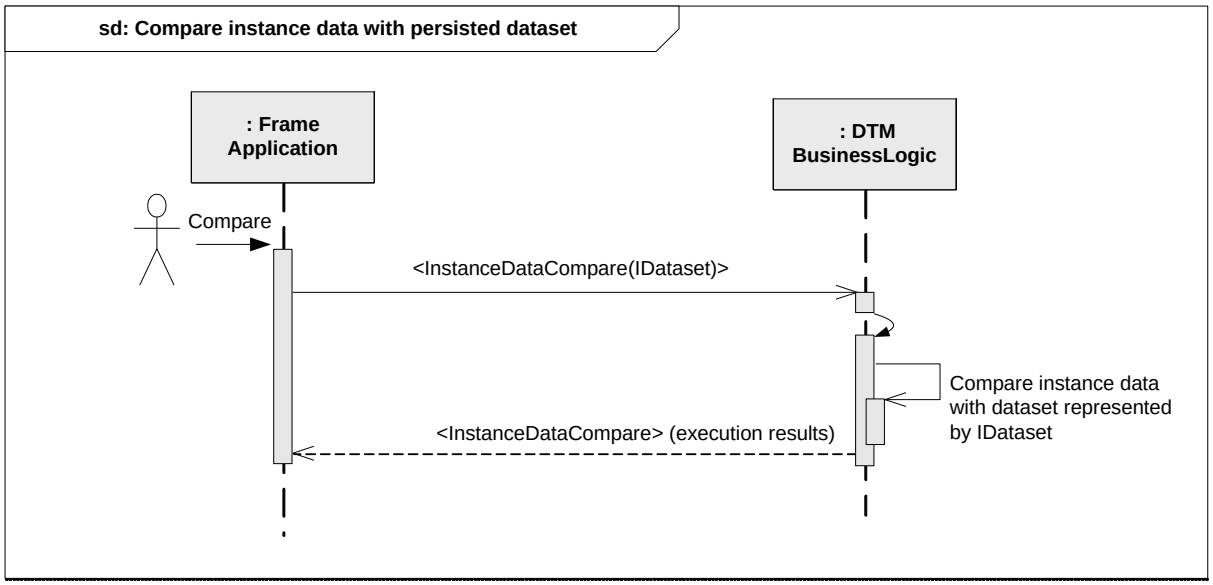
Figure 189 — Comparing device dataset and instance dataset

The comparison is executed for the data in the DTM dataset and the data that can be uploaded from the device. The comparison should include all identification, configuration, and parameterization data. Dynamic data and status data should not be included in the comparison.

8.16.2 Comparing different instance datasets

In order to compare the data of one DTM instance with the data of a different DTM instance, the action <InstanceDataCompare()> is executed (see Figure 190).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IComparison.BeginInstanceDataCompare() / IComparison.EndInstanceDataCompare()

Figure 190 — Compare instance data with persisted dataset

8.17 Reassigning a different DtmDeviceType at a device node

8.17.1 General

Over the lifetime of the FDT Frame Application project it can be necessary to reassign the DtmDeviceType of a device node to a different DtmDeviceType (see Figure 193). Reasons for the reassignment may be:

1. An engineer reassigns a DtmDeviceType during offline planning of the FDT topology.
2. A DTM is available which supports the same device type better than the currently assigned DTM (for instance instead of a Generic DTM, a specific DTM can be assigned).

Note: The DTM of the DtmDeviceType may be updated or upgraded. If the device of the device node is unchanged, a reassignment is not required due to FDT rules regarding DTM replacing installations for DTM Updates and DTM Upgrades (See chapter 10).

3. A physical device was or is going to be exchanged. This means, the device which is logically connected to a device node in the FDT topology will be replaced. The replacement may require a reassignment of the DtmDeviceType if the DtmDeviceType, which is currently in use, does not support the new device type or the version of the new device.

Note: Relevant is the identification of the device firmware. A device replacement as well as a firmware update can be incompatible in respect to the DtmDeviceType.

In regard to cases 1 and 2: Do not consider scanned information from a connected device. Usually, an existing dataset cannot be migrated in these cases.

Chapter 8.17.2 describes the scenario, where a DTM detects that the device type of the connected device can be better supported by a different DtmDeviceType.

Chapter 8.17.3 and 8.17.4 show sequence diagrams explaining the steps in relation to use case 3(device exchange).

Whenever a DtmDeviceType is reassigned, two post conditions need to be considered:

- Device support:

  The new DtmDeviceType shall be able to operate the device connected to the device node (refer to lifecycle concept regarding evaluation of device support in advance).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

- Dataset support:

  Dependent on the dataset format support of old and new DtmDeviceType, the existing dataset could be migrated to the new DtmDeviceType. The dataset migration is not possible in all cases. If a migration is not possible, the existing dataset cannot be used by the new DtmDeviceType. Frame Applications are responsible to inform the user about this and propose following action: An upload should be performed with the new DtmDeviceType in order to synchronize and store the device data with the project data.

Note: In general all descriptions in this chapter do not only apply to DtmDeviceTypes, but also apply to the two other DtmTypes: DtmModuleTypes and DtmBlockTypes.

8.17.2 DTM detects a change in connected device type

This section describes the scenario, where a DTM detects that the device type of the connected device can be better supported by a different DtmDeviceType.

3 possible scenarios are shown in the sequence diagram:

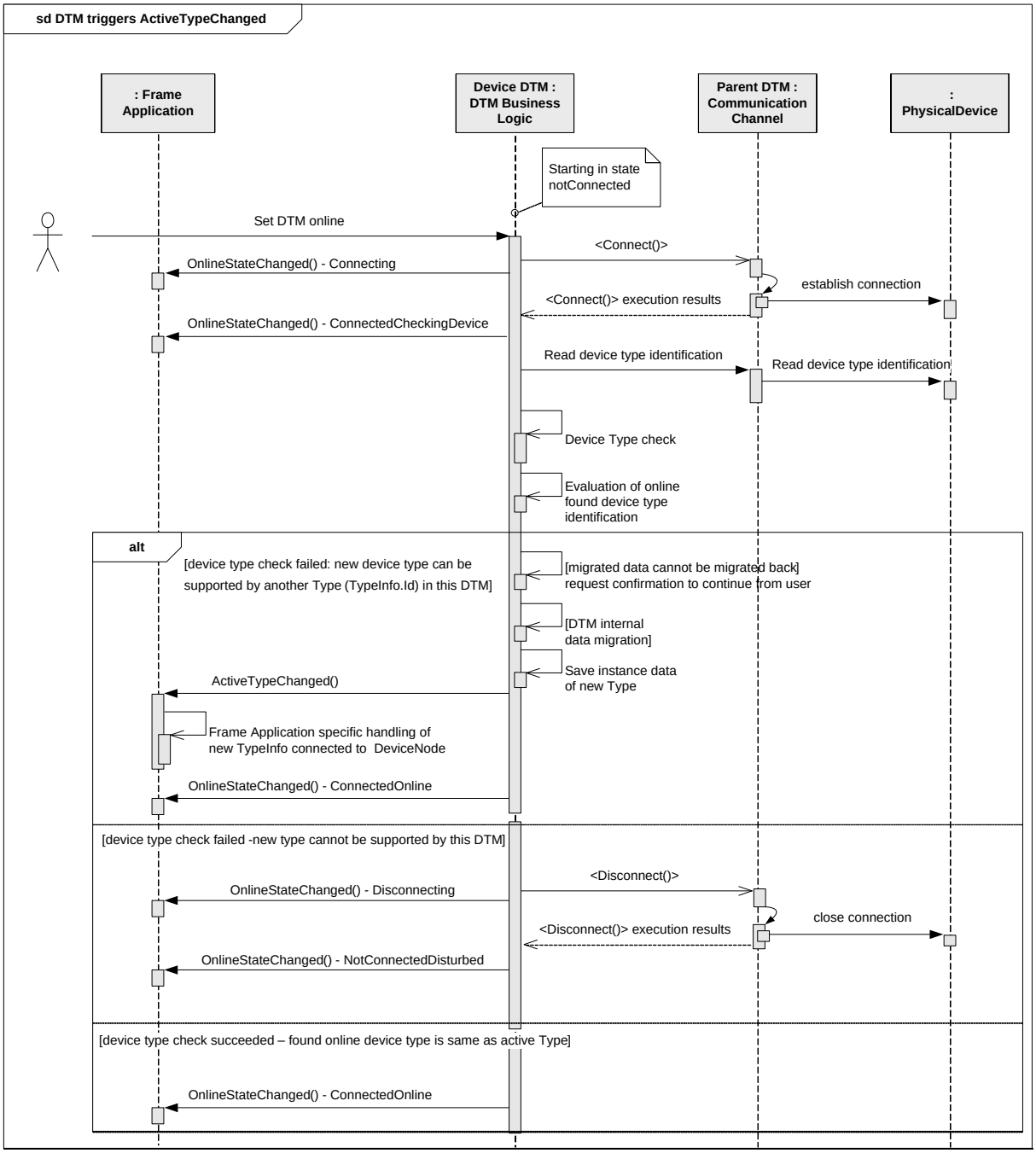
a) The connected device type can be better supported by a different DTM Type. In this case the DTM internally activates another DTM Type and informs the Frame Application with ActiveTypeChanged about the change.

b) The connected device type cannot be supported by the DTM

c) The unchanged connected device type: DtmDeviceType is not changed.

Scenario a) may occur if a DTM connects to the device again, after the device has been replaced by a compatible device type. Also Scenario a) may occur when the DTM was assigned with a generic DtmDeviceType to the device (e.g. during offline engineering) and detects that it can provide better support for the connected device with a different DtmDeviceType.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

Event IDtm3.OnlineStateChanged()
ICommunication.BeginConnect() / ICommunication.EndConnect()
ICommunication.BeginCommunicationRequest()
ICommunication.EndCommunicationRequest()
ICommunication.BeginDisconnect() / ICommunication.EndDisconnect()
Event IDtm3.ActiveTypeChanged()

Figure 191 — DTM triggers ActiveTypeChanged event

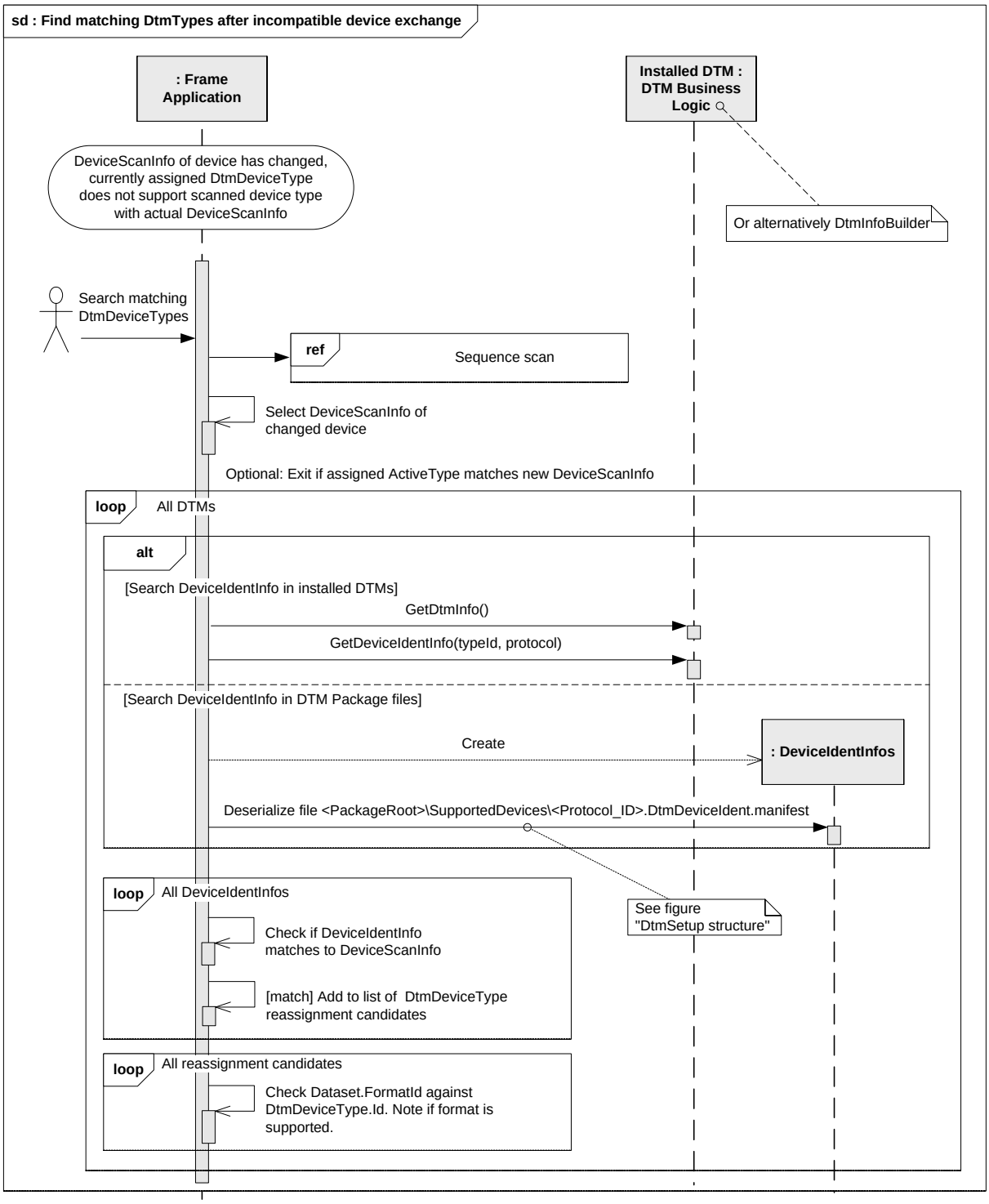
Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8.17.3 Search matching DtmDeviceTypes after incompatible device exchange

After a device exchange, a Frame Application should support the verification of the DtmDeviceType currently assigned to a device node in the FDT topology. In addition to the identification of the device types supported by installed DTMs, FDT provides a concept to explore DTM package files and the included DTMs before the DTMs are installed (see chapter 10). This can be used to find out if there are DTMs available (uninstalled DTMs), which include DtmDeviceTypes to support a scanned device.

The following sequence diagram (Figure 192) shows, how a list of matching DtmDeviceTypes in installed DTMs and DTM package files can be determined by a Frame Application.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

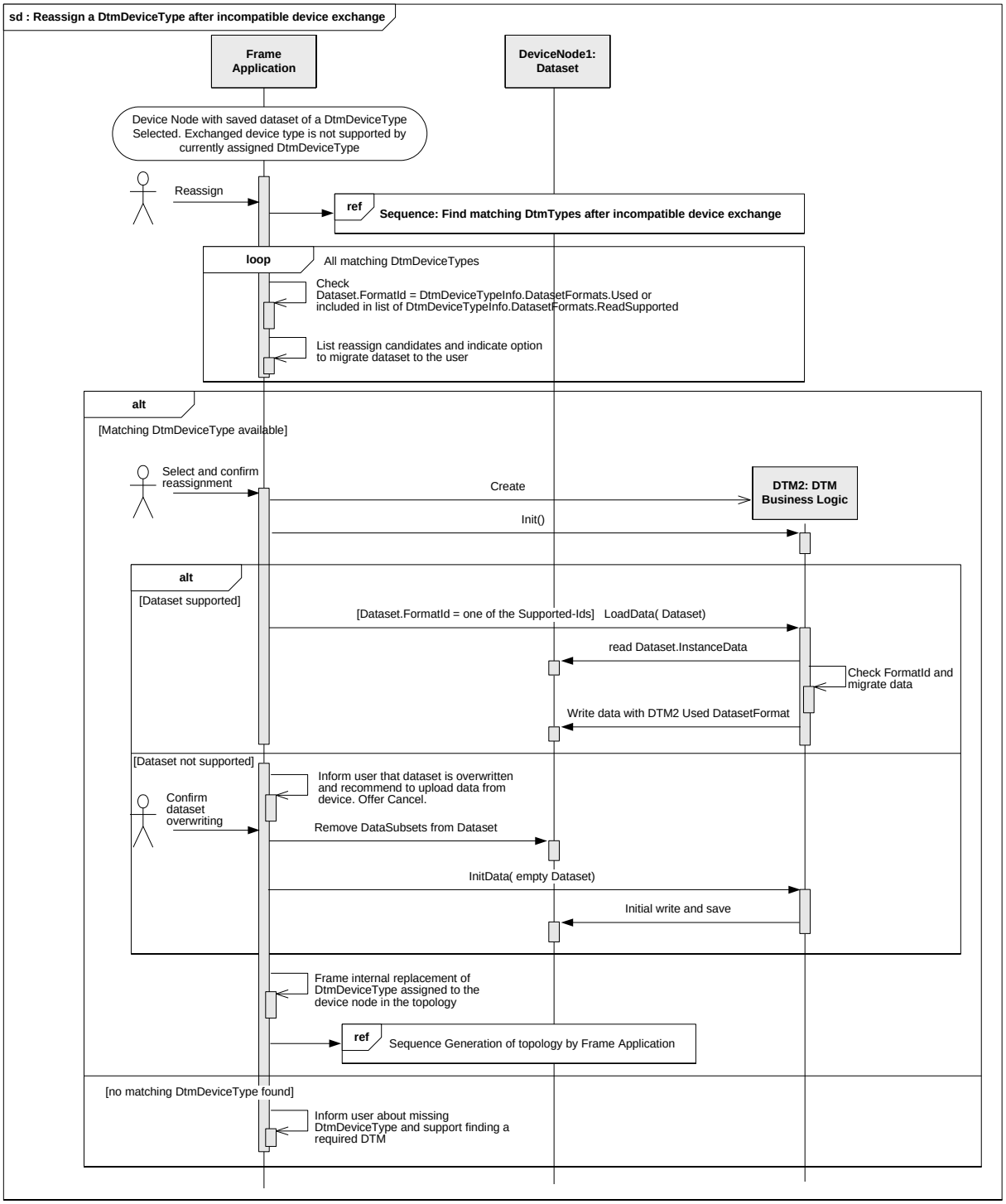
IDtmInformation.GetDtmInfo()
IDtmInformation.GetDeviceIdentInfo()

Figure 192 — Find matching DtmDeviceTypes after incompatible device exchange

8.17.4 Reassign DtmDeviceType after incompatible device exchange

The following sequence diagram (Figure 193) shows how a Frame Application verifies the validity of a currently assigned DtmDeviceType after a device change. The sequence diagram describes the DtmDeviceType reassignment if a better matching or newer DtmDeviceType is found.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

IDtm3.Init()
IDtm3.InitData()
IDtm3.LoadData()
IDataSubset.ReadData()
IDataSubset.WriteData()
IDtmInformation.GetDtmInfo()
IDtmInformation.GetDeviceIdentInfo()

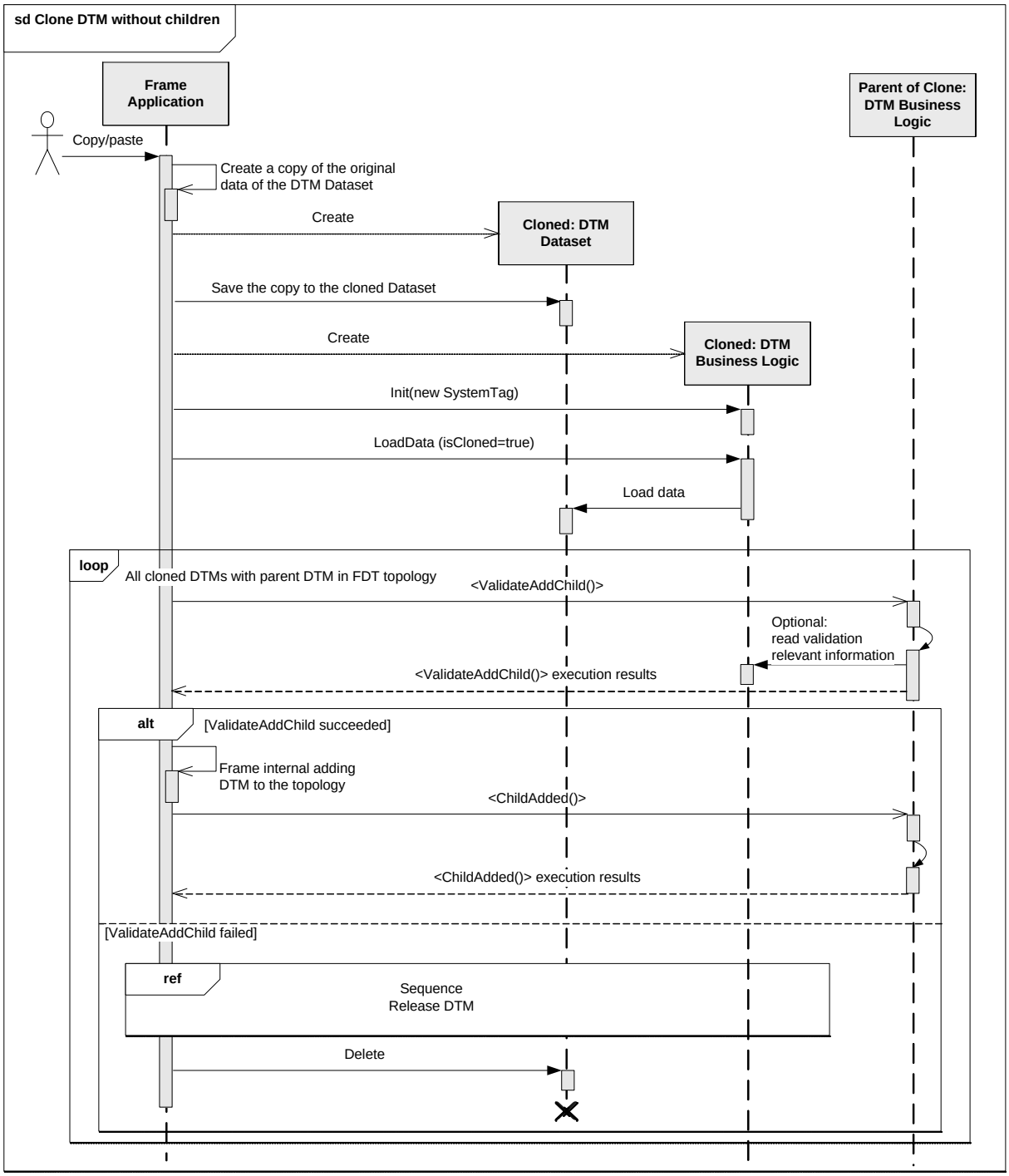
Figure 193 — Reassign a DtmDeviceType after incompatible device exchange

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8.18 Copying part of FDT Topology

8.18.1 Cloning of a single DTM without Children

A Frame Application might provide functionality to copy and paste a DTM which has no children to the same parent or to another one. Figure 194 shows the workflow for this functionality.



Used methods:

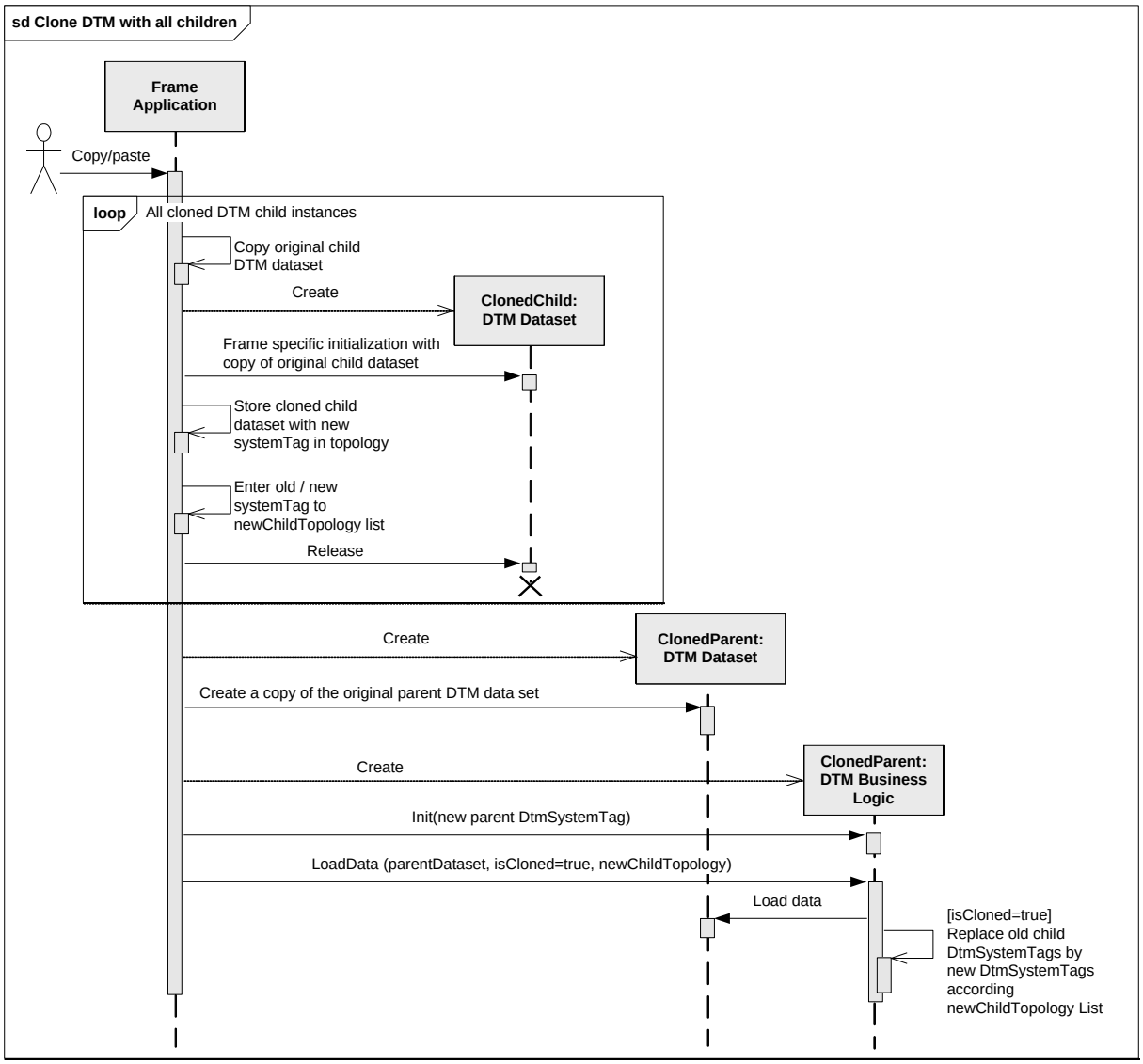
IDtm3.LoadData()
IDtm3.Init()
ISubTopology.BeginValidateAddChild()
ISubTopology.EndValidateAddChild()
BeginChildAdded() / EndChildAdded()

Figure 194 — Clone DTM without children

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8.18.2 Cloning of a DTM with all its Children

A Frame Application might provide functionality to copy and paste a DTM with all its children to the same parent or to another parent. Figure 195 shows the workflow of this functionality.



Used methods:

IDtm3.LoadData()
IDtm3.Init()

Figure 195 — Clone DTM with all children

8.19 Sequences for audit trail

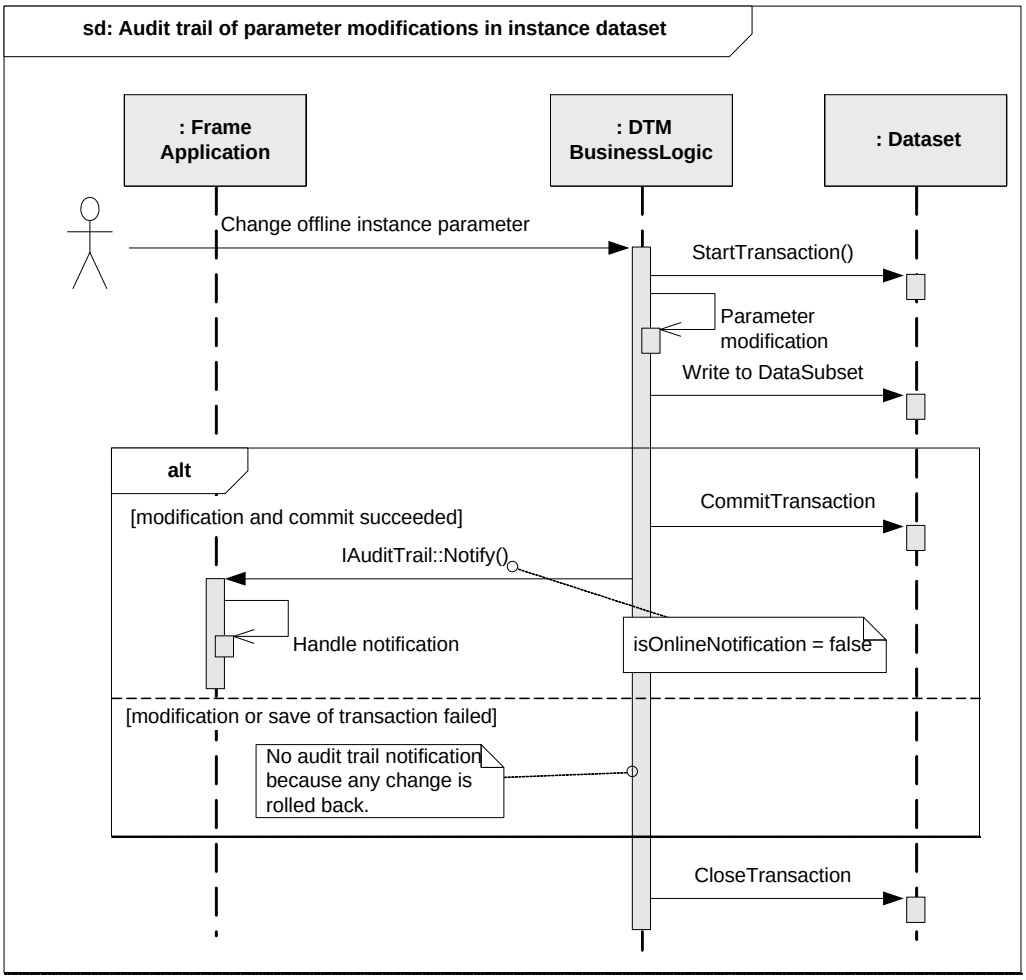
8.19.1 General

This section shows how the audit trail concept (described in 4.18) is implemented.

8.19.2 Audit trail of parameter modifications in instance dataset

Figure 196 shows how changes in the instance dataset are traced.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



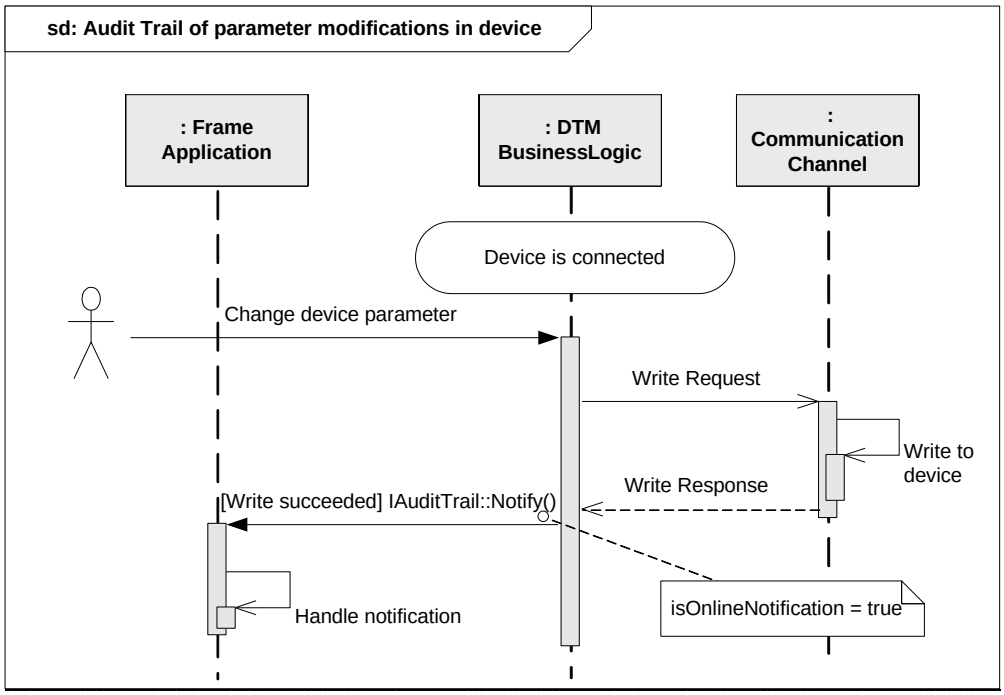
Used methods:
IAuditTrail.Notify()

Figure 196 — Audit trail of parameter modifications in instance dataset

8.19.3 Audit trail of parameter modifications in device dataset

Figure 197 shows how changes in the device data are tracked.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

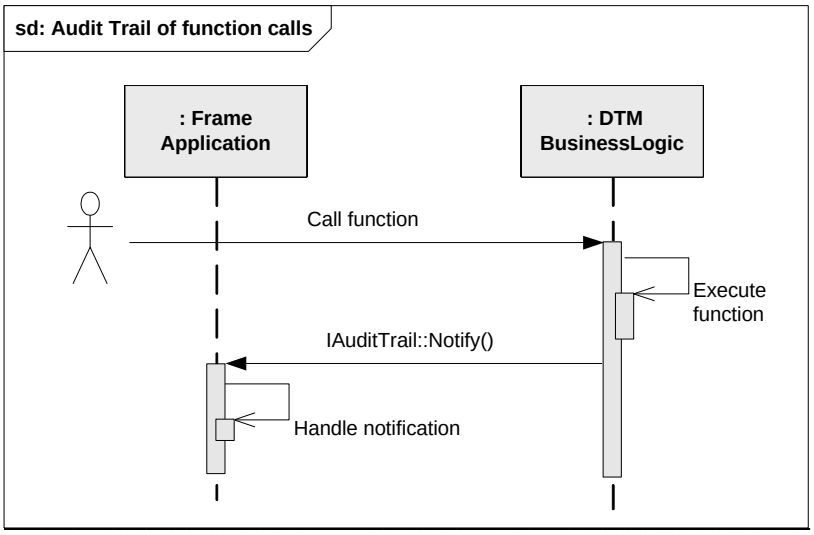


Used methods:
IAuditTrail.Notify()

Figure 197 — Audit trail of parameter modifications in device

8.19.4 Audit trail of function calls

Figure 198 shows how function calls are tracked.



Used methods:
IAuditTrail.Notify()

Figure 198 — Audit trail of function calls

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8.19.5 Audit trail of general notification

This shall only be used in case it is not a Function Notification or a Parameter Change Notification. General notifications are used by the DTM to provide audit trail notifications in the scenarios like device state information updates.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

9 Installation

9.1 General

This chapter describes the installation of FDT core assemblies, FDT communication protocols and DTMs as well as the structure and rules for creating DTM packages.

9.2 Common rules

9.2.1 Predefined installation paths

This chapter defines the common installation paths where FDT core assemblies, FDT protocol assemblies and DTMs are installed and registered (see Table 45 and Figure 199).

Table 45 — Predefined FDT installation paths

Path Name	Value	Description
<NuGet Repository>	<FDT3>\NuGet-Packages	Path of the FDT-specific NuGet repository (different from global-packages).
<CommonApplication Data>	(OS specific)	The directory for application data is also used to store the executable binaries. This is the known folder for .NET Core.
<FDT3>	<CommonApplicationData>\FDT3	Root folder for the FDT3 DTMs and protocols.
<FDT_Protocols>	<FDT3>\NuGet-Packages	Folder contains the communication protocol manifest files and protocol assembly files. Frame Applications search this folder for installed communication protocols.
<FDT_DTMs>	<FDT3>\DTMs	Folder contains all DTM components (e.g. manifest files and assemblies). Frame Applications search this folder for installed DTMs in order to create a device catalog.
<DTM_root>	<DTM>\<Vendor>.<PackageName>.<PackageVersion> NOTE 1 <Vendor> is the name of the DTM vendor. NOTE 2 <PackageName> is the name of the DTM Package. NOTE 3 <PackageVersion> is the 4-component version of the DTM Package. Example ”FDT_Group.DemoDTM.1.0.0.0”.	The folder contains all DTM binaries (assemblies) and data files. This includes main DTM BL assembly, DTMInfoBuilder assembly, resource assemblies, DTM WebUI container files and other files.
<FDT_GUIs>	<DTM_root>\UserInterfaces	Folder contains the user interface manifest files and WebUI container files. Frame Applications search this folder for installed user interfaces.
<FDT_Conformity>	<DTM_root>\ConformityRecords	The folder <FDT_Conformity> contains the respective conformity record and certification files for the DTM.

Note:
<CommonApplicationData> is the known folder for .NET Core (see [26][27]). It is the folder returned by the System.Environment.GetFolderPath() method for the special folder ID “System.Environment.SpecialFolder CommonApplicationData“.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

The FDT core assemblies (interfaces, datatypes, and exceptions) and communication protocol assemblies are installed in the NuGet Repository. Shared .NET assemblies may also be installed in the NuGet Repository. Please refer to chapter 5.6.4 when using shared .NET assemblies.

All communication protocols shall be registered with their manifest and installed with their assemblies in the predefined path <FDT_Protocols>.

All DTMs shall be registered with their manifest and installed with their assemblies in the predefined path <FDT_DTMs> (see Figure 199).

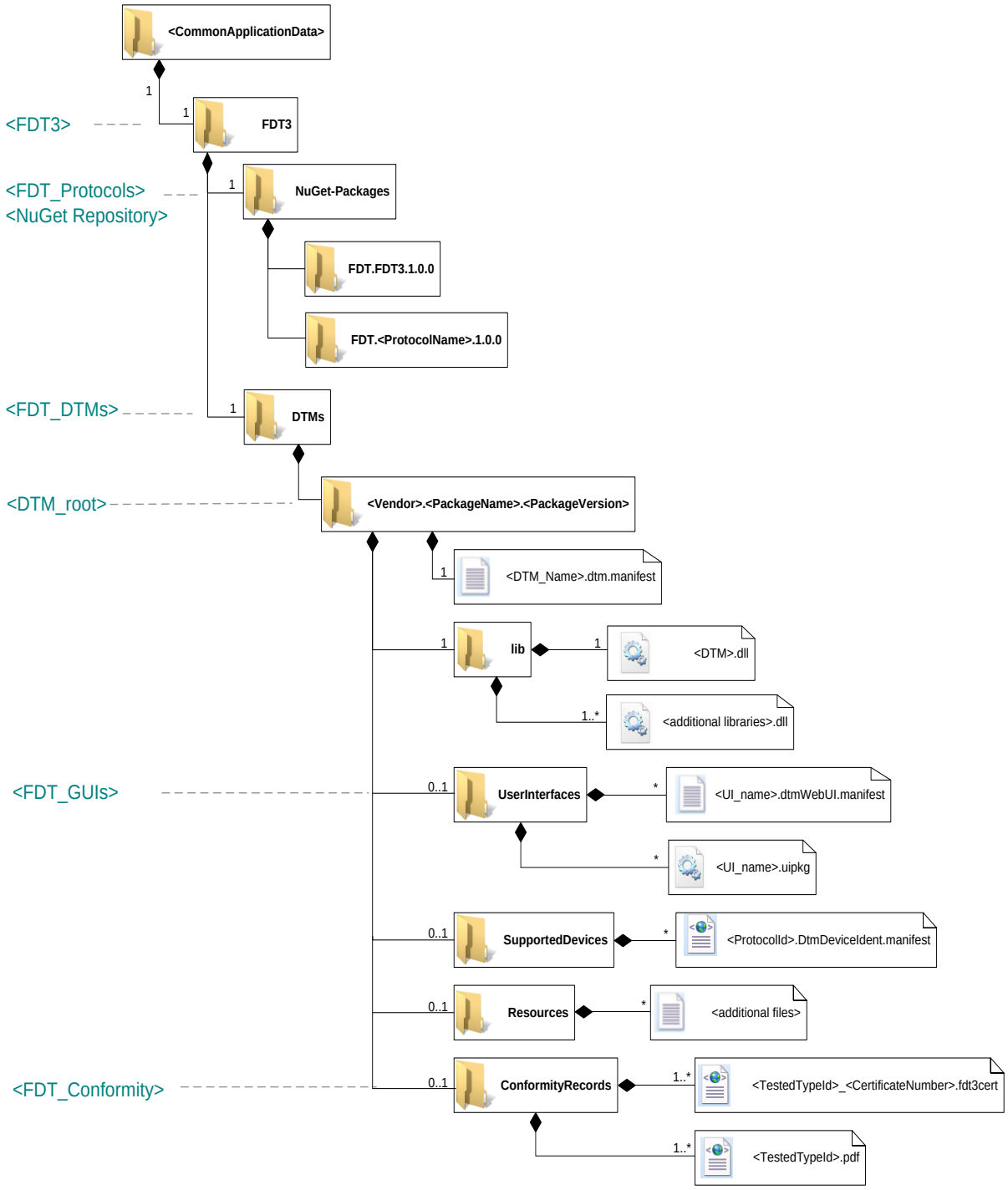


Figure 199 — Installation paths (with example DTM)

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

9.2.2 Predefined web-server paths

This chapter defines the common paths related to deployment via web-server (see Table 46 and Figure 199).

Table 46 — Predefined web-server paths

Path Name	Value	Description
<document root>	(web-server specific)	The document root is a directory (a folder) that is stored on the host of the web-servers. The folder is designated for holding web pages. In order for a WebUi to be accessible to web-browsers, it must be published to the "document root."
<FDT_CommonFiles >	<document root>/FDT_CommonFiles	Folder providing common files related to FDT WebUIs.
<WebDataConnector.js>	<FDT_CommonFiles>/WebDataConnector.js	JavaScript file providing the WebDataConnector. – The file is a server-specific file, providing the JS methods as defined for IDtmWebUiMessaging in E.4.

9.2.3 Manifest files

All components related to FDT provide manifest files in order to register the component in the FDT system (except for the FDT core assemblies) or to provide pre-installation information. Manifest files are XML files, which follow a defined format. The format corresponds to the .NET datatypes, which are part of the FDT core specification.

9.2.4 Paths in manifest files

All manifest files may include paths to assembly or resource files.

If some parameter in any manifest file represents a path to an assembly, icon, bitmap, documentation or data file, then it shall be relative to the component root path.

NOTE For example, if a PDF document is provided for a DTM, which is installed in “<FDT_DTMs>\Vendor1.MyDtm.1.0.0”:
- The document is located in “<FDT_DTMs>\Vendor1.MyDtm.1.0.0\Resources\Documentation\help.pdf”, the reference to the document is “Resources\Documentation\help.pdf”.
- The component is the DTM BL. The component root path is “<FDT_DTMs>\Vendor1\MyDtm.1.0.0”.

In order to access the referenced files, the Frame Application shall add the component root path at the beginning of relative paths.

9.2.5 DTM Installation package format

DTM installation packages are provided in form of DTM package files (extension ‘.dtmpkg’). A DTM package file shall conform to the Open Packaging Conventions(OPC) [32][33].

The DTM installation package references NuGet packages that are required for execution of the DTM. A DTM shall reference the required FDT3 core assembly package as well as the required protocol package(s). The Frame Application is responsible for retrieving and installing the required NuGet packages.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

9.2.6 Digital signatures of package components

The open packaging convention (OPC) supports a feature to sign and validate the parts of an OPC package or the entire package. Frame Applications shall use signature validation for detecting when the content of an OPC package has been altered after the file was signed. If a DTM package file has been changed, this will be detected by the Frame Application during the installation process. Authors of DTM package files shall adhere to the following rules in order to ensure that all Parts of an installation are covered by a digital signature:

- All resources for an installation shall be embedded in the DTM package file. All Parts of the package, including the signature origin part shall be signed by the originator of the package.

Installation packages provided by FDT Group (e.g. for core assemblies and protocol-specific files) will be signed accordingly.

9.3 Installation of FDT core assemblies

FDT core assemblies (see 5.1) are provided as NuGet package (called ‘core assembly package’) and shall be installed by each Frame Application into a common location (see <NuGet Repository> in 9.2.1).

The Frame Application executes the installation of the DTMs (see 9.5). During the installation of a DTM, the Frame Application is responsible to add the FDT core assemblies, as well as the protocol assemblies, to the common location.

During runtime, the Frame Application shall browse the common location and load the highest available version of the FDT assemblies into the process.

DTMs shall reference the FDT assembly packages within their Package Manifests (see 9.6.8).

9.4 Installation of communication protocols

9.4.1 General

All DTM package files shall reference the supported or required communication protocols. The protocol assemblies (see 5.7.10) shall be provided as NuGet packages (called ‘protocol package’) and shall be installed by the Frame Application in the FDT-specific NuGet repository (see Figure 200).

Protocols defined by an FDT Protocol Annex are provided as NuGet package files by the FDT Group. Vendor-specific communication protocols are provided as separate NuGet package files together with the corresponding DTM package. The protocol packages shall be referenced by the DTM package manifest (see Dependencies in 9.6.8).

During runtime, the Frame Application shall browse the common location and load the highest available version of the protocol assemblies into the process when needed.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

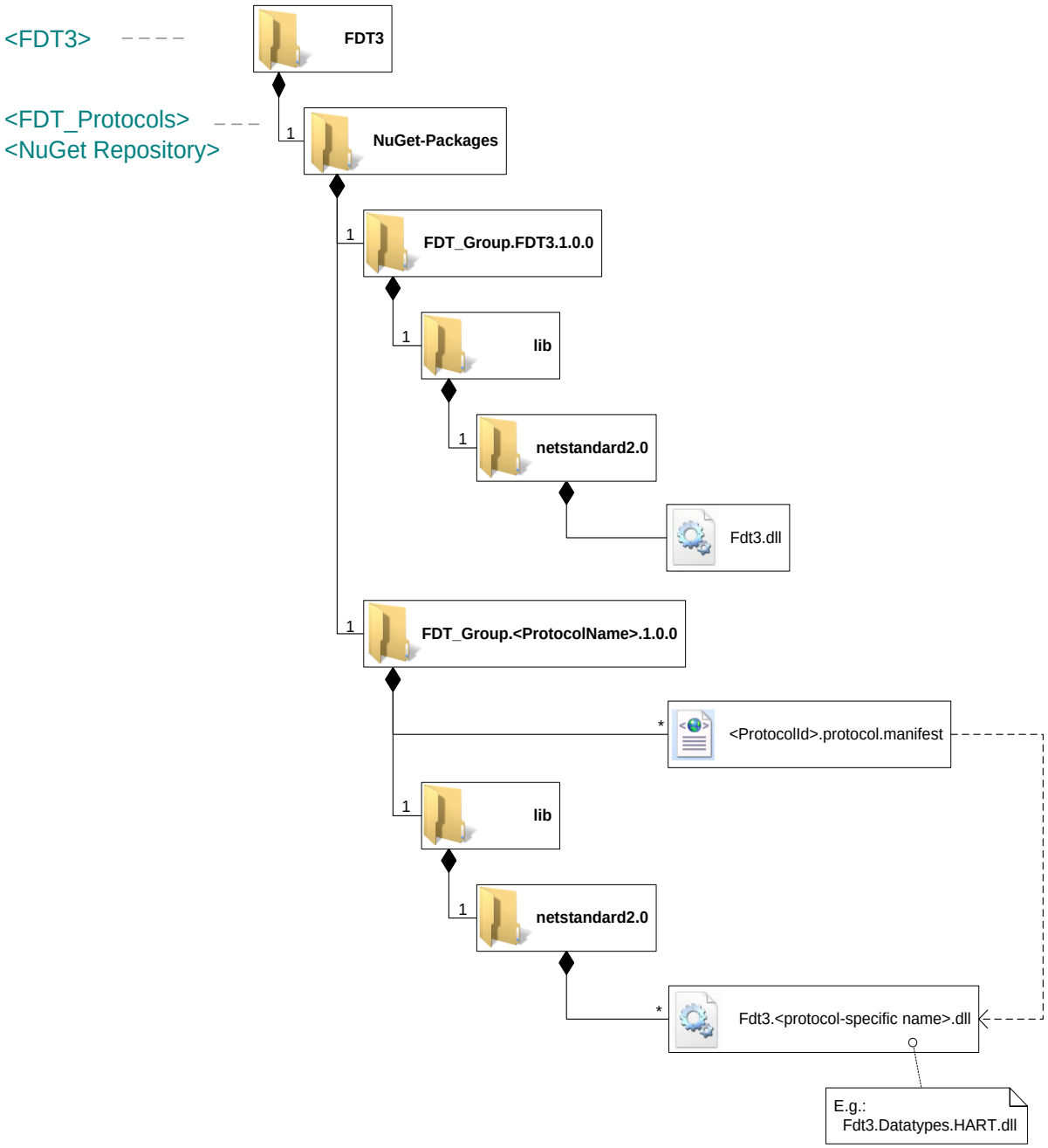


Figure 200 — Installed FDT and protocol assemblies

9.4.2 Registration

Communication protocols are registered by protocol manifest files that are installed in the <FDT_Protocols> path. A protocol manifest file describes a communication protocol with its ID and assembly reference.

9.4.3 Protocol manifest

A protocol manifest is used to register additional communication protocol assemblies in the system in order to enable Frame Applications and DTMs to find it. Protocol manifest files shall be installed in the <FDT_Protocols> path (see 9.2.1). The file name shall be composed of the unique communication protocol ID and the suffix ".protocol.manifest".

A protocol manifest xml file contains following information:

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

- AssemblyInfo: Information about the protocol assembly that contains the communication protocol classes and data structures.
- ProtocolId: Unique identifier of the protocol (as UUID).
- ProtocolName: Human readable name of the protocol

Figure 201 shows an example for a protocol manifest.

```
<?xml version="1.0" encoding="utf-8"?>
<ProtocolManifest xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/Fdt.Deployment">
  <ProtocolId>b803f1b4-d992-44bc-a62d-08ec71b0b4cd</ProtocolId>
  <ProtocolName>XyzBus</ProtocolName>
  <AssemblyInfo>
    <Name>Fdt3.XyzBus</Name>
    <Version xmlns:d3p1="http://schemas.datacontract.org/2004/07/System">
      <d3p1:_Build>0</d3p1:_Build>
      <d3p1:_Major>1</d3p1:_Major>
      <d3p1:_Minor>0</d3p1:_Minor>
      <d3p1:_Revision>0</d3p1:_Revision>
    </Version>
    <PublicKeyToken>1234567890123456</PublicKeyToken>
    <SupportedNetStandardVersions>
      <TargetNetStandard>
        <NetStandardVersion xmlns:d5p1="http://schemas.datacontract.org/2004/07/System">
          <d5p1:_Build>-1</d5p1:_Build>
          <d5p1:_Major>2</d5p1:_Major>
          <d5p1:_Minor>0</d5p1:_Minor>
          <d5p1:_Revision>-1</d5p1:_Revision>
        </NetStandardVersion>
      </TargetNetStandard>
    </SupportedNetStandardVersions>
    <SupportedTargetPlatform>Any</SupportedTargetPlatform>
    <Path i:nil="true" />
  </AssemblyInfo>
</ProtocolManifest>
```

Figure 201 — Example: Protocol manifest

9.5 Installation of DTMs

9.5.1 General

Prior to installation of a DTM, it is possible to retrieve information about a DTM from the respective DTM Package Manifest (see 9.6.3).

After installation the respective information can be retrieved within the FDT system (see 9.7).

All DTMs shall be installed in the predefined < FDT_DTMs> path (see 9.2.1). For each installed DTM a subfolder <Vendor Name>.<PackageName>.<PackageVersion> (same name as the name of the DTM package file) is created. Each DTM is placed in such a subfolder. The path to this folder is the <DTM_root> path. All DTM assemblies and data files for a single DTM installation shall be installed in this path. Following components of a DTM are installed here:

- DTM BL assembly (see DTM component in Figure 199). It implements the main DTM Business Logic. This assembly can use or reference some other dependent assemblies. There are no limitations on file names.
- DTM WebUI Package Files (see DtmWebUI component in Figure 199 and see 9.5.5). This component provides one or more DTM WebUIs. There are no limitations on file names.
- DTM Information Builder assembly (see DtmInfoBuilder concept in 4.4.2). This component implements support for getting dynamic DTM information (e.g. for DD-Interpreter DTMs). There are no limitations in regard to file name. The DtmInfoBuilder can be integrated with the main DTM Business Logic assembly.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

- DTM Conformity Records (see 5.18.5): If the DTM has been certified, then a subfolder "ConformityRecords" provides all conformity record files providing details about the FDT compliance certification of the DTM.

9.5.2 Registration

The Frame Application shall be able to retrieve information about installed DTMs and supported device types. The registration of DTMs is performed using manifest files that are installed in the predefined installation paths (see 9.2.1).

The <FDT3> folder and its subfolders contain all manifest files that describe the installed DTMs and their DTM WebUIs. Following manifest files are defined:

- *.dtm.manifest.
  This xml file describes an installed DTM (see chapter 9.5.3).
- *.dtmwebui.manifest.
  This xml file describes an installed DTM WebUI (see 9.5.4).

A Frame Application searches for information about installed DTMs in three steps (see Figure 202):

1. The Frame Application reads all DTM manifest files in < FDT_DTMs> path.
2. For each DTM, the Frame Application loads the referenced assembly and starts the DtmInfoBuilder. Further information about device-, block-, module types, supported protocols etc. is retrieved from the DtmInfoBuilder.
3. For each DTM, the Frame Application reads the DTM WebUI manifest files (referenced in *.dtm.manifest) and stores user interface assembly and function information (e.g. in a device catalogue).
4. After a DTM is found, the DTM Business Logic and the respective DTM WebUI may be executed.

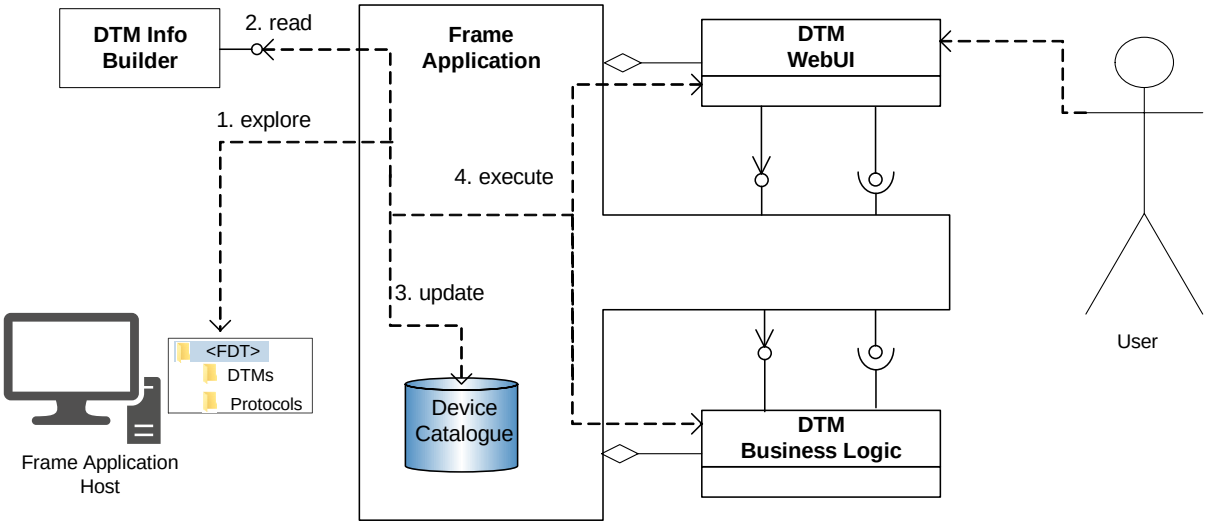


Figure 202 — Search for installed DTMs

See the descriptions of DTM manifest and DTM WebUI manifest datatypes for details about different manifest files that are used for DTM registration.

A Frame Application can check if new DTMs are registered by comparing the date of the DTM Manifest file with the last checking date.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

9.5.3 DTM manifest

A DTM manifest file is used to register a DTM in the system in order to enable Frame Applications to find it. Therefore it contains references to the main DTM BL assembly and to the class that implements the DtmInfoBuilder for the DTM. DTM manifest files shall be copied to the vendor-specific subfolder of the <FDT_DTMs> path by the DTM package file during installation or during update of the DTM. The file name is composed by a unique DTM name and the suffix ".dtm.manifest". A DTM vendor is responsible for the uniqueness of his DTMs and DTM names within the vendor-specific name space.

A DTM manifest file (DtmManifest) contains following information:

- DynamicClassReference: Information about DtmInfoBuilder class, which shall be used to request TypeInfos and corresponding device identification information supported by the DTM.
- DtmRootPath: Root installation path of the DTM (relative path from common FDT installation path). The DTM main assembly and DtmInfoBuilder assembly (if provided) is located in this path.
- DtmInitData: [Optional] DTM initialization information. This string is passed to the DTM in the IDtm3.Init call.

Note: Information about DTM device types is not included in the DTM manifest file.

DTM manifest files shall be created by using the DtmManifest datatype. See the description of DtmManifest datatype in 7.6.2 for further information about DTM manifest files.

Figure 203 shows an example for a DtmManifest.

```
<?xml version="1.0" encoding="utf-8"?>
<DtmManifest xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/Fdt.Deployment">
  <DtmRootPath>Company\XYZDtm</DtmRootPath>
  <DtmInfoBuilderRef xmlns:d2p1="http://schemas.datacontract.org/2004/07/Fdt.Dtm">
    <d2p1:AssemblyInfo>
      <Name>Company.XYZDtm.InfoBuilder</Name>
      <Version xmlns:d4p1="http://schemas.datacontract.org/2004/07/System">
        <d4p1:_Build>1</d4p1:_Build><d4p1:_Major>1</d4p1:_Major>
        <d4p1:_Minor>0</d4p1:_Minor><d4p1:_Revision>2231</d4p1:_Revision>
      </Version>
      <PublicKeyToken>1234567890</PublicKeyToken>
      <RuntimeVersions>
        <RuntimeVersion>
          <CLRVersionNumber xmlns:d6p1="http://schemas.datacontract.org/2004/07/System">
            <d6p1:_Build>0</d6p1:_Build><d6p1:_Major>4</d6p1:_Major>
            <d6p1:_Minor>0</d6p1:_Minor><d6p1:_Revision>0</d6p1:_Revision>
          </CLRVersionNumber>
        </RuntimeVersion>
      </RuntimeVersions>
      <SupportedPlatforms>Only32bit</SupportedPlatforms>
      <Path i:nil="true"/>
    </d2p1:AssemblyInfo>
    <d2p1:ClassName>Company.Dtm.XYZ.InfoBuilder.DtmInfoBuilder</d2p1:ClassName>
  </DtmInfoBuilderRef>
  <DtmCategory>DeviceDTM</DtmCategory>
  <UiManifestRefs>
    <UiManifestRef>
      <ManifestType>Fdt.Deployment.DtmWebUiManifest</ManifestType>
      <FileName>Company.XYZDTM.WebGui1.dtmwebui.manifest</FileName>
    </UiManifestRef>
  </UiManifestRefs>
  <DtmInitData i:nil="true"/>
</DtmManifest>
```

Figure 203 — Example: DtmManifest

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

9.5.4 DTM WebUI manifest

A DTM WebUI manifest file is used to register a DTM WebUI in the system in order to enable Frame Applications to find it. These files shall be copied to the <FDT_GUIs> path by the Frame Application during installation or update of the DTM. The file name is composed of the DTM WebUI name (unique for the DTM) and the suffix ".dtmwebui.manifest". Each DTM WebUI manifest file shall be referenced and declared in the DTM manifest file. The DTM WebUI manifest contains following information:

- ContainerFileInfo: Information about the DTM WebUI container file.
- WebUiFunctionInfos: Information about the DTM WebUI functions included in the container file described by ContainerFileInfo.

See the description of DtmWebUiManifest datatype in 7.6.3 for the syntax of DTM WebUI manifest files. Figure 204 shows an example for a WebUI manifest file.

```
<?xml version="1.0" encoding="utf-8"?>
<DtmWebUiManifest xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/Fdt.Deployment">
  <ContainerFileInfo>
    <Name>WebUI_Package_Name.uipkg</Name>
    <Path i:nil="true" />
  </ContainerFileInfo>
  <WebUiFunctionInfos>
    <WebUiFunctionInfo>
      <FunctionId>5</FunctionId>
      <StartPages>
        <StartPageInfo>
          <Language>en</Language>
          <Path>OfflineWebUi.html</Path>
        </StartPageInfo>
        <StartPageInfo>
          <Language>de</Language>
          <Path>OfflineWebUi_de.html</Path>
        </StartPageInfo>
      </StartPages>
      <InitData i:nil="true" />
    </WebUiFunctionInfo>
  </WebUiFunctionInfos>
</DtmWebUiManifest>
```

Figure 204 — Example: DtmWebUiManifest

9.5.5 WebUI container files

DTM WebUIs are provided with the DTM installation in form of WebUI container files (extension '.uipkg'). A WebUI container file shall conform to the Open Packaging Conventions(OPC) [32][33].

A WebUI container file shall contain all files and components related to the respective DTM WebUI. It is possible, that a WebUI container file contains more than one WebUI. Each WebUI is represented as 'Part' according to the packaging convention and shall have at least a StartPage (an HTML file). This StartPage may reference other HTML documents, JavaScript files and other HTML elements within the container file.

In order to support multiple languages, a WebUI part may provide a number of HTML5 files – one file for each language (see for example Figure 205). The information regarding the supported language is provided by means of the DtmWebUiManifest. For each StartPage the corresponding language is indicated by an language tag (see [34]). It is mandatory to provide at least a StartPage for neutral English (language code "en"). This Start Page shall be used by Frame Applications as a fallback in case the required language is not supported by the DTM WebUI.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

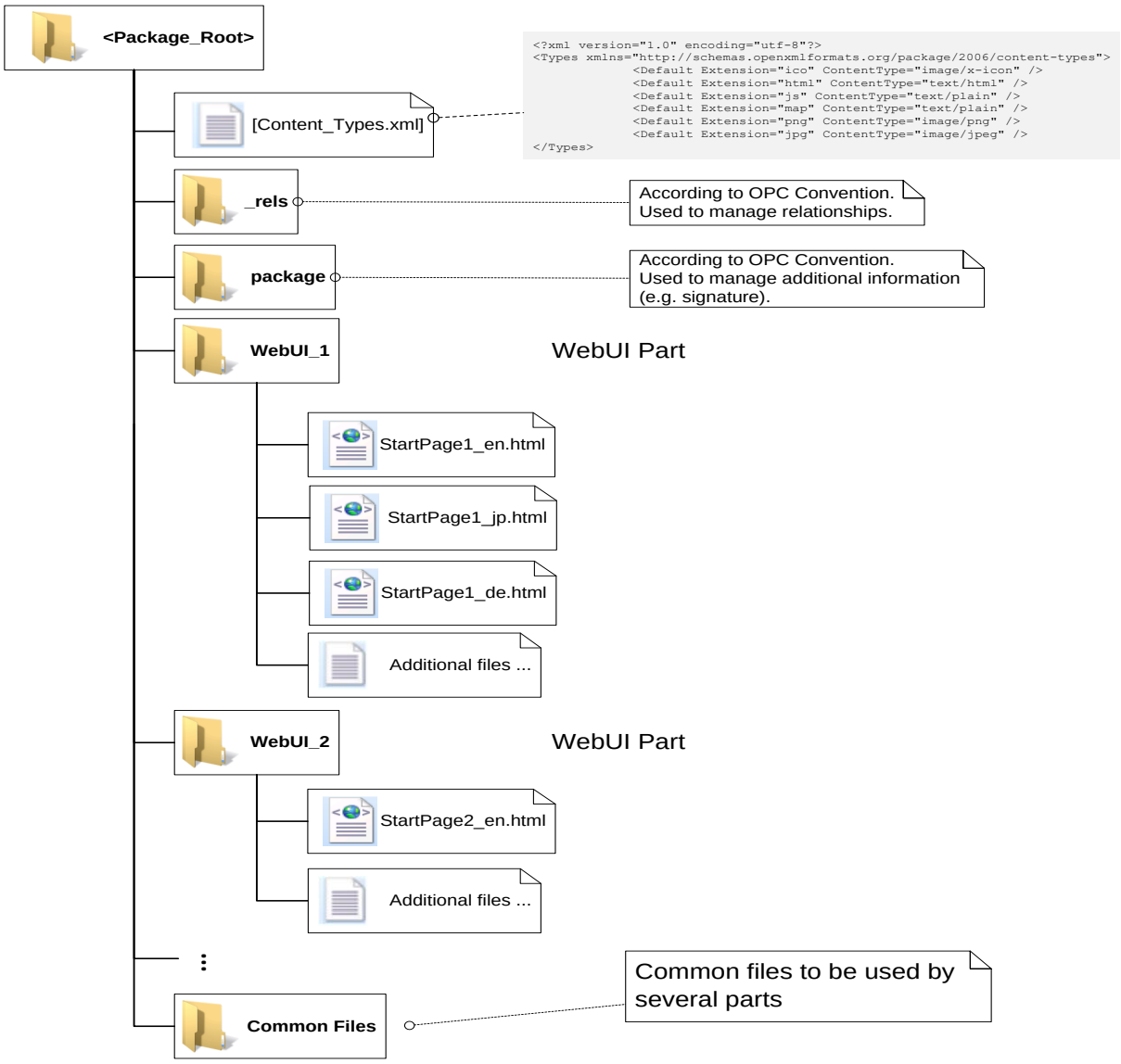


Figure 205 — Structure of a WebUI container file

In order to protect against tampering, the content of a WebUI package shall be signed.

9.6 DTM package file

9.6.1 Naming convention

The name of a DTM package file shall follow the convention of:

<Vendor>.<PackageName>.<PackageVersion>.dtmpkg

Example: Fdt.TestDTM.1.0.0.dtmpkg

The PackageVersion shall be provided with 4 version components, separated by the period character. The other parts of the file name (e.g. <Vendor>, <PackageName>) shall not contain the period character (“.”).

The <PackageVersion> shall reflect the version of the package, there is no relationship defined to the version of the DTM. It shall provide the same value as “Version” in the DtmPackageManifest.

Note: The version of the DTM also is documented in the DTM manifest.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

9.6.2 Structure (physical model)

The DTM package file structure defines the contents and structure of a DTM package file (see Figure 206).

The DTM package file is the main package file for one DTM. It contains the complete DTM and the WebUI container files. This file is used by the Frame Application to execute the installation.

Each DTM package file consists of following mandatory and optional Parts:

- Package manifest file [mandatory]. This file describes the package itself and contains information about included DTM. This file is always located in the package root folder. The DTM package manifest file shall be referenced by the package relationship “PackageManifest”.
- DTM manifest file [mandatory]. This file describes the DTM. This file is always located in the package root folder.
- Folder "lib" [mandatory]. This folder contains the runtime-specific sub-folders. This folder shall not contain any files, but only sub-folders.
  - Folder "any" [mandatory]. This folder contains the platform-independent DTM library and all related platform-independent libraries according 5.4.1.

NOTE1: The folder may contain DTM-specific sub-folders (e.g. for language resources)(see [36]).

  - Folders "{runtime_id}" [conditional]. If the package contains platform- or architecture-specific libraries (e.g. native drivers for Windows64 or Linux), then these folders shall be used to provide these libraries. The naming of the {runtime_id} is defined according [38]. [37] provides information about the “RID graph”, which describes the relationships between different {runtime_id}s.

NOTE2: Defining the naming of the {runtime_id} according [38] means that the RID graph will be considered during installation of the DTM assemblies, files may be distributed across multiple folders.

NOTE3: The combination of libraries from the any-folder and {runtime_id}-folders according to the rules described by the RID graph provides an executable and usable DTM.

- Folder “Supported Devices” [optional]. This folder contains the device identification files. These files describe identification information for device types that are supported by DTMs in this package. These protocol-specific files are always located in SupportedDevices_<DtmInfo.Id> folder. The files can be deserialized to initialize the class DtmDeviceIdentManifest.
- Folder “Resources” [optional]: This folder contains additional files relevant for the DTM (e.g. documentation, images). The content of this folder is DTM-specific, it may also contain additional sub-folders.
- Folder "User interfaces" [optional]. This folder contains the WebUI container files and the related DTM WebUI manifest files.
- Folder “ConformityRecords” [optional]: If the DTM has been certified, then a subfolder “ConformityRecords” provides all conformity record files providing details about the FDT compliance certification of the DTM.
- Documentation files [optional]: These files contain documentation relevant for the installation of the DTM (e.g. license, readme).

Additional parts (e.g. “_rels” and “package” folder) are added by signing the contents of the package.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

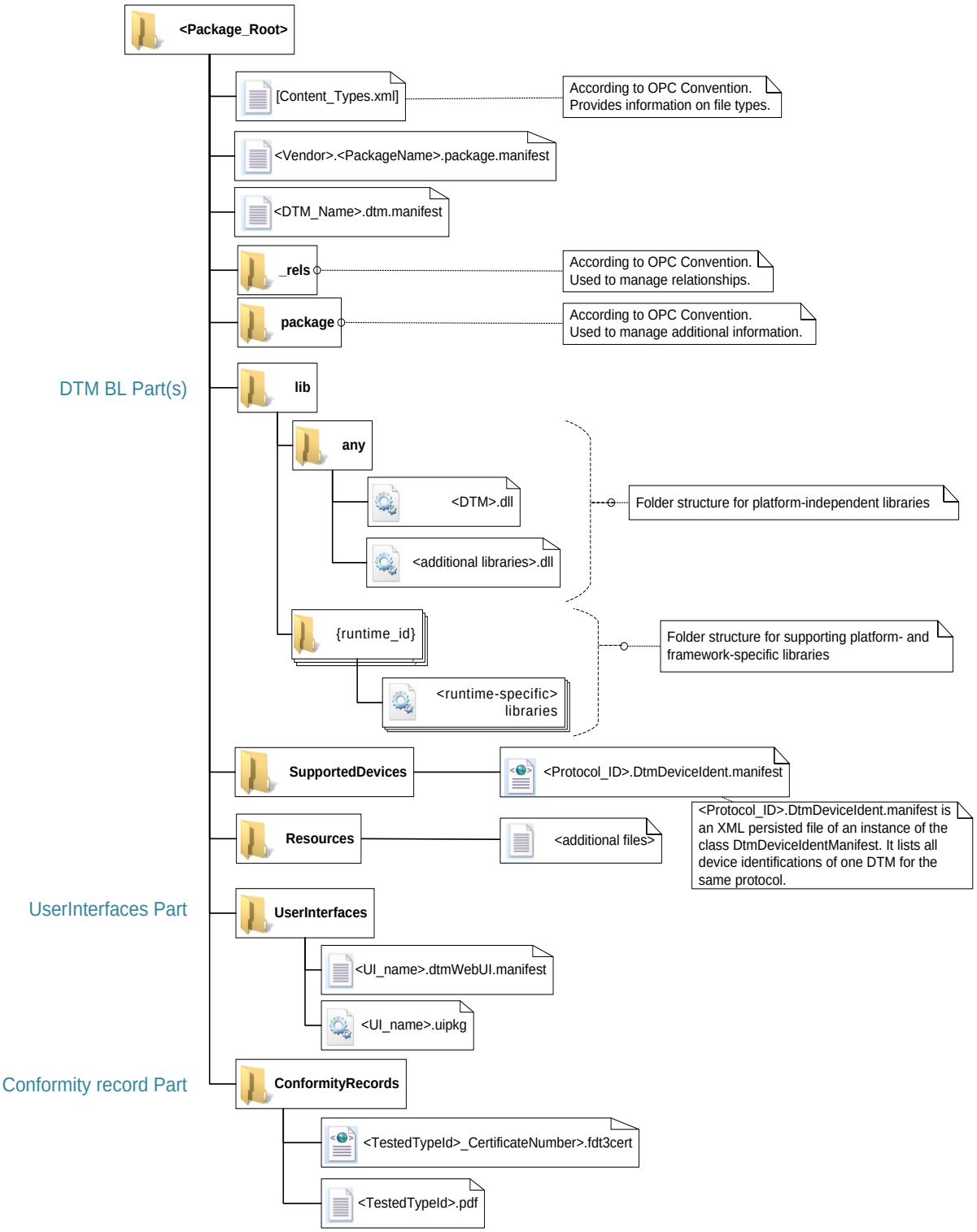


Figure 206 — DTM package file structure

9.6.3 Relationships (logical model)

The OPC Package convention intents to use the logical model to describe the package structure as a directed graph. This capability is only partially used with DTM Packages for:

- Signing the package contents
- Referencing the DTM package manifest file.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

The relationships supported in a DTM package are shown in Figure 207.

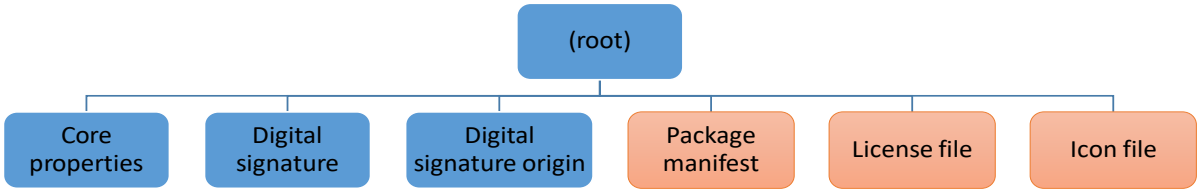


Figure 207 — DTM package file relationships

“Core properties”, “Digital signature” and “Digital signature origin” relationships are defined by OPC Packaging convention and shall be used accordingly.

The “Package manifest” relationship is used to locate the DTM package manifest file. It serves as starting point when installing the DTM.

The “License file” relationship is used to locate the license file (see 9.6.5).

The “Icon file” relationship is used to locate the Icon file (see 9.6.6).

Table 47 — Overview of relationships

Description	URI
Core properties	http://schemas.openxmlformats.org/package/2006/relationships/metadata/core-properties
Digital signature	http://schemas.openxmlformats.org/package/2006/relationships/digital-signature/signature
Digital signature origin	http://schemas.openxmlformats.org/package/2006/relationships/digital-signature/origin
Package manifest	http://www.fdtgroup.com/package/2019/relationships/package-manifest
License file	http://www.fdtgroup.com/package/2019/relationships/license
Readme file	http://www.fdtgroup.com/package/2019/relationships/readme
Icon file	http://www.fdtgroup.com/package/2019/relationships/icon

A DTM package may use additional relationships (e.g. thumbnail).

9.6.4 Core properties

The OPC packaging convention defines a set of core properties for a package file. Table 48 describes how the properties shall be used.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 48 — Overview of core properties

Property	Usage	Description
category	M	Always the value “DTM Package”
contentStatus	M	The status of the package. Allowed values are “Prototype”, “Release”, “Certified”.
contentType	O	The type of the package. Allowed values are “Initial”, “Update”.
created	M	Date of creation of the DTM package.
creator	M	The organization providing the package.
description	O	An explanation of the content of the package.
identifier	M	A GUID providing an unique identifier of the package. The value of this property shall have the same value as ProductCode in the DtmPackageManifest (see 9.6.8).
keywords	M	A set of keywords. Allowed keywords are: “CommDTM”, “DeviceDTM”, “GatewayDTM”, “ModuleDTM”, “BTM”
language	O	The language of the package.
lastModifiedBy	M	The organization who performed the last modification. This could be the name of the creator or “FDT Group”.
modified	M	Date on which the package was modified.
revision	O	The revision number.
Title	M	The name of the DTM package. The value here shall be the same value as provided for <PackageName> in the file name (see 9.6.1).
Version	M	The version of the DTM package. The value here shall be the same value as provided in the file name (see 9.6.1) and in the package manifest (see 9.6.7).

9.6.5 License file

The package relationship “License File” allows the package creator to indicate a license file (in ASCII Format), which provides license information to an end user.

If a user interface is available, then the Frame Application shall display this license as part of the installation process and ask the user to accept the license before installing the DTM.

9.6.6 Readme file

The package relationship “Readme File” allows the package creator to indicate a readme file, which provides additional information to an end user.

If a user interface is available, then the Frame Application shall offer to display this readme file as part of the installation process.

9.6.7 Icon file

An image file within the package, which may be displayed as the package icon (e.g. on FDT®hub). Image file size is limited to 1 MB. Supported file format is PNG. An image resolution of 128x128 is recommended.

9.6.8 DTM package manifest

A package manifest describes a DTM package file and shall be provided within a DTM package file. It provides information about the DTM before installation of the DTM. Using this information the Frame Applications can install the DTM automatically. Also a Frame Application can use this information to check whether a new version of a DTM package is available. The file name

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

shall be composed of the <Vendor>.<PackageName> (as defined for the package file name) and the suffix “package.manifest”.

A DTM package manifest xml file contains following information:

- DtmInfo: Information about the DTM which is included in the package
- ProductCode: Unique identifier (GUID) of the DTM package file
- PackageName: Name of the DTM package file (<PackageName>).
- PackageVersion: Version of the DTM package file (4 digit number).
- VendorName: Name of the company which provides the DTM package (<Vendor>).
- Description: Provides the description of the package, which may be read by a user prior to installing the DTM.
- SupportedRuntimes: Provides a list of runtimes supported by the DTM. The allowed values of the list is defined by [38].

NOTE: This is not a list of folders in the ‘lib’ part of the DTM package (see 9.6.2), but the list of runtimes in which the DTM can be installed and used by the Frame Application. Each supported runtime is mapped to one or more {runtime_id}-folder according to rules defined by the RID graph.

NOTE: The list provided by [38] may extend in the future. This means that the list of allowed values may change.

- Dependencies: Provides the information about dependencies similar to NuGet version 2.0. This field is used for describing the dependencies in regard to FDT library and protocol libraries only. It references the corresponding NuGet packages.

NOTE: Protocol libraries can be provided by FDT Group for standard protocols or by DTM vendors for vendor-specific protocols

See 7.6.1 for further information about package manifest files.

Figure 208 shows an example for a DTM package manifest file.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```xml
<?xml version="1.0" encoding="utf-8"?>
<DtmPackageManifest xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/Fdt.Deployment">
  <VendorName>VendorX</VendorName>
  <PackageName>DtmY</PackageName>
  <ProductCode>6b6719d5-12c0-488f-897c-af440e6c5a36</ProductCode>
  <PackageVersion xmlns:d2p1="http://schemas.datacontract.org/2004/07/System">
    <d2p1:_Build>0</d2p1:_Build><d2p1:_Major>2</d2p1:_Major>
    <d2p1:_Minor>3</d2p1:_Minor><d2p1:_Revision>-1</d2p1:_Revision>
  </PackageVersion>
  <DtmInfo xmlns:d2p1="http://schemas.datacontract.org/2004/07/Fdt.Dtm">
    <d2p1:DtmRef>
      <d2p1:AssemblyInfo>
        <Name>Fdt.VendorX.DtmY</Name>
        <Version xmlns:d5p1="http://schemas.datacontract.org/2004/07/System">
          <d5p1:_Build>0</d5p1:_Build><d5p1:_Major>2</d5p1:_Major>
          <d5p1:_Minor>3</d5p1:_Minor><d5p1:_Revision>0</d5p1:_Revision>
        </Version>
        <PublicKeyToken>1234567890123456</PublicKeyToken>
        <SupportedNetStandardVersions>
          <TargetNetStandard>
            <NetStandardVersion xmlns:d7p1="http://schemas.datacontract.org/2004/07/System">
              <d7p1:_Build>-1</d7p1:_Build><d7p1:_Major>2</d7p1:_Major>
              <d7p1:_Minor>0</d7p1:_Minor><d7p1:_Revision>-1</d7p1:_Revision>
            </NetStandardVersion>
          </TargetNetStandard>
        </SupportedNetStandardVersions>
        <Path>/VendorX/DtmY/</Path>
      </d2p1:AssemblyInfo>
      <d2p1:ClassName>Fdt.VendorX.DtmY.DtmMain</d2p1:ClassName>
    </d2p1:DtmRef>
    <d2p1:Name>HART Device DTM</d2p1:Name>
   <d2p1:Vendor> T&amp;T VendorX</d2p1:Vendor>
    <d2p1:Id>7db71a19-debc-4caa-883d-eb65ce1bda35</d2p1:Id>
    <d2p1:Version xmlns:d3p1="http://schemas.datacontract.org/2004/07/System">
      <d3p1:_Build>0</d3p1:_Build><d3p1:_Major>0</d3p1:_Major>
      <d3p1:_Minor>3</d3p1:_Minor><d3p1:_Revision>0</d3p1:_Revision>
    </d2p1:Version>
    <d2p1:FdtVersion xmlns:d3p1="http://schemas.datacontract.org/2004/07/System">
      <d3p1:_Build>-1</d3p1:_Build><d3p1:_Major>3</d3p1:_Major>
      <d3p1:_Minor>0</d3p1:_Minor><d3p1:_Revision>-1</d3p1:_Revision>
    </d2p1:FdtVersion>
    <d2p1:ProgIdsOfSupportedFdt1Dtms xmlns:d3p1="http://schemas.datacontract.org/2004/07/Fdt"
       i:nil="true" />
  </DtmInfo>
  <Description>Demo DTM Package</Description>
  <Dependencies>
    <PackageDependency>
      <Id>FDT_Group.FDT3</Id><Version>1.0.0</Version>
    </PackageDependency>
    <PackageDependency>
      <Id>FDT_Group.HART</Id><Version>1.0.0</Version>
    </PackageDependency>
  </Dependencies>
  <SupportedRuntimes xmlns:d2p1="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
    <d2p1:string>win</d2p1:string>
    <d2p1:string>linux</d2p1:string>
  </SupportedRuntimes>
  <MinimumNetStandardVersion xmlns:d2p1="http://schemas.datacontract.org/2004/07/System">
    <d2p1:_Build>-1</d2p1:_Build><d2p1:_Major>2</d2p1:_Major>
    <d2p1:_Minor>0</d2p1:_Minor><d2p1:_Revision>-1</d2p1:_Revision>
  </MinimumNetStandardVersion>
</DtmPackageManifest>
```


Figure 208 — Example: DtmPackageManifest

9.6.9 DTM device identification manifest

The device identification manifest file describes additional physical device parameters that are required for device identification. These files are dependent on the respective communication

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

protocol. Each file has to be placed into special subfolder that has the same name as used communication protocol identifier. Device identification manifest files are used only during installation. They are not required for installed DTMs because the DtmInfoBuilder deliver the same information.

The file name is composed of a unique name identifier and the fixed suffix ".deviceident.manifest".

A device type manifest xml file contains following information:

- DeviceIdentInfo: This information is used to describe physical device types which are supported by a DTM Device Type. It contains identification elements of a physical device type or device type group.

Figure 209 shows an example for a device identification manifest.

```
<?xml version="1.0" encoding="utf-8"?>
<DtmDeviceIdentManifest xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/Fdt.Deployment">
  <BusCategory xmlns:d2p1="http://schemas.datacontract.org/2004/07/Fdt">
    <d2p1:CommunicationType>Required</d2p1:CommunicationType>
    <d2p1:PhysicalLayers>
      <d2p1:PhysicalLayer>
        <d2p1:Id>bab2091a-c0a7-4614-b9de-fcc2709dcf5d</d2p1:Id>
        <d2p1:Name>HART FSK Physical Layer</d2p1:Name>
      </d2p1:PhysicalLayer>
    </d2p1:PhysicalLayers>
    <d2p1:ProtocolId>036d1498-387b-11d4-86e1-00e0987270b9</d2p1:ProtocolId>
    <d2p1:ProtocolName>HART</d2p1:ProtocolName>
  </BusCategory>
  <DeviceIdentInfos xmlns:d2p1="http://schemas.datacontract.org/2004/07/Fdt.Dtm">
    <d2p1:DeviceIdentInfo i:type="d2p1:DeviceIdentInfo_HartDeviceIdentInfo">
      <d2p1:DeviceSpecificProperties i:nil="true" />
      <d2p1:SupportLevel>SpecificSupport</d2p1:SupportLevel>
      <d2p1:ProtocolSpecificIdentInfo
xmlns:d4p1="http://schemas.datacontract.org/2004/07/Fdt.Dtm.Hart">
        <d4p1:BusProtocolVersion>
          <d2p1:ProtocolSpecificName i:nil="true" /> <d2p1:RegularExpressions i:nil="true"
/><d2p1:Value>5</d2p1:Value>
        </d4p1:BusProtocolVersion>
        <d4p1:DeviceCommandRevisionLevel>
          <d2p1:ProtocolSpecificName i:nil="true" /> <d2p1:RegularExpressions i:nil="true" />
<d2p1:Value>4</d2p1:Value>
        </d4p1:DeviceCommandRevisionLevel>
        <d4p1:DeviceFlags>
          <d2p1:ProtocolSpecificName i:nil="true" /> <d2p1:RegularExpressions i:nil="true" />
<d2p1:Value>3</d2p1:Value>
        </d4p1:DeviceFlags>
        <d4p1:DeviceProfile>
          <d2p1:ProtocolSpecificName i:nil="true" /> <d2p1:RegularExpressions i:nil="true" />
<d2p1:Value>0</d2p1:Value>
        </d4p1:DeviceProfile>
        <d4p1:DeviceTypeCode>
          <d2p1:ProtocolSpecificName i:nil="true" /><d2p1:RegularExpressions i:nil="true" />
<d2p1:Value>123</d2p1:Value>
        </d4p1:DeviceTypeCode>
…
        <d4p1:SoftwareRevision>
          <d2p1:ProtocolSpecificName i:nil="true" /> <d2p1:RegularExpressions i:nil="true" />
<d2p1:Value>3</d2p1:Value>
        </d4p1:SoftwareRevision>
      </d2p1:ProtocolSpecificIdentInfo>
    </d2p1:DeviceIdentInfo>
  </DeviceIdentInfos>
</DtmDeviceIdentManifest>
```

Figure 209 — Example: DeviceIdentManifest

9.6.10 DTM package file creation rules

This chapter describes the rules and recommendations that each DTM package file shall follow in order to achieve a reliable and standardized behavior of the installation process. This is

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

required in order to enable a Frame Application to retrieve information about a DTM package file, automatically install or remove it, perform updates, etc.

The (mandatory) rules are:

1. **Use OPC package as base for the installation**. The package will be installed by the Frame Application.
2. **Always sign the complete content of the package with a digital signature**. This allows for check of package consistency. If the digital signature of a contained file differs from the signature stored in the package, the file was broken during download.
3. **All DTM components (assemblies) shall have strong names**. Strong naming avoids version conflicts if components are shared between different DTMs or between different versions of the same DTM.
4. **All FDT components shall be installed using official FDT packages**. FDT binary files cannot be used directly in packages (e.g. as automatic dependencies). All central FDT settings (e.g. Registry entries) shall be provided by FDT packages only. FDT components are installed in the FDT-specific NuGet repository.
5. **A DTM package shall be uniquely identifiable**. That means support information (version number and build index) shall be available in the core properties in order to identify the version of the DTM package.
6. **The conformity records contained in the DTM package shall be signed by FDT Group**. The certification test is executed based on the DTM package, which has been signed by the originator. After successful test, the FDT Group generates the conformity record and adds the signed files to the package file [35].
7. The content of a certified DTM package shall be signed by FDT Group. After successful certification test, the FDT Group counter signs the content of the package.

9.6.11 Countersignatures

More than one signature can be applied to the content of a DTM package. For example, the certification process of FDT Group requires that the content of a DTM package is signed by the originator of the package and that a certified DTM also will be signed by FDT Group office.

Adding a new signature always creates a new relationship within the relationships part attached to the Digital Signature Origin part. In order to add new signatures to a DTM package without invalidating existing signatures, this relationships part must remain unsigned (although a subset of relationships may be signed). If the relationships part is included in a signature, that signature will be invalidated by all subsequently applied signatures.

9.6.12 NuGet packages

The Frame Application will install the referenced dependencies (NuGet packages) prior to installing the DTM.

The referenced NuGet packages will be installed in the default folder <NuGet Repository> (see 9.2.1).

This includes also the protocol assemblies.

NOTE The general expectation is, that the location of the referenced NuGet packages is the same location as of the referencing DTM package file. I.e. if the DTM package file is used for installation from a repository, the NuGet packages are also available from the repository or if the DTM is installed from an USB stick, the NuGet packages are also available from the USB stick.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

9.7 DTM deployment

A DTM package shall support the features listed below in order to enable Frame Applications to perform automatic DTM deployment.

- A DtmPackage manifest file containing basic DTM information (also used for DTM registration, 9.6.3) is provided together with the package
- The DTM package can be installed by the Frame Application without a user interface (silent installation)

Figure 210 outlines the automatic DTM deployment and installation flow of event in a client / server system.

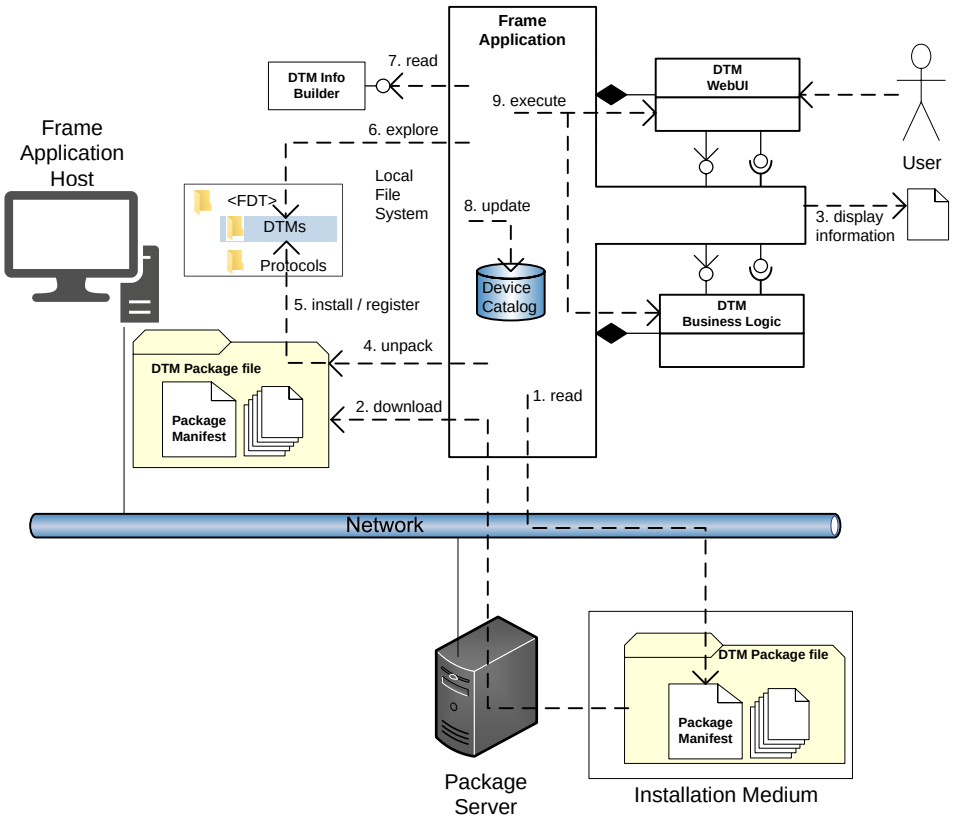


Figure 210 — DTM deployment

Flow of Events:

1. Frame Application connects to the package server (vendor-specific) and reads the DtmPackage manifest provided with the DTM package(s). The Frame Application can for example use this information to detect that the installation on the client should be updated (for rules of updates see 10.4).
   Before the DtmPackage is downloaded and installed, the Frame Application shall check the prerequisites of the DTM installation.
2. Frame Application downloads the DTM package file to the FA host (vendor-specific).
3. If the deployment is executed with user interaction (i.e. a user interface exists), then the Frame Application shall display the License file (indicated by the package relationship “License File”) and offer to display the Readme file (indicated by the package relationship “Readme File”).
4. Frame Application unpacks the DTM package into the < FDT_DTMs> directory
5. all necessary files are installed on FA host
6. – 7. Exploring for registered DTMs as described in 9.5.2.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

8. Update the device catalogue.
9. Execute the DTM BL and the DTM WebUI as needed.

When unpacking the DTM package (see step 3), the Frame Application shall collect the libraries required for the target runtime from the DTM package according to the RID graph into the “lib” folder of the installation path (see Figure 211). This means, that the Frame Application copies the files and sub-folders contained in the respective source folders to the “lib” folder of the installation path. Also the other folders of the DTM package are copied to the DTM installation path.

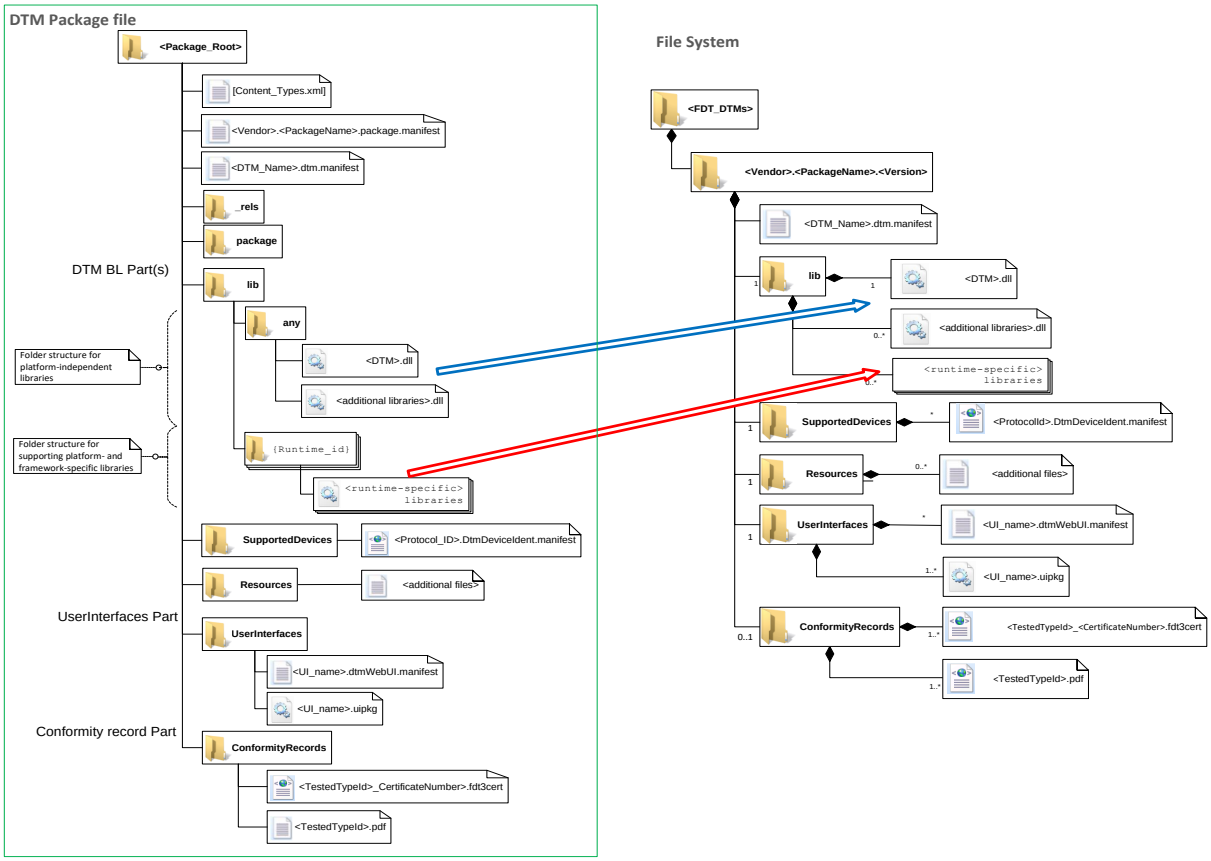


Figure 211 — Unpacking the DTM package

9.8 Paths and file information

9.8.1 Path information provided by a DTM via IFunction

There are possibilities where a DTM has to provide paths to files on the local file system:

- documentation (DocumentFunction)
- protocol-specific files (e.g. GSD file)

Usually, these files will be copied on the local file system during DTM installation. The DTM has to provide the full path (on Windows including the drive).

9.8.2 Paths and persistence

Installation on different PCs may lead to different path information.

Since a DTM instance dataset could be copied from one PC to another PC, a DTM shall not rely on path information, which is stored in its instance dataset.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

General rule: A DTM should never store path information in its instance dataset.

This rule is also valid for paths that are provided by the Frame Application and used by the DTM (e.g. for temporary storage).

9.8.3 Multi-user systems

Some Frame Applications provide multi-user capability, which means that the system is distributed over several PCs. In such a system the Frame Application should not retrieve path information from one PC and expect that the file is also available on all other PCs in the same path.

General rule: Path information is only valid on the local PC.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

10 Life cycle concept

10.1 General

FDT based systems and FDT software components are used in a plant to provide access to field devices. Device or configuration changes during the plant life cycle may affect FDT based systems and require changes of FDT components. FDT components life cycle is also considered, covering development, deliverables, installation, changes and use in an FDT system.

An FDT system shall maintain a consistent configuration of hardware and software components. Consistency requires that all installed and used components are compatible and interoperable to each other. Hardware comprises automation systems (PLCs, DCS), field devices and related infrastructure which are used to communicate with software executed on computers. Software components include FDT components and their runtime environment. All components have relations to other components. Life cycle management means ensuring a consistent configuration after changing components or their configuration.

Frame Applications provide a runtime environment for DTMs and functionality to manage DTM changes based on FDT defined identifiers.

10.2 Technical concept

10.2.1 General

This chapter describes the general technical concept of FDT3 life cycle support. The technical concept defines rules to identify software and hardware components as well as rules to ensure backward compatibility of a component from one version to another.

The FDT specification defines two types of attributes for Frame Applications, DTMs, physical devices, datasets and environment.

- Identification attributes

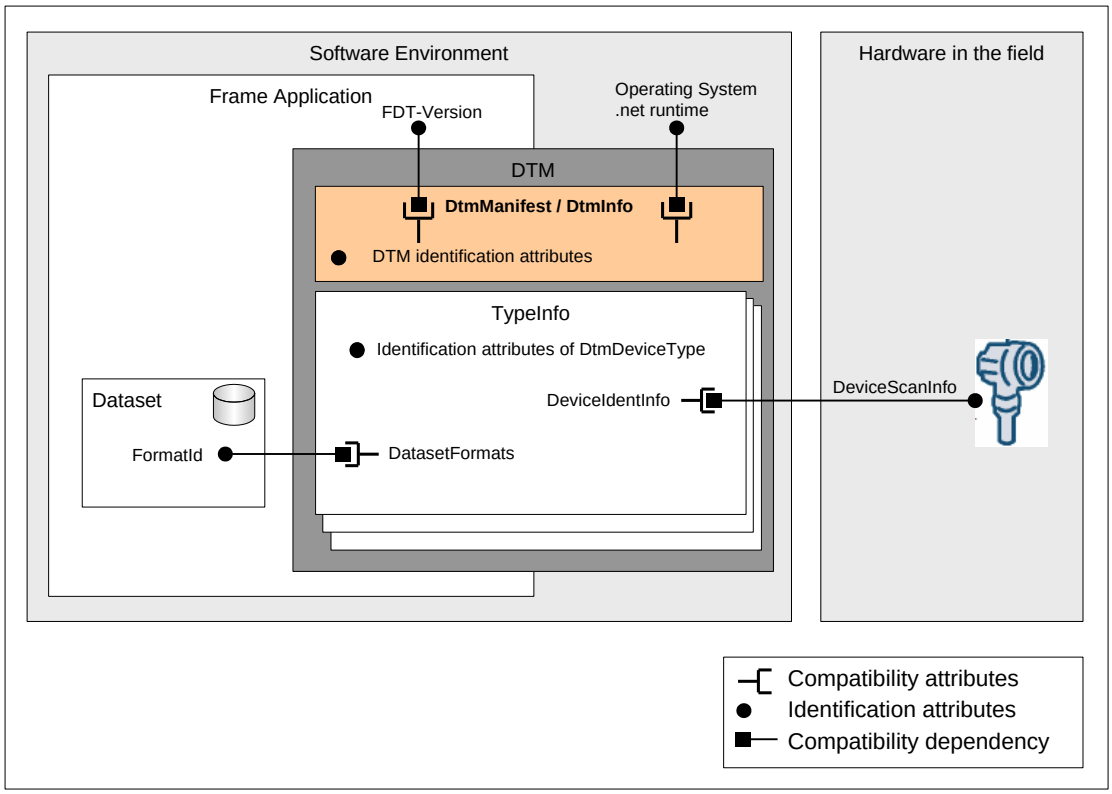
  describe the identity of FDT related components. They include version attributes distinguishing newer and older versions.

- Compatibility attributes

  allow the assessment of component compatibility before installation, after installation and at runtime.

Figure 212 shows life cycle relevant identification information of DTM and corresponding compatibility information of related objects.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Note: This example shows DtmDeviceType, but the same concept applies for DtmModuleType and DtmBlockType.

Figure 212 — Overview DTM identification

The following chapters define identification information for components which shall be considered to manage the configuration of an FDT system. Such component identifications are: device identification and DTM identification before and after installation, format of saved dataset, protocol, and environment.

10.2.2 DtmManifest / DtmInfo

The DTM information is provided in the DTMManifest. See Table 49 for attributes of the DTM manifest.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 49 — DTM identification

Identification attribute	Description
DtmInfo.Name DtmInfo.Vendor	Human readable Information about the DTM software, which can be used for communication with DTM vendor. Name and vendor may be different in different DTM versions.
DtmInfo.Id	Unique identifier used to identify DTM (e.g. in DtmPackage). This id is defined by the DTM vendor and shall be the same for all subsequent DTM Update / Upgrade versions. A Frame Application shall store this Id in the context of a device node and instantiate the DTM of this Id. E.g.: LoadFrom() shall be called with the assembly name which is referenced from this Id. The installation of a new DTM version with the same Id replaces the currently installed version. DTMs with a new Id shall be installed in parallel to other DTMs with other Ids. DTMs with higher Version and same Id shall • at least support all DtmTypes (TypeInfo.Id) of previous versions • not reduce set of functions • not reduce supported operating systems and .NET runtimes
DtmInfo.Version	Newer versions of DTMs shall increment this version: Update Revision: incremented minor version Upgrade Revision: incremented major version.
DtmInfo.FdtVersion	FDT version to which the DTM is compliant. Read by the Frame Application to ensure interoperability according to the compatibility rules defined in the specific FDT specification versions.
AssemblyInfo.SupportedNetStandardVersions	.NET Standard versions for which this assembly was released. The Frame Application uses this property to identify, in which runtime version the DTM shall be executed.
ProgIdsOfSupportedFdt1Dtms	[Optional] List of COM ProgIDs of FDT 1.2.x DTMs for which the data stored via IPersistStream can be loaded.

10.2.3 TypeInfo

TypeInfo describes a component (DeviceTypeInfo or ModuleTypeInfo or BlockTypeInfo) supported by a DTM. Table 50 shows the human readable information supported by the TypeInfo (see 7.4 for information on TypeInfo datatype).

Table 50 — TypeInfo – user readable description of supported types

Identification attribute	Description
ProductManufacturerName	Human readable manufacturer name of supported product (device/block/module) types.
ProductName	Human readable name of supported product. The product name is defined by the product manufacturer and shall describe the supported device type. In case the type supports a range of products (e.g.: product family), the product name should allow an end user to recognize the supported range of products. The same range of products shall be identified by the list of DeviceIdentInfos of this DtmType.
ProductRevision	Human readable version string of the supported products. The returned string shall cover all software versions (e.g. firmware version) and, if needed also hardware versions, of supported DeviceIdentInfos. ProductRevision may include asterisks to indicate ranges of revision numbers. Example: “V 1.3.* - 2.*”

The following properties are used to identify a component type (e.g. DeviceTypeInfo) included in a DTM (Table 51).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table 51 — TypeInfo identification

Identification attribute	Description
Id	Unique identifier for the TypeInfo. The Id shall be unique within a DTM.

The DTM vendor defines the Id and shall ensure, that all future DTMs are backward compatible to previous versions. Compatibility includes dataset format, device support (DeviceIdentInfo) and range of functionality.

A TypeInfo with a specific Id can be defined only once in a DTM. The same Id may be used in different DTMs. This indicates to the Frame Application that those different DTMs support the same product.

10.2.4 Supported DataSet formats

The FormatId of a dataset defines the format of the saved data. Table 52 lists dataset related compatibility attributes of a dataset.

Table 52 — DtmType – Dataset support identification

Compatibility attribute	Explanation
DatasetFormats.Used	Used FormatId defines the format in which the data are saved by the DTM.
DatasetFormats.ReadSupported	Supported FormatId lists additional formats, which can be loaded and migrated by the DTM. The Frame Application can check with this property if a DtmType supports an existing dataset.

If the attribute IDataset.FormatId of a persisted dataset has the value 'Used', then the DTMDeviceType can load the dataset and save the same format.

If the attribute IDataset.FormatId of a persisted dataset has the value 'ReadSupported', then the DTMDeviceType may migrate the dataset and may save the data in its currently used format (indicated by DatasetFormats.Used).

The rules for DTM Update Revisions and DTM Upgrade Revisions are: If the format of the dataset changes, the DTM vendor shall define a new FormatId as 'Used'. The list of ReadSupported FormatId shall include all Formats used in previous revisions of the DTM.

A DTM vendor defines unique Ids. Typically different dataset FormatIds are defined for each DtmType. A DTM Vendor may define the same FormatId in multiple DeviceTypes. In this case the dataset migration shall be assured between these types.

The list of ReadSupported FormatId provided by an FDT3 DTM may include FormatIds used by DTMs based on previous FDT versions (e.g. FDT2.0 and FDT2.1). This allows for migration of existing projects (e.g. with DTMs based on FDT2.0) to projects with FDT3 DTMs.

10.2.5 DeviceIdentInfo

A DTM exposes in DeviceIdentInfo compatibility properties of physical types, which are supported by a DtmType. These properties are used by a Frame Application to automatically detect which DtmTypes are compatible with scanned physical types.

The Frame Application compares the following compatibility attributes programmatically with DeviceScanInfo identification to check if the DtmType supports the connected device.

- ProtocolId
- PhysicalLayer
- ManufacturerId
- DeviceTypeId

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

- HardwareRevision
- SoftwareRevision
- ProtocolIdentificationProfile
- DeviceSpecificProperties
- ProtocolSpecificProperties

Note: Not all properties of DeviceScanInfo can be used for identifying supported device types.

10.2.6 Dataset

Table 53 shows the identification property of a persisted DTM dataset.

Table 53 — Dataset identification

Identification property	Description
IDataset.FormatId	A Frame Application can identify the format of a saved dataset by the FormatId property.

When saving a dataset, the DTM shall write its DatasetFormats.Used to IDataset.FormatId.

10.2.7 DeviceScanInfo

A Frame Application is responsible to manage the assignment of a correct DtmType (e.g. DtmDeviceType) to a Device Node, which is logically connected to a device. This assignment uses identification information of the connected device, provided by the Communication Channel service ScanRequest. ScanRequest returns the datatype DeviceScanInfo for each scanned device (see Table 54).

Table 54 — DeviceScanInfo – scanned device identification

Identification property	Description
ProtocolId	Used by Frame Application to compare with DeviceIdentInfo.BusCategory.ProtocolId of the DtmTypes.
PhysicalLayer	Reflects the physical media that the physical device is connected to. Used by Frame Application to compare with DeviceIdentInfo.PhysicalLayers of the DtmTypes.
ManufacturerId DeviceTypeId HardwareRevision SoftwareRevision ProtocolSpecificProperties DeviceSpecificProperties ProtocolIdentificationProfile	Used by Frame Applications to compare with corresponding DeviceIdentInfo properties of DtmTypes.
SerialNumber Address Tag	Used to identify the unique device equipment, the fieldbus communication address and the TAG set in the device. The device serial number can be used to detect a device replacement with a device of same type.
Note: CommunicationErrorInformation - cannot be used for identification, but helps to detect errors in communication.	

10.3 DTM installation

10.3.1 General

The Frame Application is responsible for installing a DTM from a DTM package file.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

The FA shall check if the target operating system version is one of the tested versions (as indicated in the DTM package manifest). Otherwise the FA shall inform the user and request a confirmation to proceed.

The FA shall check if the minimum required .NET Standard (as indicated in the DTM package manifest) is installed on the target operating system. The FA may provide the possibility to install a compatible .NET Standard library. Otherwise if the FA does not have the mechanism to install a compatible .NET Standard libraries, the FA shall not proceed to install the DTM.

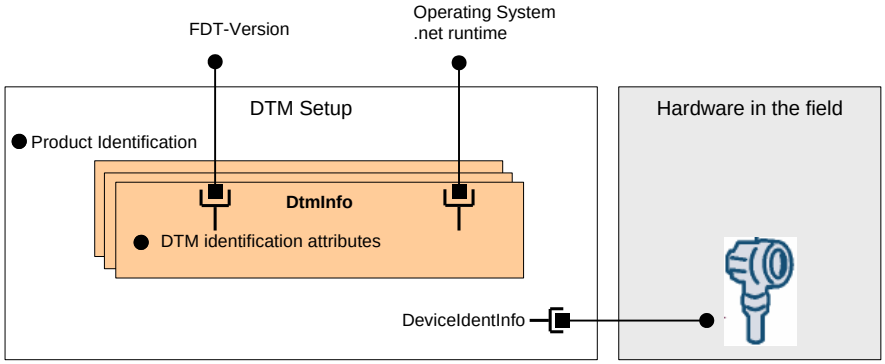


Figure 213 — Identification attributes in DTM package

A number of properties in the DtmPackageManifest identify a DTM package (see Figure 213 and Table 55) and allow the identification of included DTMs. (See also 9.5.3)

Table 55 — Package information

Identification attribute	Description
PackageName / VendorName	Human readable Information about the DTM software package, which can be used for communication with DTM vendor.
ProductCode / PackageVersion	Unique identifier and version used to identify the DTM package. The ProductCode shall be defined according installation rules (see clause 9).
DtmInfo	All information from DtmInfo – see Table 49. This information can be used to reference DTM package and installed DTMs.
Compatibility attribute	**Description**
DeviceIdentInfo	see 10.2.5
Note: Other properties of DtmPackageManifest, like MinimumNetStandardVersion and Description, are not used to support identification.	

The Frame Application shall compare SupportedNetStandardVersions and SupportedTargetPlatform properties of DtmPackageManifest with the version information of the installed .NET implementation runtime.

10.3.2 Handling of DTM installations

Based on the rules for DTM installation (see 9.5.1), during installation of DTM packages different folders are created for different versions of a DTM package. This means that different DTM versions may coexist side-by-side.

The general expectation is that the Frame Application will use the most recent version of the DTM installed on the system. Depending on the requirements of the Frame Application, the DTM instances used in existing projects may be replaced by the new DTM version (e.g. automatically or by user-choice) or preserve the used DTM version (e.g. in order to ensure stable execution of the projects). Also, a Frame Application should provide the possibility for a user to select a specific version of an installed DTM for use.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

10.4 Life Cycle Scenarios

10.4.1 Overview

Life cycle scenarios cover hardware or software changes. This chapter describes the impact of a change to the FDT system, and explains different cases and required actions to ensure a consistent system after the change (see Table 56).

Table 56 — Changing DTM – overview

DTM change reason (case)	Conditions	Resulting Actions
A) Add device to the field	DTM is not installed	i) Contact device vendor to request and order DTM. Continue with action ii).
		ii) Search in DTM packages, install DTM if matching DtmDeviceType is found. Continue with case B).
	DTM is installed	iii) Offline: User selects matching DtmDeviceType and adds it to the topology. See compatibility attributes in Figure 212.
		iv) Online: Scan. See 10.2.5 and 10.2.7.
B) Exchange physical device. (DeviceIdentInfo – DeviceScanInfo relation)	Same or compatible device	v) Same DtmDeviceType, matching version can be used. Download existing offline dataset or upload device dataset.
	Same device type, but incompatible device due to different versions.	vi) Install DTM Update/Upgrade Revision. vii) Updated DtmType has extended support range and can support device without reassignment. viii) Download existing offline dataset or upload device dataset.
		ix) Reassign (see 8.17) to a matching DtmDeviceType based on comparison of DeviceScanInfo and DeviceIdentInfo. x) Find DtmDeviceType with matching version and reassign DtmDeviceType of device node. Consider offline dataset migration.
	Different device type	xi) Search in installed DTMs for DtmDeviceType supporting DeviceIdentInfo corresponding to DeviceScanInfo. xii) Reassign DtmDeviceType of device node. Consider offline dataset migration.
		xiii) Search for DTM package including required DeviceIdentInfo corresponding to DeviceScanInfo, install and continue in action xi).
C) DTM bug fix or missing feature	No DTM package available	xiv) Contact device vendor to request, order DTM and make package available for step xi).
	DTM Update / Upgrade package available, but not yet installed	xv) Search in available DTM packages for DTMs with same DTM Identifier (DtmInfo.Id) in DTM Manifest and newer DTM version.
		xvi) Install the found new version of the DTM side by side with the old version of the DTM.
		xvii) Depending on the user requirements, the Frame Application may replace the old DTM in the project with the new version of DTM. The Frame Application passes the old dataset to the new version of DTM. New DTM versions are responsible to support all dataset formats of older versions (see Figure 212).

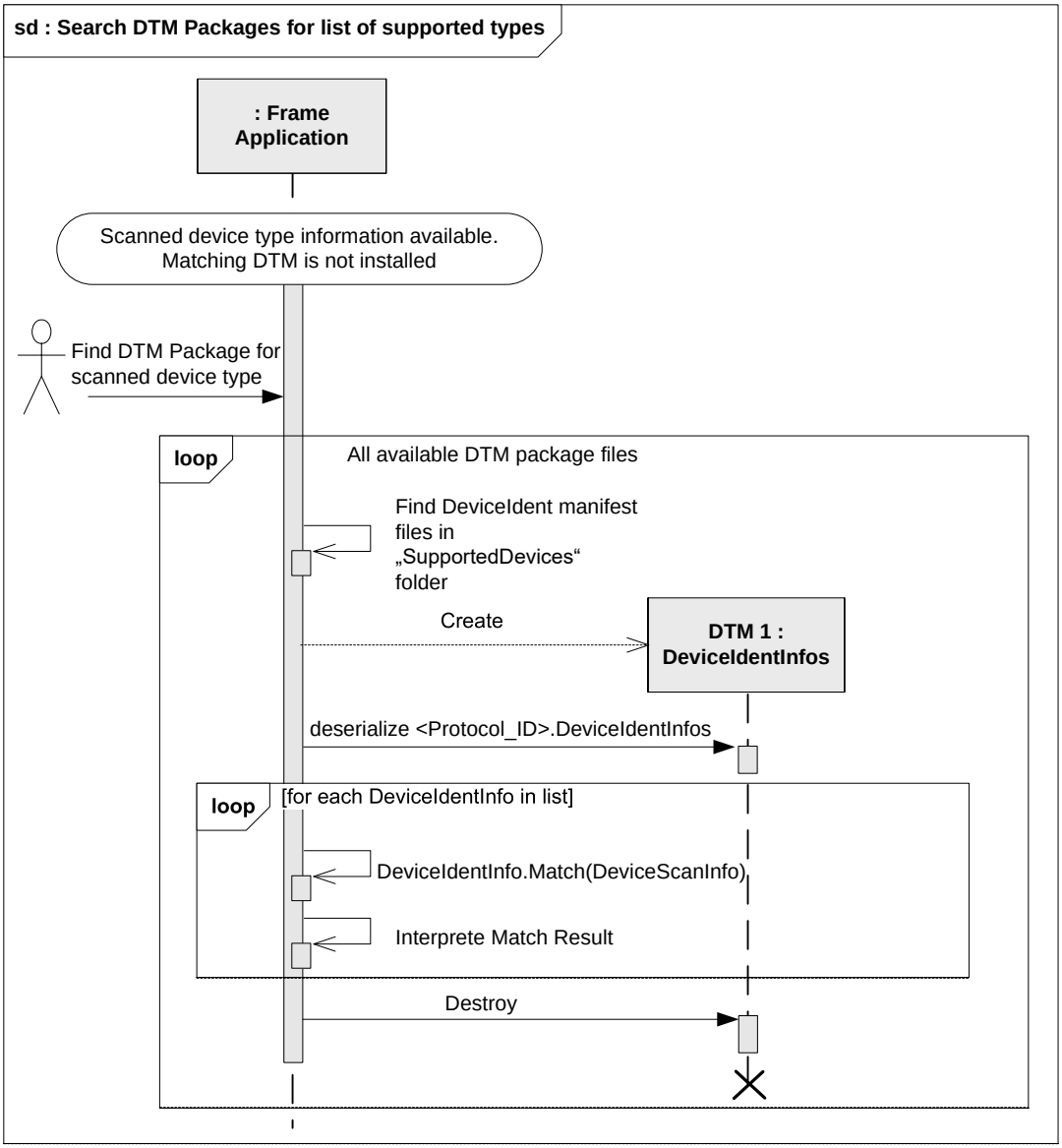
Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

DTM change reason (case)	Conditions	Resulting Actions
D) Assign different DTM to device node. E.g.: replace generic DTM by custom DTM or DTM vendor released a new DTM product.	Different DTM package has been made available (different DtmInfo.Id in DTM Manifest) supporting the same DtmType DeviceIdentInfo.	xviii) Search in DTM packages for DeviceIdentInfo, which supports the physical device identification, e.g. DeviceScanInfo.
		xix) Install found DTM, which does not replace the original DTM with the different DTM Identifier.
		xx) The Frame Application shall offer a reassignment to the new DTM and inform the user about the possibility (see Figure 212) to migrate the dataset of the device node. Note: This possibility is likely not available if the new DTM is manufactured by a different DTM vendor.

10.4.2 Search for device type in DTM packages

If (for instance during scan) a Frame Application recognizes a device type for which no DTM has been installed, it is possible for the Frame Application to use the device identification information to find a package for a DTM which supports the specific device type (see Figure 214).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

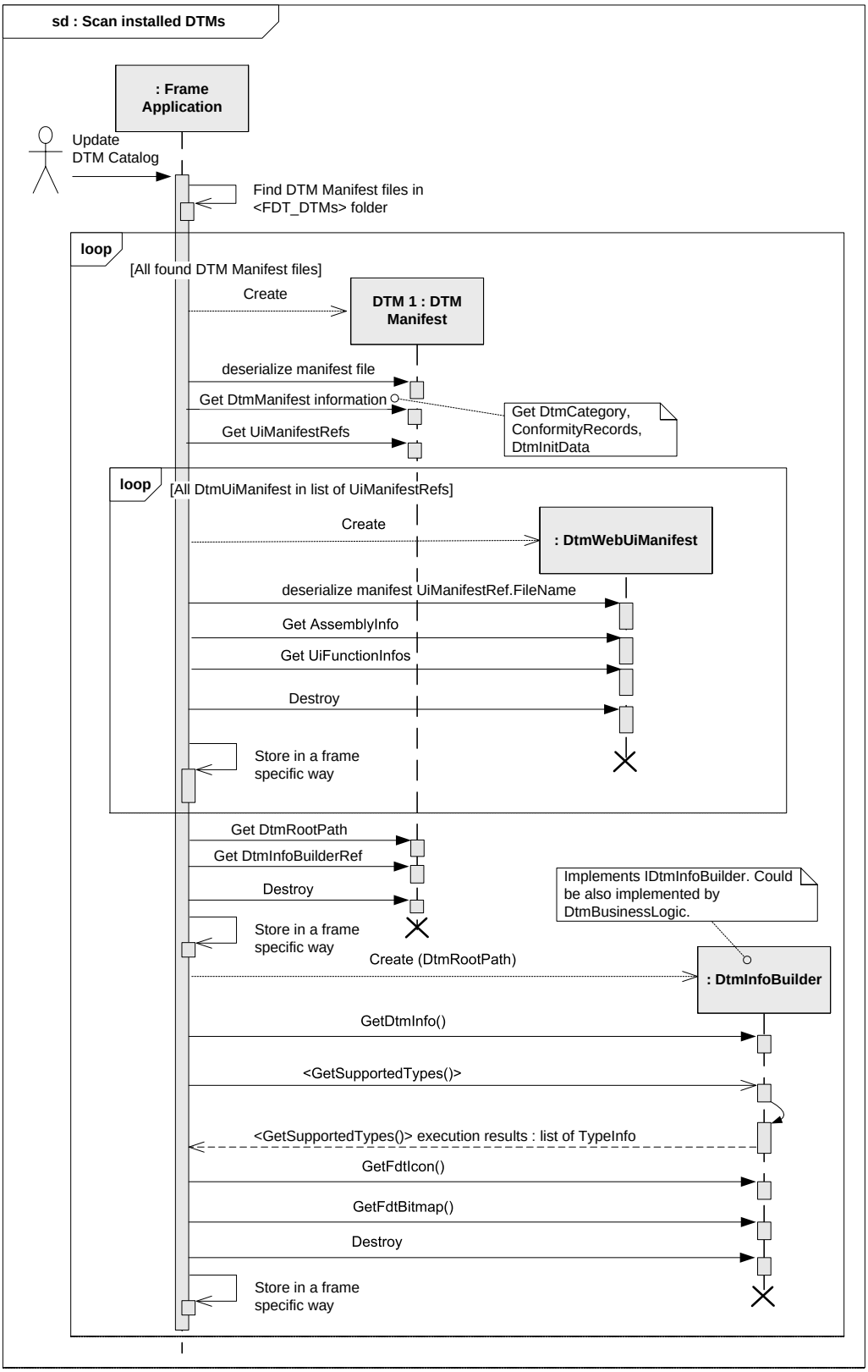
DeviceIdentInfo
DeviceIdentInfo.Match()

Figure 214 — Search DTM Package for list of supported types

10.4.3 Search for installed DTMs

Typically Frame Applications maintain a list of installed DTMs and their supported types. The workflow in Figure 215 describes, how this DTM information can be retrieved from installed DTMs.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.



Used methods:

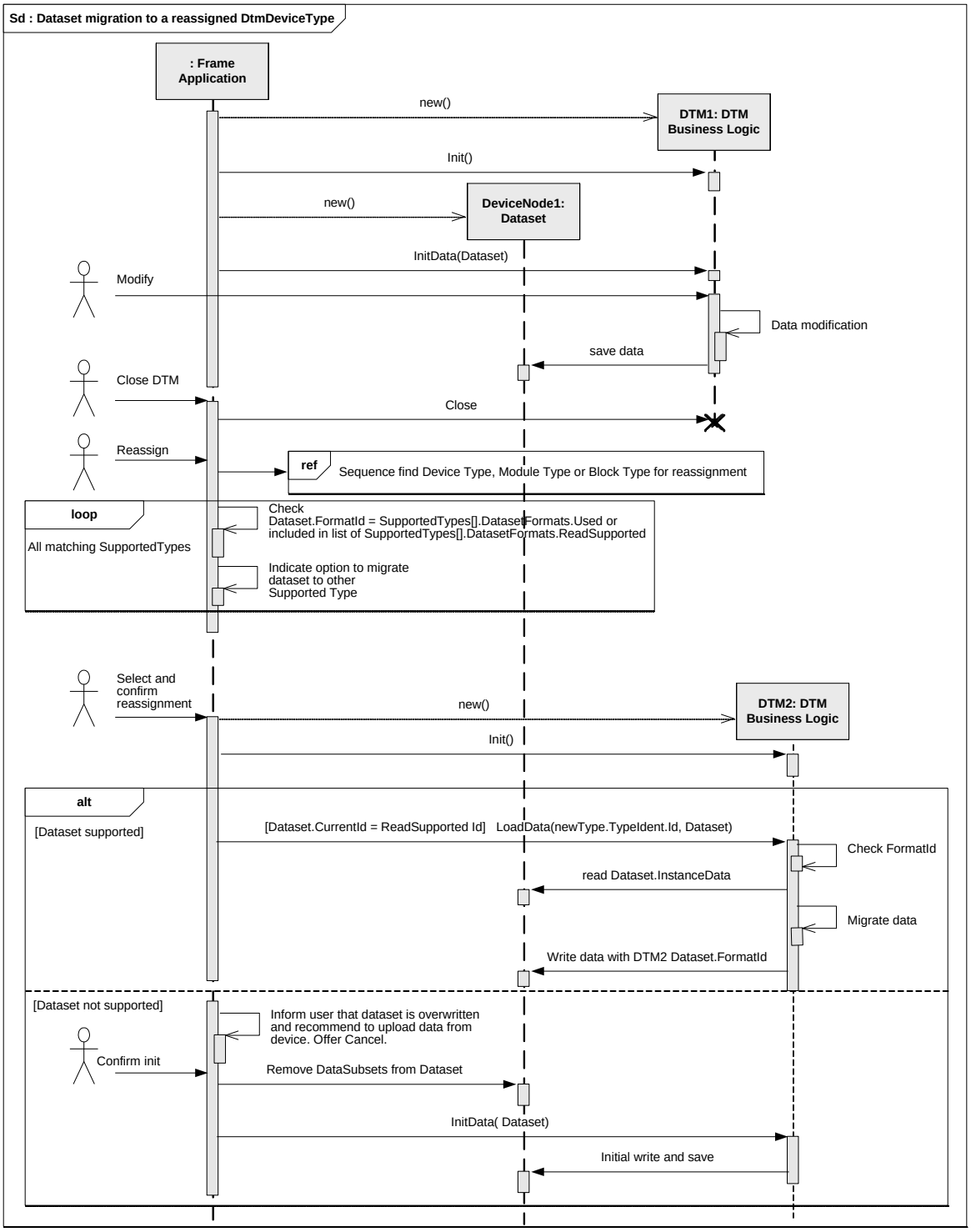
GetDtmInfo()
BeginGetSupportedTypes() / EndGetSupportedTypes()
GetFdtIcon()
GetFdtBitmap()

Figure 215 — Scan installed DTMs

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

10.4.4 Dataset migration for reassigned DTM

Figure 216 shows a sequence related to dataset migration for a reassigned DtmDeviceType.



Used methods:

IDtm3.Init()
IDtm3.InitData()
IDtm3.LoadData()
TypeInfo.DatasetFormats
IDataset.FormatId

Figure 216 — Dataset migration to a reassigned DtmDeviceType

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

11 Frame Application architectures

11.1 General

The target of FDT3 is to support different architectures for Frame Applications. Examples are:

- Standalone application
- Remoted User Interface
- Distributed multi-user application
- OPC UA server
- Web services

The implementation of these different architectures is always specific for a Frame Application product. DTMs should be able to support all possible application scenarios.

11.2 Standalone application

For configuration tools supporting only one user at a time, the most common architecture will be the standalone application, where GUI and business logic are executed in the same process.

Note: Since FDT3 supports multiple CLR versions, it will be necessary to consider execution of DTMs in different processes even for standalone systems.

11.3 Remoted user Interface

The Frame Application can execute the DTM Business Logic and DTM WebUI on different PCs as shown in Figure 217. In such a scenario the Frame Application starts the DTM WebUI on a Client PC (this applies to all types of DTM User Interface).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

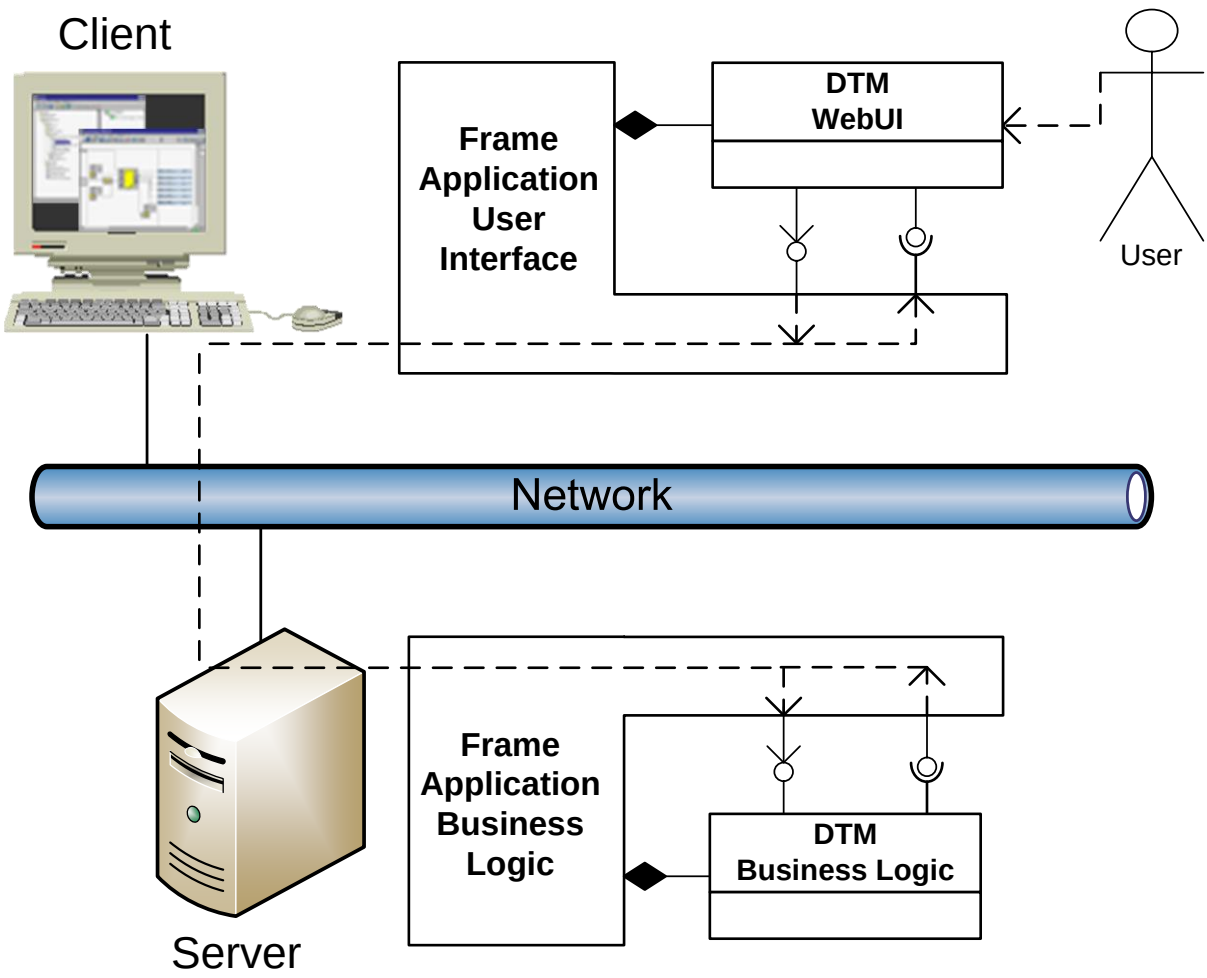


Figure 217 — Client / Server Application

The interaction management between DTM Business Logic and DTM WebUI is done by Frame-Application-specific means. The DTM Business Logic and DTM WebUI are not aware of the execution on different PCs.

Note: If a Frame Application with remoted UI supports multi-user scenarios, it should never start multiple DTM WebUIs for different users accessing the same DTM BL instance. Instead the FA should start a different DTM BL instance for each user.

11.4 Distributed multi-user application

A Frame Application may be implemented as a distributed environment, consisting of multiple workstations and servers. Such an environment will support multiple users (in different roles) working at the same time (see Figure 218).

If multiple users access the same field device, Frame Applications shall start several DTM instances of the same DTM type and for the same physical device. The different DTM instances access the same persistent DTMDataSet.

Note: A Frame Application should never start multiple DTM WebUIs for different users accessing the same DTM BL instance.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

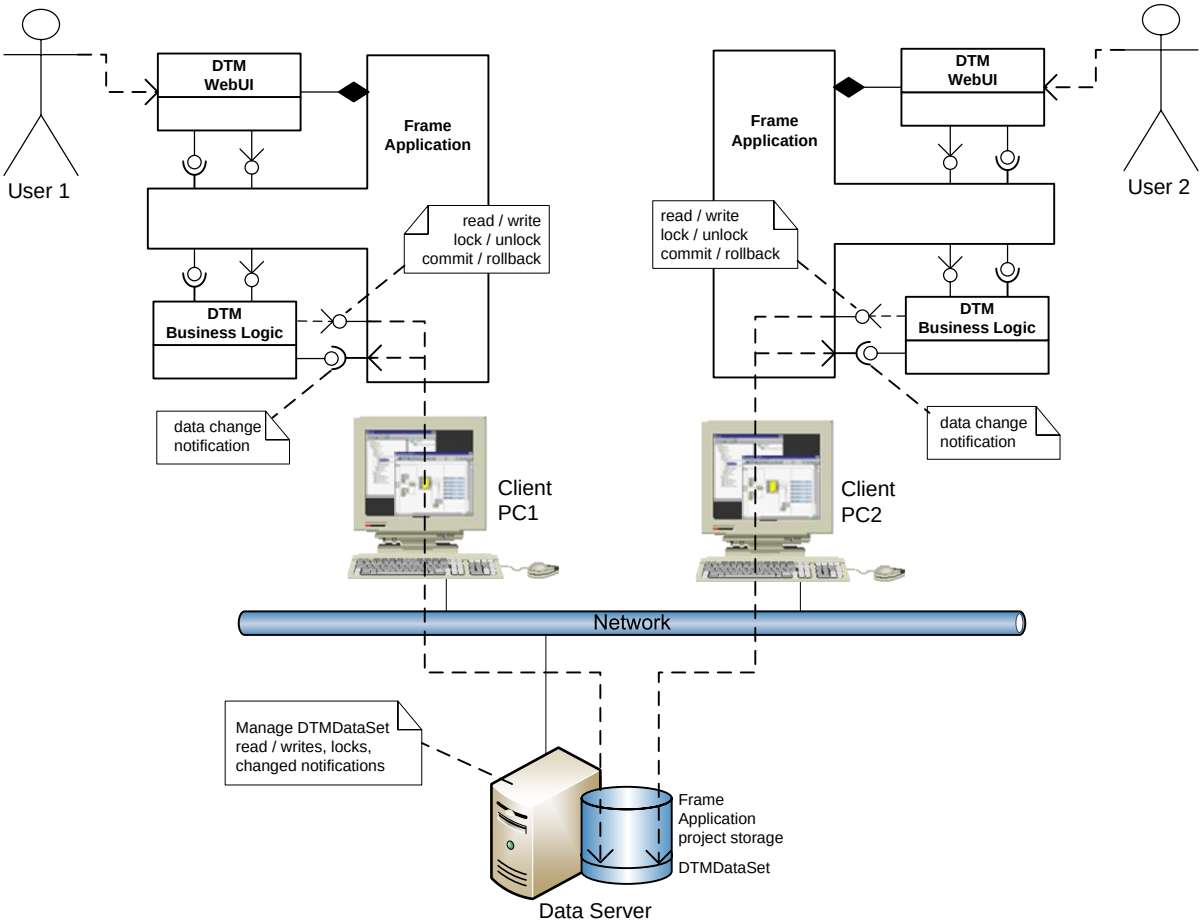


Figure 218 — Example for distributed multi-user application

11.5 OPC UA

A Frame Application may provide OPC UA support according to the definition in the FDT OPC UA mapping document [13][14].

Figure 219 outlines an example for an OPC UA Server based on FDT3.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

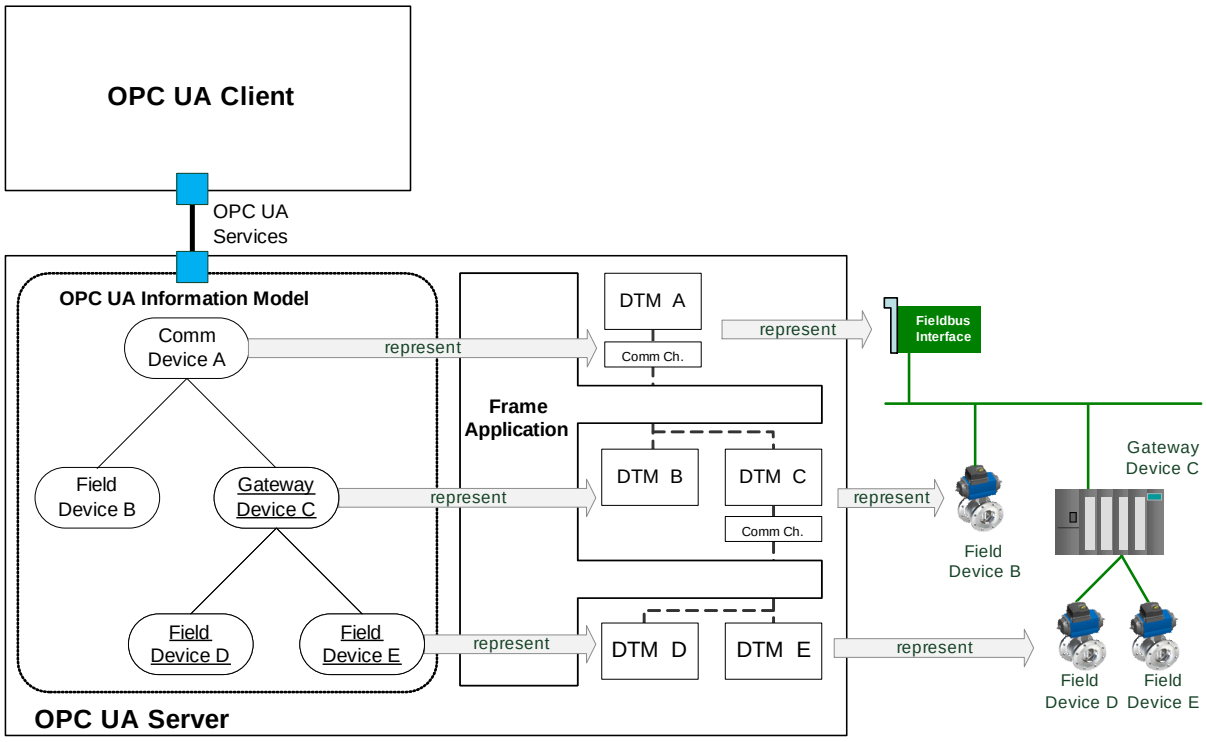


Figure 219 — OPC UA server based on FDT3

The OPC UA Server contains a Frame Application, which integrates the DTMs representing the physical devices. The information provided by these DTMs is used to create an OPC UA Information Model. Especially the DTM Device Type (see 7.4), Device Data (see 7.9), and Process Data Info (see 7.11.1) are used for this purpose.

The Information Model represents the DTM information and functions within the OPC UA Server. The information and functions are modeled as hierarchically organized nodes. The format and content is defined by an FDT / OPC UA Information Model specification.

The OPC UA Server exposes the Information Model nodes by corresponding OPC UA services. Client applications use these services to read and write data in the Information Model. Depending on the DTM information which was used to create the accessed Information Model node the actions triggered in the Frame Application may differ.

11.6 Web services

A Frame Application may provide web services support according to the definition in the FDT WebServices document [31].

Figure 220 outlines an example for a web server based on FDT3.0.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

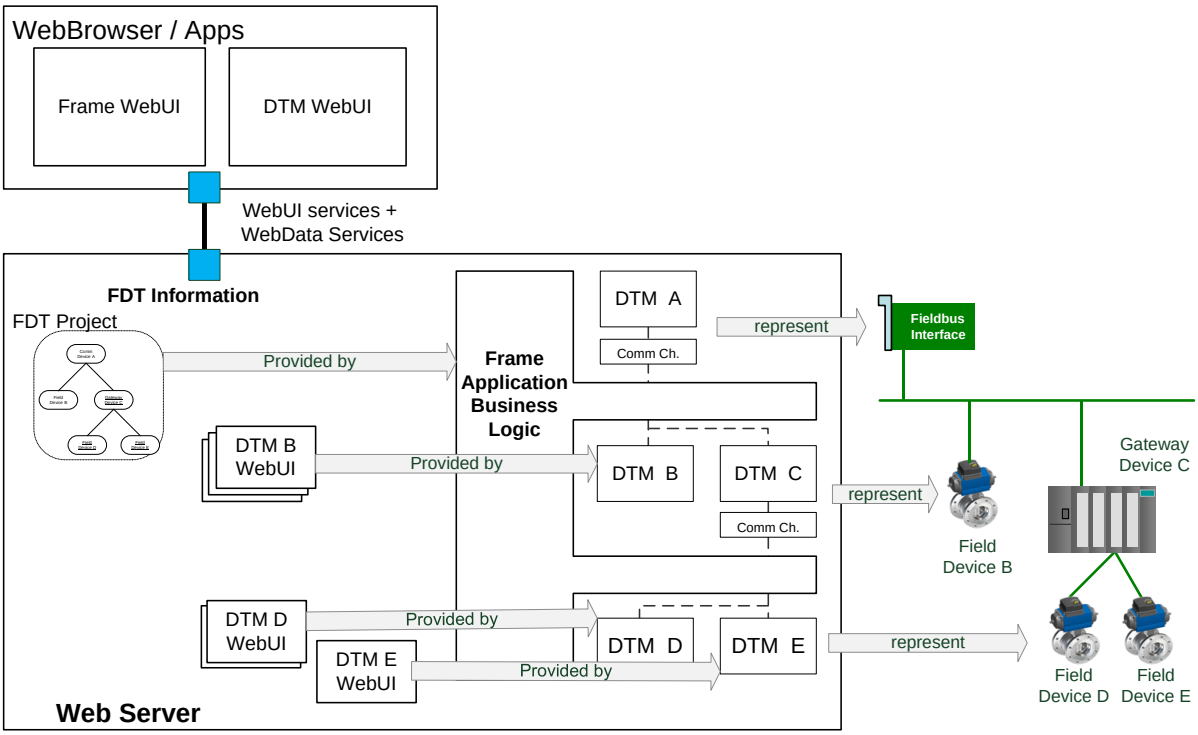


Figure 220 — Web server based on FDT3.0

The Web Server contains a Frame Application, which integrates the DTMs representing the physical devices. The information provided by these DTMs is transported via the WebUI services and WebData services. WebUI services are used to transport the DTM WebUI and WebData services are used to transport the DTM data, which is displayed in the DTM WebUI.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Annex A
(normative)
FDT3 Use case model

A.1 Use case model overview

This section describes the use cases that were considered when designing the FDT Specification. A Frame Application may support only a subset of these use cases. Also a Frame Application may support additional use cases that are not defined in this section.

Figure A.1 shows the main use cases considered in FDT3 and their relation to the user levels defined in FDT3.

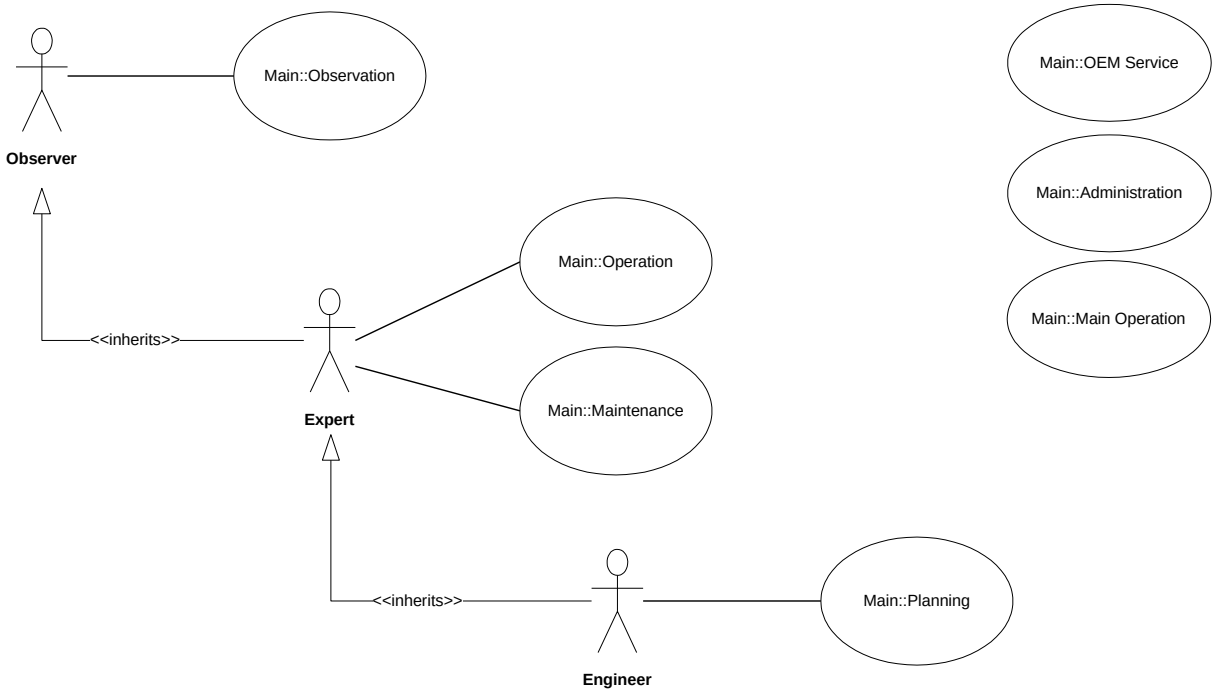


Figure A.1 — Main use case diagram

They are specified in more detail in the following sections, including textual descriptions and optionally some necessary scenarios.

A.2 Actors

The actor types in the above displayed use case diagram are organized in a hierarchical way.

The “Expert” actor in general inherits the permissions of the “Observer” actor. The “Engineer” actor inherits the use cases of the “Expert” actor. Use cases, which are passed on to an actor of higher level, may act in an extended way to match their intention. The following table (Table A.1) provides a brief description of the actor roles:

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table A.1 — Actors

Actor Name:	Observer
Brief Description:	This actor stands for a person that may observe the current process.
Inherits:	None
User Level:	UserInfo.UserLevel = Observer
Access Verification:	The access as "Observer" actor may have no password.
Use Cases:	Online View, Report Generation, Asset Management
Actor Name:	**Expert**
Brief Description:	This actor stands for a person who has to observe and manage the current operation (i.e. an Operation Expert) or for a person executing maintenance (i.e. a Maintenance Expert). Since the “Expert” actor may have many different permission sets in different applications, the permissions shown in this Annex A present the super set of permissions. Each Frame Application may adapt the permission set for the “Expert” according to more specific user levels.
Inherits:	Observer
User Level:	UserInfo.UserLevel = Expert
Access Verification:	The access as "Expert" typically requires a password.
Use Cases:	User Login, Audit Trail, Archive, Operation, Maintenance
Actor Name:	**Engineer**
Brief Description:	An "Engineer" shall have the possibility to perform all necessary maintenance operations including device exchange, teaching, calibration, adjustment, etc. The person may therefore download verified parameter sets, modify a subset of parameters online or offline, perform device-specific online operations and at the end of the processing, may have the possibility to upload the complete parameter set. A Frame Application may control the ability of an “Engineer” to perform actions like a plant engineer, a specialist (s. VDI/VDE2187) or any fully authorized person. That is why an “Engineer” can use DTM functions without any restrictions. Only OEM-specific operations may require additional DTM-specific login.
Inherits:	Expert
User Level:	UserInfo.UserLevel = Engineer
Access Verification:	The access as "Engineer" actor typically requires a password.
Use Cases:	Simulation, Online Operation, Repair, Offline Operation, Bulk Operation User Login*, Online View*, Audit Trail*, Archive*, Report Generation*, Asset Management* DTM Instance Handling, Simulation*, Online Operation*, Repair*, Offline Operation*, Bulk Operation *, User Login*, Online View*, Audit Trail*, Archive*, Report Generation*, Asset Management* (* Inherited use cases)

A.3 Use cases

Use case overview

The following section describes all use cases of the use case diagram in tabular form. To reduce information, all attributes of included use cases, which are identical with the related main use case, are not displayed.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Observation use cases

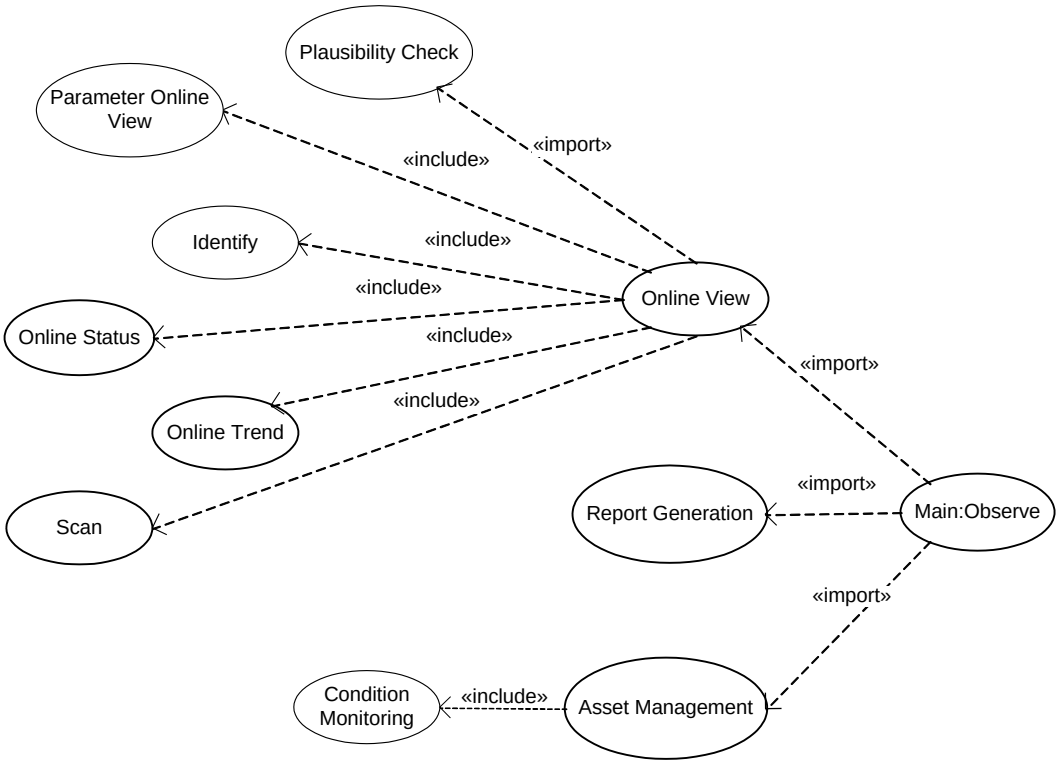


Figure A.2 — Observation use cases

Only the observation use cases can be executed by the actor “Observer” (see Figure A.2 and Table A.2). The “Observer” stands for a person that has the lowest access level.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table A.2 — Observation use cases

Use Case:		Online View
Brief Description:		This use case offers a complete set of operations for viewing of device parameters and status.
GUI:		The GUI is part of the DTM. The Frame Application can offer online views for a group of devices.
Device Connection:		Needs an established connection to one or more devices
UserLevel	Observer:	Accessible
	Expert:	
	Engineer:	
UserFlags:		No changes
Included Use Case:		Parameter Set Online View
Brief Description:		Enables the actor to load parameters from the device for viewing and documenting (printing). (DTM applicationId: Observe)
Included Use Case:		Online Status
Brief Description:		Use case for viewing and analyzing the current device status. (DTM applicationId: Diagnosis)
Included Use Case:		Online Trend
Brief Description:		Use case for generating and watching trends of dynamic online values.
Included Use Case:		Plausibility Check
Brief Description:		Every time the device parameters or the configuration data is changed, a plausibility check is automatically performed. As long as the plausibility check fails, the data cannot be written to the device.
Included Use Case:		Identify
Brief Description:		Displays identification of device instance information e.g. address, TAG, Fieldbus-Rev., DD-Rev., Firmware. This information is protocol dependent. It also may contain device-type-specific information. Further information may be provided: references to documentation, area to add comments to the device instance. Refer to Device Identification chapter in protocol-specific Annex specifications. (DTM applicationId: Identify)
Included Use Case:		Scan
Brief Description:		Retrieves the list of available devices from a channel object. The channel object may be provided by a DTM or by Frame Application.
Device Connection:		Needs at least a temporary connection. (The channel shall have online access.)
Use Case:		**Report Generation**
Brief Description:		The print function of a use case can normally be started from its GUI. In addition to printing the actual view of a GUI some Frame Applications may implement a configurable documentation mechanism which enables the actor to generate reports. This report may be generated by combining the prints of several use cases or several devices. For example a report could be generated containing "Online View", "Audit Trail" and "Online Operation" printouts for one device or a report may be generated containing the configuration of a HART multiplexer and its clients.
GUI:		Frame Application
Device Connection:		Depends on the type of report
UserLevel, UserFlag:		The printed information may depend on user level and user flags.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Use Case:		Asset Management
Brief Description:		Frame Application provides mechanism for asset management
GUI:		The component, which supports asset management functions, has to provide an adequate GUI. (DTM applicationId: AssetManagement)
Device Connection:		Not necessary
UserLevel	Observer:	Accessible for viewing only
	Expert:	
	Engineer:	
Included Use Case:		Condition Monitoring
Brief Description:		Enables the actor to monitor the operation condition of the device for viewing and documenting (printing).

Operation use cases

The Operation use cases are available for the actor “Expert” (see Figure A.3).

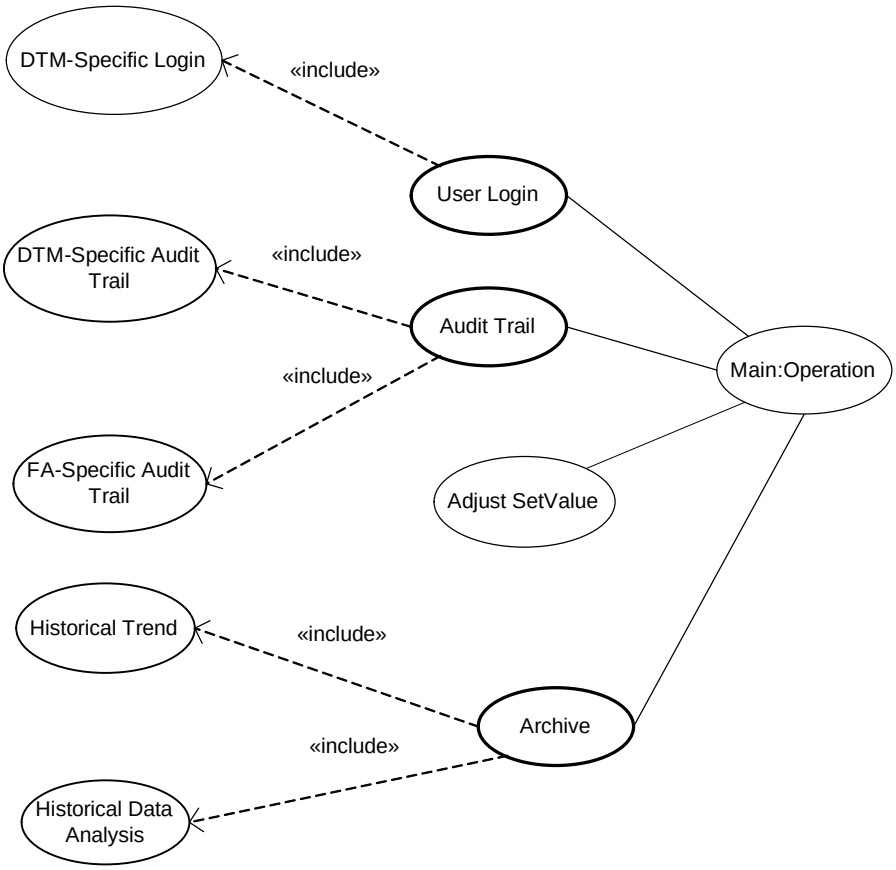


Figure A.3 — Operation use cases

The following table (Table A.3) provides a description of Operation use cases.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table A.3 — Operation use cases

Use Case:		User Login
Brief Description:		A person of specified actor type logs into the Frame Application. If verification fails the person may act as an "Observer" actor only.
GUI:		Part of the Frame Application
Device Connection:		Not established
UserLevel	Observer:	Depends on the Frame Application
	Expert:	Accessible
	Engineer:	
Included Use Case:		DTM-Specific Login
Brief Description:		Access to manufacturer-specific information e.g. production codes, changes log
GUI:		Part of the DTM
Device Connection:		Typically needs an established connection
Use Case:		**Audit Trail**
Brief Description:		Use case to enable the actor viewing and deleting audit trail data.
GUI:		The component, which supports audit- trail, has to provide an adequate GUI.
Device Connection:		No connection necessary
UserLevel	Observer:	No access
	Expert:	Accessible for viewing only
	Engineer:	View, export and delete. Frame Application may restrict access.
Included Use Case:		DTM-Specific Audit Trail
Brief Description:		Every DTM might support an audit trail on its own. (DTM applicationId: AuditTrail)
Included Use Case:		FA-Specific Audit Trail
Brief Description:		The Frame Application may implement a system global audit trail using internal data and named DTM properties exported from the DTM via the audit trail services.
Use Case:		**Adjust SetValue**
Brief Description:		Enables the actor to adjust the SetValues of a device (e.g. positioner, controllers).This may involve change of the modes on the device (e.g. change of block mode (MODE_BLK) or change of device mode). (DTM applicationId: AdjustSetValue)
GUI:		The component, which supports adjust SetValue, has to provide an adequate GUI.
Device Connection:		Shall have a connection established. (Adjust SetValue may influence device data)
UserLevel	Observer:	Accessible for viewing only
	Expert:	View, export and write. Frame Application may restrict access.
	Engineer:	View, export and write. Frame Application may restrict access.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Use Case:		Archive
Brief Description:		The “Archive" use case describes the access to a Frame Application archive for viewing and data analysis.
GUI:		The component, which supports archive functions, has to provide an adequate GUI.
Device Connection:		Not necessary
UserLevel	Observer:	No access
	Expert:	Accessible for viewing only
	Engineer:	View, export and delete. Frame Application may restrict access.
Included Use Case:		Historical Trend
Brief Description:		The archive operations may support visualization of historical data (for example trending).
Included Use Case:		Historical Data Analysis
Brief Description:		Some Frame Applications and DTMs may support extended operations to analyze historical data.

Maintenance use cases

The maintenance use cases are available for the actor “Expert” (see Figure A.4). These use cases focus on maintenance of devices.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Figure A.4 — Maintenance use cases

The Maintenance use cases are described in the following table (Table A.4).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table A.4 — Maintenance use cases

Use Case:		Simulation
Brief Description:		This use case simulates the behavior of a device. Therefore the actor is allowed to perform temporary changes within the device parameters, like forcing the device to set a fixed current analog output.
GUI:		DTM (application ID: Force)
Print:		DTM-specific
Device Connection:		Shall have a connection established
UserLevel	Observer:	No access
	Expert:	
	Engineer:	Accessible. Frame Application may restrict access.
Use Case:		**Offline Operation**
Brief Description:		This use case includes all operations which do not require an established connection to a device. Only for up- and download a connection to a device may be temporarily established.
GUI:		DTM and Frame Application
Print:		DTM
Device Connection:		Depends on the included use case
UserLevel	Observer:	No access
	Expert:	
	Engineer:	Accessible.
Included Use Case:		**Plausibility Check**
Brief Description:		Every time the parameter values are changed a plausibility check is automatically performed. As long as the plausibility check fails, the data cannot be saved to the database or written to the device. There may be two options depending on rules or application environment: After display of a warning, inconsistent data may be stored. For safety reasons after display of a warning writing of data is prohibited.
Users:		The plausibility check may be more tolerant for actors of higher level.
Included Use Case:		**Offline Parameterization**
Brief Description:		The user may change parameter values. The changes will only affect data in the Frame Application database after stored as persistent data. Data should be signed as transient data until they are stored.
GUI:		DTM (applicationId: OfflineParameterize)
Print:		DTM
Device Connection:		No connection necessary
Included Use Case:		**Persistent Data Comparison**
Brief Description:		Allows the user to compare the offline parameter set with parameter sets of other instances, of device default or of project default parameter sets. The datasets may be editable and data may be transferred from one dataset to the other.
GUI:		DTM (applicationId: OfflineCompare)
Print:		May be supported by the DTM
Device Connection:		Not necessary

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Use Case:		Device Repair
Brief Description:		This use case stands for operations which shall be performed to repair or change a device. Example: A Frame Application supports a temporary deletion of a device with automatically parameterization download when the device has been reinstalled.
UserLevel	Observer:	No access
	Expert:	
	Engineer:	Accessible.
Use Case:		**DTM Instance Upgrade and Replacement**
Brief Description:		This use case stands for operations which shall be performed to upgrade or to replace a DTM. Upgrade or replacement is used when the new DTM is different to the previous DTM. The data format of the new DTM can be either compatible (Upgrade) or incompatible (Replacement) to the data format of the previous DTM.
UserLevel	Observer:	No access
	Expert:	
	Engineer:	Accessible.
Included Use Case:		**Replacement**
Brief Description:		This use case stands for operations which shall be performed to replace a DTM in the case of incompatible dataset format.
Included Use Case:		**Upgrade**
Brief Description:		This use case stands for operations which shall be performed to upgrade a DTM in the case of compatible dataset format.
Use Case:		**Online Operation**
Brief Description:		This use case includes all operations which are performed directly with the device.
GUI:		DTM-specific
Print:		DTM-specific
Device Connection:		Connected or disconnected
UserLevel	Observer:	No access
	Expert:	
	Engineer:	Accessible.
Included Use Case:		**Online Functions**
Brief Description:		The“"Online Functions“" use case offers all procedures which include direct communication with the device. The use case may include simple procedures like resetting but also more complex procedures with several steps like teach in.
Included Use Case:		**Online Parameterization**
Brief Description:		When this use case starts all parameters are read from the device and exposed to the user within a GUI. Within the GUI the user may change parameters, depending on the user level. The device is updated immediately if the changed parameterization is in a verified state. (DTM applicationId: OnlineParameterize)
Included Use Case:		**Device-/Persistent Data Comparison**
Brief Description:		This use case gives the user the possibility to compare persistent and device data. The user may edit data and copy between these two sources. (DTM applicationId: OnlineCompare)
Included Use Case:		**Calibration**
Brief Description:		This use case invokes calibration procedures to adjust the input for e.g. pressure transmitters or the current for the output. (DTM applicationId: Calibration)

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Included Use Case:	Plausibility Check
Brief Description:	Every time the device parameters or the configuration data is changed, a plausibility check is automatically performed. As long as the plausibility check fails, the data cannot be written to the device.
UserLevel, UserFlag:	The plausibility check may be more tolerant for actors of higher level.
Included Use Case:	**Upload**
Brief Description:	Reads the whole parameter set from the device into the DTM instance data. After this, the data may be stored in the Frame Application database. The DTM may need to do Upload to synchronize the data when the data is changed in the Instance or in the device. The upload list of parameters does not change it is still remains as specified by the vendor. The “IsChangeEnabled” flag is ignored and not changed by the DTM during the upload or download actions
GUI:	If the Frame Application supports up- and download for a group of devices, the Frame Application may offer a GUI for a better control of the upload process.
Print:	No support
Device Connection:	Needs a temporary connection
Included Use Case:	**Download**
Brief Description:	Writes the whole parameter set from the DTM instance data into the device.
GUI:	If the Frame Application supports up- and download for a group of devices, the Frame Application may offer a GUI for a better control of the download process. The DTM may need to do Download to synchronize the data when the data is changed in the Instance or in the device. The download list of parameters does not change it is still remains as specified by the vendor. The “IsChangeEnabled” flag is ignored and not changed by the DTM during the upload or download actions
Print:	No support
Device Connection:	Needs a temporary connection
Use Case:	**Bulk Operation**
Brief Description:	This use case stands for the possibility to handle a group of devices in one operation (e.g. up- and download, parameter adjustment or report generation)
GUI:	Frame Application
Print:	Not supported
Device Connection:	Needs a connection
UserLevel Observer:	No access
Expert:	
Engineer:	Accessible.
Included Use Case:	**Upload**
Brief Description:	Reads the whole parameterization from the devices of the group into the Frame Application database.
Included Use Case:	**Download**
Brief Description:	Writes the whole parameterization from the Frame Application database into a group of devices.

Planning use cases

The Planning use cases are available for the actor “Engineer” (see Figure A.5). These use cases focus on planning and management of the system topology.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Figure A.5 — Planning use cases

Table A.5 provides a description of the Planning use cases.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table A.5 — Planning use cases

Use Case:		DTM Instance Handling
Brief Description:		With this use case an "Engineer" actor can handle (E.g. instantiate, import, export, delete) a device instance in the Frame Application.
GUI:		Frame Application and DTM
Print:		Not supported
Device Connection:		Not necessary
UserLevel	Observer:	No access
	Expert:	
	Engineer:	Accessible. Frame Application may restrict access.
Included Use Case:		**Create Instance**
Brief Description:		In this use case a new device instance can be created and placed into the project. After creation of a new device instance, the device parameter set shall be instantiated. This action is performed by the DTM.
GUI:		Frame Application and DTM (if it supports device default selection)
Included Use Case:		**Import**
Brief Description:		The parameter set may be instantiated by importing data from a database (for example a template database) or by copying the parameter set from an already instantiated and verified device.
GUI:		FA
Included Use Case:		**Export**
Brief Description:		To copy data from one device instance to another, the device instance data can be exported.
GUI:		FA
Included Use Case:		**Delete Instance**
Brief Description:		Deletion of a device instance
GUI:		FA
Use Case:		**Device Configuration**
Brief Description:		This use case enables the “Engineer” actor to configure a complex device (this may include configuration of modules).
GUI:		DTM (DTM applicationId: Configuration)
Print:		May be supported
Device Connection:		Shall not have a connection established
UserLevel	Observer:	No access
	Expert:	
	Engineer:	Accessible. Frame Application may restrict access.
Use Case:		**System Generation**
Brief Description:		Use case to perform all necessary actions for creation of topology based on the result of scan.
GUI:		Frame Application and DTM
Print:		Frame Application and DTM
Device Connection:		Necessary
UserLevel	Observer:	No access
	Expert:	
	Engineer:	Accessible. Frame Application may restrict access.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Included Use Case:		Scan
Brief Description:		Retrieves the list of available devices with service Scan from a channel object. The channel object may be provided by a DTM or by Frame Application.
GUI:		The Frame Application may offer a GUI for representation of the list of devices.
Print:		No support
Device Connection:		Needs a temporary connection. (The channel shall have online access.)
Included Use Case:		**Network Management**
Brief Description:		Use case for network management purposes involves activities between different FDT Objects (e.g. Communication DTM and Device DTM). Related activities are for example address setting or setting of device identity information. A Communication DTM may offer functions to manage a network. (Communication DTM applicationId: NetworkManagement)
Included Use Case:		**DTM matching**
Brief Description:		Use case for finding a DTM that matches best the information retrieved in the Scan use case.
Included Use Case:		**Create Instance**
Brief Description:		In this use case a new device instance can be created and placed into the project. After creation of a new device instance, the device parameter set shall be instantiated. This action is performed by the DTM.
GUI:		Frame Application and DTM (if it supports device default selection)
Use Case:		**System Planning**
Brief Description:		Use case to perform all necessary actions for the editing of topology information.
GUI:		Frame Application and DTM
Print:		Frame Application and DTM
Device Connection:		Not necessary
UserLevel	Observer:	No access
	Expert:	
	Engineer:	Accessible. Frame Application may restrict access.
Included Use Case:		**Network Management**
Brief Description:		Use case for network management purposes, e.g. address setting, as function of a Communication DTM. (DTM applicationId: NetworkManagement)
Included Use Case:		**Bus master Configuration**
Brief Description:		Use case for configuration of bus master DTMs
Included Use Case:		**Channel Assignment**
Brief Description:		Use case for editing the FDT channel assignments. This use case include assignment of device IO values to variables in a DCS or PLC program and changes in the IO-Signal Information of a DTM (frame tag, marking as used)

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Use Case:		List of Supported Devices
Brief Description:		In this use case a list of all supported devices within a project can be viewed and edited. An “Engineer” actor can select a device from this list to create a new device instance.
GUI:		FA
Print:		No support
Device Connection:		No connection
UserLevel	Observer:	No access
	Expert:	
	Engineer:	Accessible. Frame Application may restrict access.

Main Operation

This use case is the composition of all use cases that are available for the respective user level. It allows that a DTM provides a compact User Interface integrating all functions. This use case has the DTM applicationId: MainOperation.

The DTM should provide all the integrated functions also as separate functions, which can be invoked by the Frame Application for individual use cases.

OEM Service

The “OEM Service” use cases may provide extended access to device functions. The implementation and function of the use case OEM Service is specific for a DTM.

Administration

The Administration use cases are available to users that have additional Administrator permissions. The implementation and related functions of the use case Administration are specific for a Frame Application. The general expectation is that such functions may include installation and de-installation of DTMs and the management of the device catalogue.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Annex B
(normative)

FDT interface definition and datatypes

This is a place holder page, used for reference to an external part of the specification.

For a definition of the FDT3.0 interfaces and datatypes see the separate document “FDT30 Interfaces and Data Types” which is provided in different file formats.

Available document formats are:

- MS Help 1: FDT30 Interfaces and Data Types.chm
- MS Help 3: FDT3_Interfaces_Datatypes_VS_100_en-us_5.cab

All these files provide the same contents.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Annex C
(normative)

Mapping of services to interface methods

C.1 General

This annex describes the mapping of IEC 62453-2 services to interface, methods and XML information.

IEC 62453-2 does not define whether it is optional or mandatory to implement the defined services. This definition is provided by the following sections.

IEC 62453-2 does not define whether the services are implemented as synchronous or asynchronous calls. This implementation specification (FDT3.0) uses the asynchronous model, if it is likely that execution of service takes a longer time. This part defines separated request, callback and canceling interfaces / methods for this kind of services. Synchronous services are implemented by a single interface / method.

C.2 DTM services

Table C.1 — General services

IEC 62453-2 service	FDT3.0 interface / method	
PrivateDialogEnabled	*not available*	*A DTM Business Logic is not allowed to open private dialogs or user interfaces. A DTM Business Logic shall always use the Frame Application user interface services (see IFrame.Ui property). See chapter 5.11.2*
SetLanguage	*Request/Response:*	*All user interface related functions rely on the property CurrentUICulture whereas other culture-dependent functions use the property CurrentCulture. For details see chapter 5.10*
SetSystemGuiLabel	*Request/Response:*	*IDtm3.DtmSystemGuiLabel*

Table C.2 — DTM service related to installation

IEC 62453-2	FDT3.0
<Not defined>	*Registration of DTM at the system. See chapter 9.5*
<Not defined>	*Check for new DTM installations or updates: see chapter 9.5.2*

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table C.3 — DTM service related to DTM Information

IEC 62453-2 service	FDT3.0 interface / method	
GetTypeInformation	*Request/Response:*	*IDtmInformation.GetDtmInfo*
	Request:	*IDtmInformation.BeginGetSupportedTypes*
	Response:	*IDtmInformation.EndGetSupportedTypes*
GetIdentificationInformation	*Request/Response:*	*IDtmInformation.GetDeviceIdentInfo*
HardwareInformation	*Request:*	*IHardwareInformation.BeginHardwareScan*
	Response:	*IHardwareInformation.EndHardwareScan*
	Cancel:	*IHardwareInformation.CancelHardwareScan*
GetActiveTypeInfo	*Property:*	*IDtm3.ActiveType*

Table C.4 — DTM services related to DTM state machine

IEC 62453-2 service	FDT3.0 interface / method	
Initialize	*Request/Response:*	*IDtm3.Init* *IDtm3.InitData* *IDtm3.BeginConfiguration* *IDtm3.EndConfiguration* *IDtm3.Run* *IDtm3.LoadData* *see chapter 6.3.2.2*
SetLinked CommunicationChannel	*Request/Response:*	*IDtm3.EnableCommunication2*
EnableCommunication	*Request/Response:*	*IDtm3.EnableCommunication2*
ReleaseLinked CommunicationChannel	*Request/Response:*	*IDtm3.DisableCommunication*
ClearInstanceData	*Request:*	*IDtm3.BeginRelease*
	Response:	*IDtm3.EndRelease*
Terminate	*Request:*	*IDtm3.BeginRelease*
	Response:	*IDtm3.EndRelease*

Table C.5 — DTM services related to function

IEC 62453-2 service	FDT3.0 interface / method	
GetFunctions	*Property:*	*IFunction.FunctionInfo*
InvokeFunction	*Request:*	*ICommandFunction.BeginExecute*
	Response:	*ICommandFunction.EndExecute*
GetGuiInformation	*Property:*	*IFunction.FunctionInfo* *DtmWebUiManifest*
OpenPresentation	*Request:*	*IDtmWebUiFunction.initAsync*
	Response:	*<callback>*
ClosePresentation	*Request:*	*IDtmWebUiFunction.closeAsync*
	Response:	*<callback>*

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table C.6 — DTM services related to documentation

IEC 62453-2 service	FDT3.0 interface / method	
GetDocumentation	*Request:*	*IReporting.BeginGenerateReport*
	Response:	*IReporting.EndGenerateReport*
	Cancel:	*IReporting.CancelGenerateReport*

Table C.7 — DTM services to access the instance data

IEC 62453-2 service	FDT3.0 interface / method	
InstanceDataInformation	*Request:*	*IInstanceData.BeginGetDataInfo*
	Response:	*IInstanceData.EndGetDataInfo*
	Event	*IInstanceData.DataInfoChanged* *IInstanceData.DataValueChanged* *IInstanceData.ModifiedInDtmChanged*
InstanceDataRead	*Request:*	*IInstanceData.BeginRead*
	Response:	*IInstanceData.EndRead*
InstanceDataWrite	*Request:*	*IInstanceData.BeginWrite*
	Response:	*IInstanceData.EndWrite*
<Not defined>	*Request:*	*IInstanceCustomConfiguration.BeginGetAllDataInfo*
	Response:	*IInstanceCustomConfiguration.EndGetAllDataInfo*
<Not defined>	*Request:*	*IInstanceCustomConfiguration.BeginEnableParameters*
	Response:	*IInstanceCustomConfiguration.EndEnableParameters*

Table C.8 — DTM services to access diagnosis

IEC 62453-2 service	FDT3.0 interface / method	
Verify	*Request/Response:*	*not available – DTM shall ensure dataset is always valid*
CompareDataVaueSet	*Request:*	*IComparison.BeginInstanceDataCompare*
	Response:	*IComparison.EndInstanceDataCompare*
	Cancel:	*IComparison.CancelInstanceDataCompare*

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table C.9 — DTM services to access to device data

IEC 62453-2 service	FDT3.0 interface / method	
DeviceDataInformation	*Request:*	*IDeviceData.BeginGetDataInfo*
	Response:	*IDeviceData.EndGetDataInfo*
	Cancel:	*IDeviceData.CancelGetDataInfo*
	Event	*IDeviceData.DataInfoChanged* *IDeviceData.ModifiedInDeviceChanged*
DeviceDataRead	*Request:*	*IDeviceData.BeginRead*
	Response:	*IDeviceData.EndRead*
	Cancel:	*IDeviceData.CancelRead*
DeviceDataWrite	*Request:*	*IDeviceData.BeginWrite*
	Response:	*IDeviceData.EndWrite*
	Cancel:	*IDeviceData.CancelWrite*
<Not defined>	*Request:*	*IDeviceCustomConfiguration.BeginGetAllDataInfo*
	Response:	*IDeviceCustomConfiguration.EndGetAllDataInfo*
<Not defined>	*Request:*	*IDeviceCustomConfiguration.BeginEnableParameters*
	Response:	*IDeviceCustomConfiguration.EndEnableParameters*

Table C.10 — DTM services related to network management information

IEC 62453-2 service	FDT3.0 interface / method	
NetworkManagementInfo Read	*Request/Response:*	*INetworkData.GetAddressInfo* *INetworkData.GetNetworkDataInfo*
NetworkManagementInfo Write	*Request/Response:*	*INetworkData.SetAddressInfo* *INetworkData.SetNetworkData*

Table C.11 — DTM services related to online operation

IEC 62453-2 service	FDT3.0 interface / method	
DeviceStatus	*Request:*	*IOnlineOperation.BeginReadDeviceStatus*
	Response:	*IOnlineOperation.EndReadDeviceStatus*
CompareValueDataSetWith-DeviceDataSet	*Request:*	*IComparison.BeginDeviceDataCompare*
	Response:	*IComparison.EndDeviceDataCompare*
	Cancel:	*IComparison.CancelDeviceDataCompare*
WriteDataToDevice	*Request:*	*IOnlineOperation.BeginWriteDataToDevice*
	Response:	*IOnlineOperation.EndWriteDataToDevice*
	Cancel:	*IOnlineOperation.CancelWriteDataToDevice*
ReadDataFromDevice	*Request:*	*IOnlineOperation.BeginReadDataFromDevice*
	Response:	*IOnlineOperation.EndReadDataFromDevice*
	Cancel:	*IOnlineOperation.CancelReadDataFromDevice*

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table C.12 — DTM services related to FDT-Channel objects

IEC 62453-2 service	FDT3.0 interface / method	
GetChannels	*Property:*	*IChannels.CommunicationChannelInfos*

Table C.13 — DTM services related to import and export

IEC 62453-2 service	FDT3.0 interface / method	
Export	*Request/Response:*	*not available – Frame Application is responsible to provide services to export DTM datasets*
Import	*Request/Response:*	*not available – Frame Application is responsible to provide services to import DTM datasets*

Table C.14 — DTM services related to data synchronization

IEC 62453-2 service	FDT3.0 interface / method	
OnLockInstanceData	*Event:*	*IDataset.TransactionStarted*
OnUnlockInstanceData	*Event:*	*IDataset.TransactionClosed*
OnInstanceDataChanged	*Event:*	*IDataset.TransactionCommitted*
OnChildInstanceData Changed	*Event:*	*IChildDtmEvents.InstanceDataInfoChanged* *IChildDtmEvents.InstanceDataValueChanged*

C.3 Presentation object services

The interactions and state control between Frame Application, Presentation object and DTM is technology dependent. The IEC 62453 series leaves it to the technology-specific parts (IEC/TR 62453-4z) to define necessary services. This document defines the interfaces / methods for state control of DTM UI (i.e. DTM WebUI) as shown in Table C.15.

Table C.15 — DTM UI state control

IEC 62453-2 service	FDT3.0 interface / method	
Initialize	Request:	`IDtmWebUiFunction`.initAsync (JavaScript)
	Response:	{callback function} initCallback (JavaScript)
Terminate	Request	`IDtmWebUiFunction`.closeAsync (JavaScript)
	Response:	{callback function} closeCallback (JavaScript)

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

C.4 General channel services

Table C.16 — General channel service

IEC 62453-2 service	FDT3.0 interface / method	
ReadChannelInformation	*Property:*	*IChannels.CommunicationChannelInfos* *IChannels.CommunicationChannels*
	Event:	*IChannels.ChannelsChanged*
WriteChannelInformation	*Request/Response:*	*not available*

C.5 Process channel services

Table C.17 — Channel services for IO related information

IEC 62453-2 service	FDT3.0 interface / method	
ReadChannelData	*Request:*	*IProcessData.BeginGetProcessData*
	Response:	*IProcessData.EndGetProcessData*
	Request:	*IProcessImage.BeginGetProcessImageInfo*
	Response:	*IProcessImage.EndGetProcessImageInfo*
	Cancel:	*IProcessImage.CancelGetProcessImageInfo*
	Property:	*IProcessImage.BusMasterInfo*
	Event:	*IProcessData.ProcessDataChanged* *IProcessImage.ProcessImageChanged*
WriteChannelData	*Request/Response:*	*IProcessData.SetIOSignalInfo* *IProcessImage.SetIOSignalInfo*

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

C.6 Communication Channel Services

Table C.18 — Channel services related to communication

IEC 62453-2 service	FDT3.0 interface / method	
GetSupportedProtocols	*Property:*	*ICommunicationChannel.SupportedProtocols*
Connect	*Request:*	*ICommunication.BeginConnect*
	Response:	*ICommunication.EndConnect*
	Cancel:	*ICommunication.CancelConnect*
Disconnect	*Request:*	*ICommunication.BeginDisconnect*
	Response:	*ICommunication.EndDisconnect*
AbortRequest	*Request:*	*ICommunication.BeginDisconnect*
	Response:	*ICommunication.EndDisconnect*
AbortIndication	*Callback:*	*abort callback passed in BeginConnect*
Transaction	*Request:*	*ICommunication.BeginCommunicationRequest*
	Response:	*ICommunication.EndCommunicationRequest*
	Cancel:	*ICommunication.CancelCommunicationRequest*
SequenceDefine	*Request/Response:*	*not available*
SequenceStart	*Request/Response:*	*not available*

Table C.19 — Channel services related sub-topology management

IEC 62453-2 service	FDT3.0 interface / method	
ValidateAddChild	*Request:*	*ISubTopology.BeginValidateAddChild*
	Response:	*ISubTopology.EndValidateAddChild*
ChildAdded	*Request:*	*ISubTopology.BeginChildAdded*
	Response:	*ISubTopology.EndChildAdded*
ValidateRemoveChild	*Request:*	*ISubTopology.BeginValidateRemoveChild*
	Response:	*ISubTopology.EndValidateRemoveChild*
ChildRemoved	*Request:*	*ISubTopology.BeginChildRemoved*
	Response:	*ISubTopology.EndChildRemoved*
SetChildrenAddresses	*Request:*	*ISubTopology.BeginSetChildrenAddresses*
	Response:	*ISubTopology.EndSetChildrenAddresses*

Table C.20 — Channel services related to functions

IEC 62453-2 service	FDT3.0 interface / method
GetFunctions	*<no mapping defined>*
GetGuiInformation	*<no mapping defined>*

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table C.21 — Channel services related to scan

IEC 62453-2 service	FDT3.0 interface / method	
Scan	*Request:*	*IScanning.BeginScanRequest*
	Response:	*IScanning.EndScanRequest*
	Cancel:	*IScanning.CancelScanRequest*

C.7 Frame Application Services

Table C.22 — FA services related to general event

IEC 62453-2 service	FDT3.0 interface / method	
OnErrorMessage	*Request/Response:*	*ITrace.TraceEvent* *IFrameUi.ShowMessageBox* Note: ITrace shall be used to report errors. IFrameUi can be used to show user messages in a message box.
OnProgress	*Request/Response:*	*IFrameUi.ShowProgress* *IProgressUi.UpdateProgress* Note: Specific progress information is provided by many asynchronous operations.
OnOnlineStatusChanged	*Event:*	*IDtm3.OnlineStateChanged*
OnFunctionChanged	*Event:*	*IFunction.FunctionsChanged*

Table C.23 — FA services related to topology management

IEC 62453-2 service	FDT3.0 interface / method	
GetDtmInfoList	*Request/Response:*	*ITopology.GetDtmInfoList*
CreateChild	*Request:*	*ITopology.BeginAddChild*
	Response:	*ITopology.EndAddChild*
DeleteChild	*Request:*	*ITopology.BeginRemoveChild*
	Response:	*ITopology.EndRemoveChild*
MoveChild	*Request:*	*ITopology.BeginMoveChild*
	Response:	*ITopology.EndMoveChild*
GetChildNodes	*Request/Response:*	*ITopology.GetChildNodes*
GetParentNodes	*Request/Response:*	*ITopology.GetParentNodes*
<Not defined>	*Request/Response:*	*ITopology.GetSiblingNodes*
GetDtm	*Request:*	*ITopology.BeginGetDtm*
	Response:	*ITopology.EndGetDtm*
ReleaseDtm	*Request/Response:*	*IDtmProxy.Dispose*

Table C.24 — FA services related to redundancy

IEC 62453-2 service	FDT3.0 interface / method
OnAddedRedundantChild	*<no mapping defined>*
OnRemovedRedundantChild	*<no mapping defined>*

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table C.25 — FA services related to storage of DTM data

IEC 62453-2 service	FDT3.0 interface / method
LoadInstanceData	*Request/Response:* *IDataSubset.ReadData*
SaveInstanceData	*Request/Response:* *IDatSubset.WriteData*
GetPrivateDtmStorageInfo	*Request/Response:* *IDataset.BulkData* Note: There is no private storage path. Instance-specific private data shall be stored in the BulkData part of the dataset.

Table C.26 — FA services related to DTM data synchronization

IEC 62453-2 service	FDT3.0 interface / method
LockInstanceData	*Request/Response:* *IDataset.StartTransaction*
UnlockInstanceData	*Request/Response:* *IDataset.CloseTransaction*
InstanceDataChanged	*Request/Response:* *IDataset.CommitTransaction*

Table C.27 — FA services related to presentation

IEC 62453-2 service	FDT3.0 interface / method
OpenPresentationRequest	*Request:* *IFrameUi.BeginOpenDtmUi* *Response:* *IFrameUi.EndOpenDtmUi*
ClosePresentationRequest	*Request:* *BeginCloseDtmUi* *Response:* *EndCloseDtmUi*
UserDialog	*Request:* *IFrameUi.BeginOpenDtmUiModal* *Response:* *IFrameUi.EndOpenDtmUiModal*

Table C.28 — FA services related to audit trail

IEC 62453-2 service	FDT3.0 interface / method
RecordAuditTrailEvent	*Request/Response:* *IAuditTrail.Notify*

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table C.29 — FA services related to sandboxing a DTM

Service	FDT3.0 interface / method
Unique human readable identifier of the DTM instance in the context of the Frame Application.	*Request/Response: IDtmProxy.DtmSystemGuiLabel*
Unique identification of DTM within the Frame Application.	*Request/Response: IDtmProxy.DtmSystemTag*
DTM Device Type of the DTM in the topology.	*Request/Response: IDtmProxy.DtmType*
[Conditional] Reference to the IComparison interface.	*Request/Response: IDtmProxy.Comparison*
[Conditional] Reference to the IDeviceData interface.	*Request/Response: IDtmProxy.DeviceData*
[Conditional] Reference to the IDtmMessaging interface.	*Request/Response: IDtmProxy.DtmMessaging*
Reference to the IHardwareInformation interface.	*Request/Response: IDtmProxy.HardwareInformation*
Reference to the IInstanceData interface.	*Request/Response: IDtmProxy.InstanceData*
Reference to the INetworkData interface.	*Request/Response: IDtmProxy.NetworkData*
Reference to the IPorts interface.	*Request/Response: IDtmProxy.Ports*
[Optional] Reference to the IProcessData interface.	*Request/Response: IDtmProxy.ProcessData*
[Optional] Reference to the IReporting interface.	*Request/Response: IDtmProxy.Reporting*
Returns an interface for multi role-based access	*Request/Response: IDtmProxy.GetDtmProxyRoleAccess()*

Table C.30 — FA services related to sandboxing a Communication Channel

Service	FDT3.0 interface / method
Reference to the ICommunication interface.	*ICommunicationChannelProxy.Communication*
Reference to the IScanning interface.	*ICommunicationChannelProxy.Scanning*
[Conditional] Reference to the ISubscription interface.	*ICommunicationChannelProxy.Subscription*
Gets a list of the supported protocols of the communication channel.	*ICommunicationChannelProxy.SupportedProtocols*

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Annex D
(normative)

FDT version interoperability guide

D.1 Overview

FDT is a component based standard, which is constantly enhanced to new improved versions. Control system environments typically run for 10 to 15 years in contrast. If hardware and software components in a control system have to be exchanged, a mixture of components designed for different FDT versions may emerge.

This raises the question, how components based on different FDT versions can cooperate. This annex goes further into this question and provides solutions.

D.2 General

This annex mainly deals with two end user requirements in regard to FDT version interoperability.

Persistence: Users require new versions of FDT components (Frame Applications and DTMs) to be able to load project data of older versions.

Component Interoperability: Users require components of different FDT versions to interoperate properly. DTMs of updated FDT versions are required to run in old Frame Applications and vice versa. Communication is required to work even if FDT versions of Device DTMs and Communication DTMs differ.

A limiting condition for future FDT enhancements is to ensure maximum compatibility of different FDT versions. This will result in a maximum interoperability of FDT components originally designed for different FDT versions.

FDT takes care about version interoperability on two different levels.

1. Specification level: The FDT specification targets full compatibility of different specification releases (e.g. datatype definitions of the newer releases are compatible with older versions). Properly designed FDT components will achieve compatibility without extra implementations.
2. Implementation level: FDT provides test tools to examine the FDT compliance of FDT components. Although mentioned here, this is not in the scope of this document.

The FDT version is composed by a major, a minor, a build number and a revision number (e.g. 1.2.0.3 where 1 is the major, 2 is the minor number, 0 the build number and 3 the revision number). Compatibility is defined slightly differently with respect to the major number, as follows.

- Compatibility between FDT components with the same FDT major version number is assured by the specification.
- Compatibility between FDT components with different FDT minor, build and revision numbers can be achieved by extra code paths. FDT will provide necessary information and mechanisms to support full compatibility at this level. The minor number or build number is increased if new functionality, dependent of its importance, is added to the FDT specification. The revision number is increased if a bugfix within the specification document or the type library is necessary.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

D.3 Component interoperability

Interoperability of Frame Application and DTM components is shown in Table D.1.

Table D.1 — Interoperability between components of different versions

Frame / DTM	1.2.x Windows	2.0 Windows	2.1 Windows	3.0	
				Windows	Mac / Linux
1.2.x	✓	✓	✓	(✓)	✗
2.0	✗	✓	✓	(✓)	✗
2.1	✗	✓	✓	(✓)	✗
3.0	✗	✗	✗	✓	✓
✓ - supported, (✓) – possible to support by Frame Application (product-specific), ✗ - not possible to support					

The FDT specification ensures that DTMs and Frame Applications of the same major version cooperate regardless of minor version, for example FDT version 1.2 is compatible to FDT version 1.2.1. In this case the additional functionality defined in the higher FDT version may not be provided.

The situation is different if the major version number changes.

A DTM with a higher major FDT version (than the Frame Application) may not function within a Frame Application with lower major FDT version. Interoperability in this case is optional. In order to function in the ‘old’ Frame Application the DTM needs to behave according to the FDT version provided by the Frame Application.

A Frame Application with a higher major version (than the DTM) may refuse to work with a DTM with lower major version. Interoperability in this case is optional, if not defined otherwise. In order to work with the ‘old’ DTM, the Frame Application needs to provide both sets of interfaces. In this situation Frame Application behaves like an ‘old’ FDT version toward the DTM (old version number, old datatypes, old XML schemas).

Conversion from FDT3 datatypes to FDT2 datatypes can be implemented by serialization/de-serialization.

FDT2 provides a concept allowing FDT2 Frame Applications products to integrate 1.2.x DTM (see Annex E of FDT2 specification).

To assure that Frame Applications detect only compatible installed DTMs, for each FDT major version separate category identification will be defined within the respective FDT specification. This assures that a Frame Application is able to detect installed DTMs for specific FDT major versions.

Interaction between DTMs is managed by the Frame Application. Since each DTM is sandboxed by the Frame Application, the interaction between DTMs is supported by the proxy interfaces provided by the Frame Application (see Table C.29 and Table C.30).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Annex E
(normative)
Definition of JavaScript APIs

E.1 General

The definition of the JavaScript API is provided in format of a TypeScript file, because this allows for support of different JavaScript versions and for development of WebUIs based on TypeScript-based toolkits.

E.2 Request / Response datatypes

Each call from client to server is wrapped into FdtFitsRequest object. FdtFitsRequest contains following information:

- Identifier (used to identify callback with response)
- DTM instance identifier (SystemTag)
- Function instance identifier (InvokeId of DTM WebUI)
- ServiceRequest
- Optional ServiceData

FdtFitsRequest objects are automatically serialized and de-serialized by FDT_FITS Framework into JSON format and in this form transferred from the DTM WebUI to the DTM BL.

JSON Format for service request:

```
// not used, because Cancel is not a message, but a different method
// enum ServiceRequestType {
//    SendMessages = "SendMessages",
//    CancelSendMessages = "CancelSendMessages"
//}

export interface FdtFitsRequest {
    id: string,
    systemTag: string,
    functionInvokeId: string,
//    serviceRequest: ServiceRequestType,
    serviceData?: Array<SendMessageRequest>
}
```

Each call from client to server is wrapped into FdtFitsResponse object. FdtFitsResponse contains following information:

- Identifier (used to find corresponding callback function)
- Error information in form of FdtFitsError object
- Optional Data
- Indication if the response is an event notification

FdtFitsResponse objects are automatically serialized and de-serialized into JSON format and in this form transferred from the DTM BL to the DTM WebUI.

JSON Format for service response:

```
export interface FdtFitsResponse{
```

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
    id: string, // corresponds to FdtFitsRequest.id
    error: FdtFitsError,
    responseData?: Array<DtmUiMessage>
//    isEvent: boolean - not needed, because events are handled by different eventhandlers
}


export interface FdtFitsError{
    responseCode: number,
    message?: string,
}
```

Events are formatted similar to FdtFitsResponse but without association with some concrete callback (Id is not used). The 'isEvent' data field is set to true for events.

E.3 Message datatypes

SendMessages

Sends messages as JSON object to the DTM business object. The JSON object shall correspond to the existing .NET message type from the DTM business object. The .NET JsonSerializer class is used to deserialization of such JSON message objects.

A request contains following data:

- Identifier : <…>
- DTM instance identifier. <SystemTag>
- Function instance identifier: <InvokeId of DTM WebUI>
- ServiceRequest: "SendMessages"
- ServiceData: Array of messages, each with a message type identifier

Different DTMs have different UI request/response/event messages, so the messageData structure changes with different DTM.

All the messages shall have the "messageType" property and the value shall reference the corresponding .NET message datatype in the following way: The value shall correspond to the full name of the class (including the namespace), such that the FA is able to find the corresponding class in the list of known messages (property IDtmUiMessaging.UiMessageTypes).

JSON Format for service data:

```
export interface SendMessageRequest{
    messageIndex: number,
    messageType: string,
    messageData: any
}
```

The messageData field contains data according to the messageType.

Example:

```
"serviceData":
[
   {
      "messageIndex": "101",
      "messageType": "CompanyX.ReadParametersRequest",
      "messageData" :
      {
         "parameterNames":["TAG"],
      }
   },
   ...
]
```

Note: The example is based on an example UI request message.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

The FdtFitResponse contains response data according to the requested service.

Example:

```
"ResponseData":
{
   {
      "messageIndex": "101",
      "messageType": "CompanyX.ReadParametersResponse",
      "messageData" :
      {
         "ParameterValues":[{"TAG": "TAG value"}, ...]
      }
   },
   ...
}
```

Note: The example is based on an example UI response message. Different DTMs have different UI request/response messages, so the response body changes with different DTMs.

E.4 API for WebUIs

```
/**
* namespace FdtWebUi
*/

declare namespace FdtWebUi
{

   /**
   * Enumeration for expressing the result of an Init request.
   * @enum {string} InitResult
   */
   const enum InitResult {
      /* The init request failed. */
      InitFailed,
      /* The init request  succeeded. */
      InitSucceeded }

   /**
   * Enumeration for expressing the result of an Close request.
   * @enum {string} CloseResult
   */
   const enum CloseResult {
      /* The close request failed. */
      CloseFailed,
      /* The close request succeeded. */
      CloseSucceeded }

   /**
   * Main interface of a DTM WebUI
   *
   * @interface IDtmWebUiFunction
   */
   interface IDtmWebUiFunction {

      /**
      * Initialization of the DTM WebUI
      *
      * @function initAsync
      * @argument {URL} wssUrl - target url of messages (server)
      * @argument {string} systemTag - target of message (DTM)
      * @argument {number} functionId - id of the function
      * @argument {string} initData - initialization of the UI, provided by manifest
      * @argument {string} invokeId - id of UI instance
      * @argument {boolean} showIdentificationArea - initial condition of identification area
      * @argument {callback function} closeMeRequestHandler - inform Frame to close the DTM WebUI
      * @argument {string} asyncId-id of the async method call
      * @argument {IDtmWebUiMessaging} webDataConnector - reference to the WebData connector
      * @argument {IWebTrace} trace - reference to the interface used for tracing
      * @argument {callback function} initCallback - asynchronous callback
      */
```

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
   initAsync(
      wssUrl: URL,      // target url of messages (server)
      systemTag: string,   // target of message (DTM)
      functionId: number,  // id of the function
      initData: string,    // initialization of the UI, provided by manifest
      invokeId: string,    // id of UI instance
      showIdentificationArea: boolean,
      closeMeRequestHandler: (this: void, e: Event) => void, // inform Frame to close the DTM WebUI
      asyncId: string,    // id of the async method call
      webDataConnector: IDtmWebUiMessaging, // reference to the WebData connector
      trace: IWebTrace,   // reference to the interface used for tracing
      initCallback: (this: void, asyncId: string, initResult: InitResult) => void // async callback
   ): void;

   /**
   * Closing the DTM WebUI
   *
   * @function closeAsync
   * @argument {string} invokeId - id of UI instance
   * @argument {string} asyncId - id of the async method call
   * @argument {callback function} closeCallback - asynchronous callback
   */
   closeAsync(
      invokeId: string,    // id of UI instance
      asyncId: string, // id of the async method call
      closeCallback: (this: void, asyncId: string, closeResult: CloseResult, dialogResult?: string)
               => void // async callback with optional dialogResult
   ): void;

}


/**
* Enumeration for expressing the result of a cancel request.
* @enum {string} CancelResult
*/
const enum CancelResult {
   /* The cancel failed. */
   CancelFailed,
   /* The cancel succeeded. */
   CancelSucceeded
}


/**
* Interface used for communication from the DTM WebUI to the DTM BL
*
* @interface IDtmWebUiMessaging
*/
interface IDtmWebUiMessaging {

   /**
   * Initialization of the WebData connector
   *
   * @function init
   * @argument {URL} wssUrl - target of messages (server)
   * @argument {string} systemTag - target of message (DTM)
   * @argument {number} functionId - id of the function
   * @argument {string} invokeId - id of UI instance
   * @argument {IDtmWebUiMessageEvents} dtmUiReference - Reference to DTM WebUi object that is
   *           responsible to handle messaging events
   */
   init(
      wssUrl: URL, // target of messages (server)
      systemTag: string, // target of message (DTM)
      functionId: number, // id of the function
      invokeId: string, // id of UI instance
      dtmUiReference: IDtmWebUiMessageEvents // Reference to DTM WebUi object that is responsible to
                                             // handle messaging events
   ): void;

   /**
   * Send request from DTM WebUI to the DTM BL
   *
   * @function sendMessagesAsync
```

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
      * @argument {FdtFitsRequest} data - request containing a list of messages
      * @argument {callback function} sendCallback - asynchronous callback
      */
      sendMessagesAsync(
         data: FdtFitsRequest, // list of messages
         sendCallback: (this: void, data: FdtFitsResponse) => void // async callback
      ): void;

      /**
      * Cancel the sending of messages
      * If a send request is canceled, the expectation is that the respective sendCallback is not
called.
      * Still, in some cases it might occur, that the sendCallback is called.
      *
      * @function cancelSendMessagesAsync
      * @argument {string} requestId - id of the send request
      * @argument {callback function} cancelCallback - asynchronous callback
      */
      cancelSendMessagesAsync(
         requestId: string,
         cancelCallback: (this: void, requestId: string, cancelResult: CancelResult) => void // async
                                                                                            // callback
      ) : void;

      /**
      * Termination of the WebData Connector
      *
      * @function terminate
      * @argument {string} invokeId - id of UI instance
      */
      terminate (
         invokeId: string, // id of the UI instance
      ): void;

   }

   /**
   * Interface of DTM WebUI for receiving events from the DTM BL
   *
   * @interface IDtmWebUiMessageEvents
   */
   interface IDtmWebUiMessageEvents {
      /**
      * Information that a DTM-specific event occurred.
      *
      * @function dtmSpecificEventOccured
      * @argument {Array<DtmUiMessage>} data - data that is provided with the event notification
      */
      dtmSpecificEventOccured(data: Array<DtmUiMessage>) : void;

      /**
      * Information that a transaction was closed.
      *
      * @function transactionClosed
      */
      transactionClosed() : void;

      /**
      * Information that a transaction was committed.
      *
      * @function transactionCommitted
      * @argument {Array<DataSubSetInfo>} data - data that is provided with the event notification
      */
      transactionCommitted(data: Array<DataSubSetInfo>) : void;

      /**
      * Information that a transaction was started.
      *
      * @function transactionStarted
      */
      transactionStarted() : void;
   }

}
```

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

E.5 API for tracing

```
/**
* namespace FdtWebUi
*/

declare namespace FdtWebUi
{
   /**
    * The message that can be passed into a logger. A string or a lambda returning one of
these.
    */
   export type StringLambdaType = string | (() => string);

   /**
    * The error that can be passed into a logger. An Error, null or a lambda returning one of
these.
    */
   export type ErrorLambdaType = Error | null | (() => Error | null);

   /**
    * The IWebTrace interface used for tracing.
    *
    * Usage: call debug(), trace(), info(), warn(), error() or critical() method to send a
    * message with desired logging level to the Frame Application.
    * All of these methods expect a context and message, optionally an error.
    * The context describes the message source which usually is an application / component /
    * class.
    * The 'debug, trace, [info], warn, error, critical' notation used for methods description
    * to visualize
    * the tracing level of desired method using square brackets.
    *
    * Sample: logger.debug("My.First.App", "Hello world");
    *         logger.error("My.First.App.ComponentX", "This is an error", new Error("fail"));
    */
   export interface IWebTrace {

      /**
       * Writes [debug], trace info, warn, error, critical messages to log.
       * @param context {StringLambdaType} Describes the message source which usually is an
       * application / component / class.
       * @param message {StringLambdaType} The message to write.
       * @param error {ErrorLambdaType} An optional error that belongs to the message.
       */
      debug(context: StringLambdaType, message: StringLambdaType, error?: ErrorLambdaType):
void;

      /**
       * Writes debug, [trace], info, warn, error, critical messages to log.
       * @param context {StringLambdaType} Describes the message source which usually is an
       * application / component / class.
       * @param message {StringLambdaType} The message to write.
       * @param error {ErrorLambdaType} An optional error that belongs to the message.
       */
      trace(context: StringLambdaType, message: StringLambdaType, error?: ErrorLambdaType):
void;

      /**
       * Writes debug, trace, [info], warn, error, critical messages to log.
       * @param context {StringLambdaType} Describes the message source which usually is an
       * application / component / class.
       * @param message {StringLambdaType} The message to write.
       * @param error {ErrorLambdaType} An optional error that belongs to the message.
       */
      info(context: StringLambdaType, message: StringLambdaType, error?: ErrorLambdaType):
void;

      /**
       * Writes debug, trace, info, [warn], error, critical messages to log.
       * @param context {StringLambdaType} Describes the message source which usually is an
       * application / component / class.
       * @param message {StringLambdaType} The message to write.
       * @param error {ErrorLambdaType} An optional error that belongs to the message.
       */
```

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
      warn(context: StringLambdaType, message: StringLambdaType, error?: ErrorLambdaType):
void;

      /**
       * Writes debug, trace, info, warn, [error], critical messages to log.
       * @param context {StringLambdaType} Describes the message source which usually is an
       * application / component / class.
       * @param message {StringLambdaType} The message to write.
       * @param error {ErrorLambdaType} An optional error that belongs to the message.
       */
      error(context: StringLambdaType, message: StringLambdaType, error?: ErrorLambdaType):
void;

      /**
       * Writes debug, trace, info, warn, error, [critical] messages to log.
       * @param context {StringLambdaType} Describes the message source which usually is an
       * application / component / class.
       * @param message {StringLambdaType} The message to write.
       * @param error {ErrorLambdaType} An optional error that belongs to the message.
       */
      critical(context: StringLambdaType, message: StringLambdaType, error?: ErrorLambdaType):
void;

      /**
       * Gets whether the debug logging level is enabled, i.e. debug messages will be written
       * to the log.
       */
      isDebugEnabled(): boolean;

      /**
       * Gets whether the trace logging level is enabled, i.e. trace messages will be written
       * to the log.
       */
      isTraceEnabled(): boolean;

      /**
       * Gets whether the info logging level is enabled, i.e. info messages will be written to
       * the log.
       */
      isInfoEnabled(): boolean;

      /**
       * Gets whether the warn logging level is enabled, i.e. warn messages will be written to
       * the log.
       */
      isWarnEnabled(): boolean;

      /**
       * Gets whether the error logging level is enabled, i.e. error messages will be written
       * to the log.
       */
      isErrorEnabled(): boolean;

      /**
       * Gets whether the critical logging level is enabled, i.e. critical messages will be
       * written to the log.
       */
      isCriticalEnabled(): boolean;

   }
}
```

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Annex F
(informative)
Implementation Hints

F.1 IAsyncResult pattern

In the implementation of the IAsyncResult pattern following rules have to be applied:

BeginOperationName

The BeginOperationName method begins asynchronous operation OperationName and returns an object that implements the IAsyncResult interface. IAsyncResult objects store information about an asynchronous operation.

A BeginOperationName method takes any parameters that are passed by value or by reference. Any out parameters are not part of the BeginOperationName method signature. The BeginOperationName method signature also includes two additional parameters. The first of these defines an AsyncCallback delegate that references a method that is called when the asynchronous operation completes. The caller can specify 'null' (Nothing in Visual Basic) if it does not want a method being invoked when the operation completes. The second additional parameter is a user-defined object. This object can be used to pass application-specific state information to the method invoked when the asynchronous operation completes. If a BeginOperationName method takes additional operation-specific parameters, such as a byte array to store bytes read from a file, the AsyncCallback and application state object are the last parameters in the BeginOperationName method signature.

BeginOperationName returns control to the calling thread immediately. If the BeginOperationName method throws exceptions, the exceptions are thrown before the asynchronous operation is started. If the BeginOperationName method throws exceptions, the callback method is not invoked.

EndOperationName

The EndOperationName method ends asynchronous operation OperationName and returns the operation results. The return value of the EndOperationName method is specific to the asynchronous operation. For example, the CloseTransaction(IAsyncResult) method returns an array TransactionResponse[] with the results of the requested transactions. The EndOperationName method takes any out or ref parameters declared in the signature of the synchronous version of the method. In addition, the EndOperationName method also includes an IAsyncResult parameter. Callers shall pass the instance returned by the corresponding call to BeginOperationName.

If the asynchronous operation represented by the IAsyncResult object has not completed when EndOperationName is called, EndOperationName blocks the calling thread until the asynchronous operation is complete. Exceptions occurring during the asynchronous operation are thrown from the EndOperationName method. The effect of calling the EndOperationName method multiple times with the same IAsyncResult is not defined. Likewise, calling the EndOperationName method with an IAsyncResult that was not returned by the related Begin method is also not defined.

For either of the undefined scenarios, the method shall throw an InvalidOperationException.

Implementers of this design pattern should notify the caller that the asynchronous operation completed by setting IsCompleted() to true, calling the asynchronous callback method (if one was specified) and signaling the AsyncWaitHandle().

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Asynchronous Completed callback

The asynchronous callback delegate Completed() (implemented by the client) can be provided in the BeginOperationName method call. When the operation is completed, the called object should notify the caller that the asynchronous operation completed by setting IsCompleted() to true, calling the asynchronous callback method (if one was specified) and signaling the AsyncWaitHandle().

F.2 Threading Best Practices

Consider the following guidelines [16]:

- Do not use ThreadAbort() to terminate other threads. Calling Abort on another thread is akin to throwing an exception on that thread, without knowing what point that thread has reached in its processing.
- Do not use ThreadSuspend() and ThreadResume() to synchronize the activities of multiple threads. Do use Mutex, ManualResetEvent, AutoResetEvent, and Monitor.
- Do not control the execution of worker threads from your main program (using events, for example). Instead, design your program so that worker threads are responsible for waiting until work is available, executing it, and notifying other parts of your program when finished. If your worker threads do not block, consider using thread pool threads. MonitorPulseAll(Object) is useful in situations where worker threads block.
- Do not use types as lock objects. That is, avoid code such as lock(typeof(X)) in C# or SyncLock(GetType(X)) in Visual Basic, or the use of MonitorEnter(Object) with Type objects. For a given type, there is only one instance of Type per application domain. If the type you take a lock on is public, code other than your own can take locks on it, leading to deadlocks.
- Use caution when locking on instances, for example lock(this) in C# or SyncLock(Me) in Visual Basic. If other code in your application, external to the type, takes a lock on the object, deadlocks could occur.
- Do ensure that a thread that has entered a monitor always leaves that monitor, even if an exception occurs while the thread is in the monitor. The C# lock statement and the Visual Basic SyncLock statement provide this behavior automatically, employing a finally block to ensure that MonitorExit(Object) is called. If you cannot ensure that Exit will be called, consider changing your design to use Mutex. A mutex is automatically released when the thread that currently owns it terminates.
- Do use multiple threads for tasks that require different resources, and avoid assigning multiple threads to a single resource. For example, any task involving I/O benefits from having its own thread, because that thread will block during I/O operations and thus allow other threads to execute. User input is another resource that benefits from a dedicated thread. On a single-processor computer, a task that involves intensive computation coexists with user input and with tasks that involve I/O, but multiple computation-intensive tasks contend with each other.
- Consider using methods of the Interlocked class for simple state changes, instead of using the lock statement (SyncLock in Visual Basic). The lock statement is a good general-purpose tool, but the Interlocked class provides better performance for updates that shall be atomic. Internally, it executes a single lock prefix if there is no contention.

F.3 Testing DTM messages in regard to JSON serialization/deserialization

It is possible to test whether a message type can be serialized/deserialized by the Frame with a simple test sequence. The test sequence will need to use a JSON to .NET conversion library. Several libraries are available (e.g. from the NuGet manager) for this purpose.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

The sample in Figure F.1 illustrates such a test script based on the Newtonsoft JSON.Net library.

```
var message = new Message
{
    PropA = "Hello",
    PropB = 6,
    PropC = new[] {"A", "B", "C"},
    AdditionalProps = new MessageProps
    {
        BaseProp = (float)1.6,
    }
};

string json =  JsonConvert.SerializeObject(message);

var clone = JsonConvert.DeserializeObject<Message>(json);
// => check for errors or incorrect data in the clone
```

Figure F.1 — Example: Test for JSON data

One particular pitfall in the design of such message types is the usage of base classes or even abstract base classes as property types. For instance Figure F.2 provides the .NET declaration of the class Message from the example in Figure F.1.

```
[DataContract]
public class MessageProps
{
    public float BaseProp { get; set; }
}

[DataContract]
public class Message : DtmRequestMessage
{
    [DataMember]
    public string PropA { get; set; }

    [DataMember]
    public int PropB { get; set; }

    [DataMember]
    public string[] PropC { get; set; }

    [DataMember]
    public MessageProps AdditionalProps { get; set; }
}
```

Figure F.2 — Example: Erroneous DTM-specific message example

Such a message would be serializable, but only if the value of the property AdditionalProps is actually of the type MessageProps. However, if the class MessageProps is a base class and has one or more derived classes, the deserialization process will produce incorrect output because it does not have the information about the actual (concrete) type of the value of the property. If the class MessageProps would be declared as abstract, it would even generate an exception during the deserialization.

So the variation in Figure F.3 of the sample code from Figure F.1 would not produce a correct result:

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

```
var message = new Message
{
    PropA = "Hello",
    PropB = 6,
    PropC = new[] {"A", "B", "C"},
    AdditionalProps = new MessagePropsA // MessagePropsA derives from MessageProps
    {
        BaseProp = (float)1.6,
               // Additional values declared in MessagePropsA
        PropC = "Hello again",
        PropD = 7,
    }
};

string json =  JsonConvert.SerializeObject(message);

var clone = JsonConvert.DeserializeObject<Message>(json);
// => clone.AdditionalProps will be of type MessageProps, not MessagePropsA as expected
//    So the values for PropC and PropD will be lost.
```

Figure F.3 — Example: Failing test for erroneous JSON data

A possible workaround for such type of problems is the usage of the type System.Dynamic.ExpandoObject. This is basically a generic object and the JSON deserializer can dynamically create properties on such an object. For more information please refer to the MSDN documentation.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Annex G
(informative)
Comparison of FDT2.1 and FDT3.0

This annex describes the changes between FDT2.1 and FDT3.0.

Concept	FDT2.1	FDT3
BL Technology	DTMs and Frame Applications are based on .NET Framework.	DTMs are based on .NET Standard library. Frame Applications may be based on .NET Core, .NET Framework or other .NET implementations.
DTM installation	DTM packages are based on Windows Installer	DTM package files are based on Open Packaging Convention.
UI Technology	DTM UI and Frame Application UI are based on .NET Framework (WinForm, WPF). A DTM also may provide DTM Application and DTM UI Command.	Frame Application may provide platform-specific UI (e.g. WPF) or Frame Application WebUI. DTM provides a DTM WebUI
Tiers of Frame Application	Frame Application mainly is described as monolith.	Frame Application is structured into Frame Application BL and Frame Application UI. The interfaces supported by these tiers are described.
Management of active DTM UIs	DTM-specific implementation	IDtmUiManagement allows the Frame Application to support management of DTM WebUIs in a distributed environment.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Annex H
(informative)

Trade names

The following trade names are used in this document:

FDT® is a registered trademark of the FDT Group AISBL.

HART® is the trade name of the product supplied by HART Communication Foundation. This information is given for convenience of users of this document and does not constitute an endorsement by FDT Group AISBL of the product named. Equivalent products may be used if they can be shown to lead to the same results.

ActiveX® is the trade name of a product supplied by Microsoft Corporation (Redmond, USA). This information is given for the convenience of users of this document and does not constitute an endorsement by FDT Group AISBL of the product named. Equivalent products may be used if they can be shown to lead to the same results.

Microsoft® is a trade name registered by Microsoft Corporation (Redmond, USA). This information is given for the convenience of users of this document and does not constitute an endorsement by FDT Group AISBL of the product named. Equivalent products may be used if they can be shown to lead to the same results.

Microsoft Windows® is the trade name of a product supplied by Microsoft Corporation (Redmond, USA). This information is given for the convenience of users of this document and does not constitute an endorsement by FDT Group AISBL of the product named. Equivalent products may be used if they can be shown to lead to the same results.

MSDN® is the trade name of a product supplied by Microsoft Corporation (Redmond, USA). This information is given for the convenience of users of this document and does not constitute an endorsement by FDT Group AISBL of the product named. Equivalent products may be used if they can be shown to lead to the same results.

Visual Basic® is the trade name of a product supplied by Microsoft Corporation (Redmond, USA). This information is given for the convenience of users of this document and does not constitute an endorsement by FDT Group AISBL of the product named. Equivalent products may be used if they can be shown to lead to the same results.

VC++® is the trade name of a product supplied by Microsoft Corporation (Redmond, USA). This information is given for the convenience of users of this document and does not constitute an endorsement by FDT Group AISBL of the product named. Equivalent products may be used if they can be shown to lead to the same results.

.NET is a trade name of a product supplied by Microsoft Corporation (Redmond, USA). This information is given for the convenience of users of this document and does not constitute an endorsement by FDT Group AISBL of the product named. Equivalent products may be used if they can be shown to lead to the same results.

Microsoft Authenticode® is a trade mark of a product supplied by Microsoft Corporation (Redmond, USA). This information is given for the convenience of users of this document and does not constitute an endorsement by FDT Group AISBL of the product named. Equivalent products may be used if they can be shown to lead to the same results.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Annex I
(informative)
UML Notation

I.1 General

Note
contains information additional to the UML diagram (or model). Notes provide additional context to help explain details that are not apparent in the diagram (see Figure I.1).

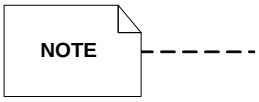


Figure I.1 — Note

I.2 Class diagram

The class diagram is one of the UML specification methods. The UML elements, which are used in the class diagrams of this specification, are explained in the following.

Class
is a description of a set of objects that share the same attributes, operations, methods, relationships, and semantics (see Figure I.2).

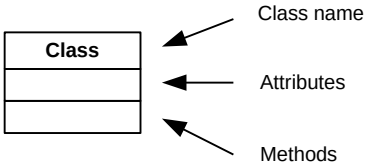


Figure I.2 — Class

The members of a class may be annotated with Icons, showing the type of members as indicated in Figure I.10.

Icon	UML stereotype
	<<property>>
	<<method>>
	<<event>>

Figure I.3 — Icons for class members

Abstract Class
is a class that cannot be directly instantiated and is used only for specification purposes. A class is abstract if it has no instances. An abstract class is used only to inherit from the class. Abstract classes are represented by an italicized class name.

Association
is a semantic relationship, between two or more classifiers, that specifies a link between their respective instances. (see Figure I.4)

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

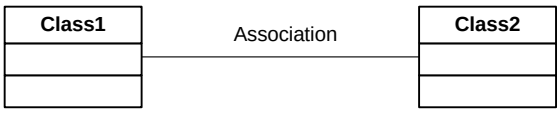


Figure I.4 — Association

An arrowhead on one end of a relationship denotes that the relationship is navigable in only one direction. A non-navigable end is indicated with a small x on the end of an association (see Figure I.5).

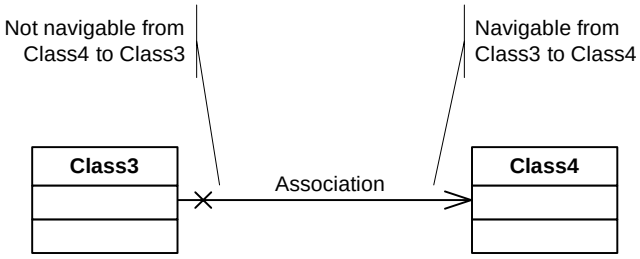


Figure I.5 — Navigable Association

Composition
is a form of symmetric association that specifies a whole-part relationship between the composition (whole) class and a subordinate (part) class in which removing the whole also removes the parts. (see Figure I.6)

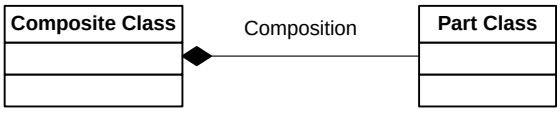


Figure I.6 — Composition

Aggregation
is a form of asymmetric association that specifies a whole-part relationship between the aggregate (whole) class and a subordinate (part) class. (see Figure I.7)

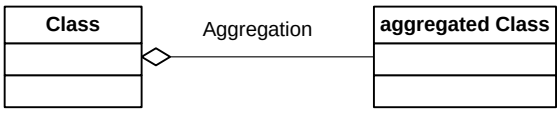


Figure I.7 — Aggregation

Dependency
is a form of association that specifies a dependency relationship between two classes. An arrowhead can be used to indicate an asymmetric dependency. (see Figure I.8)

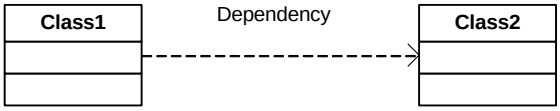


Figure I.8 — Dependency

Association class
provides additional information to an association, for instance attributes of that association or operations on the association. An association class may be used to explain how an association is managed (Figure I.9).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

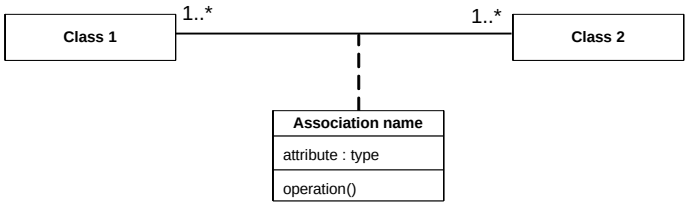


Figure I.9 — Association class

Generalization
is the taxonomic relationship between a more general element and a more specific element that is fully consistent with the first element and that adds additional information. It is used for classes, packages, use cases, and other elements. The construct is also used to describe Inheritance. (see Figure I.10)

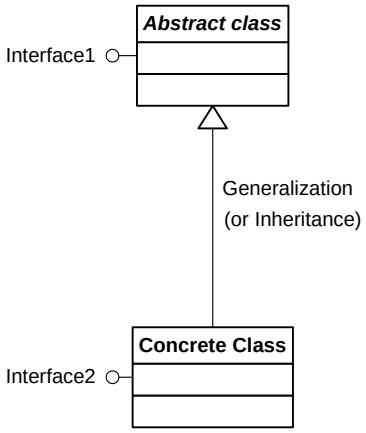


Figure I.10 — Abstract class, Generalization and Interface

Interface
is a mechanism used to conveniently package and reuse a collection of methods (method signatures) and constants. An interface is an abstract class that only contains method signatures and can also contain constants. There is no underlying implementation for the methods. Essentially, an interface is a promise to implement a standard package of methods and constants. An interface may be inherited by an abstract class as well as by a concrete class, which also may implement an interface. In Figure I.10 the Concrete Class implements Interface1 (inherited with Abstract Class) as well as Interface2. Figure I.11 shows additional notations related to interfaces. There are two different notations to show that a class implements an interface (Class1 implements Interface2) and a notation that shows if a class is accessing an interface (Class2 accesses methods on Interface3).

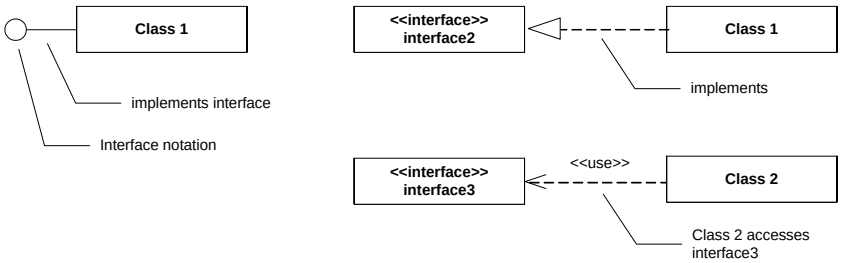


Figure I.11 — Interface related notations

Multiplicity
Multiplicity specification is shown as a text string comprising a comma-separated sequence of integer intervals (see Figure I.12), where an interval represents a (possibly infinite) range of integers, in the format: lower-bound .. upper-bound where lower-bound and upper-bound are

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

literal integer values, specifying the closed (inclusive) range of integers from the lower bound to the upper bound. In addition, the star character (*) may be used for the upper bound, denoting an unlimited upper bound.

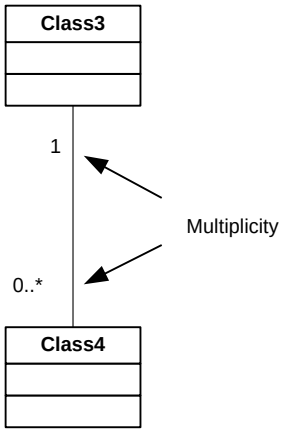


Figure I.12 — Multiplicity

Enumeration class

A form of class that acts as a container of enumeration literals. For instance, an enumeration ‘Color’ holds enumeration literals ‘red’, ‘green’ and ‘blue’.
This document uses enumeration classes to represent enumeration datatypes in UML diagrams.

The enumeration literals may be shown in the second compartment (see Figure I.13).

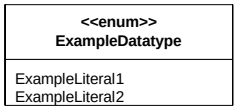


Figure I.13 — Enumeration datatype

I.3 Component diagram

A component diagram shows the relationships between different component of a system. It uses the elements of a class diagram and additional elements.

Component
is a class that represents an autonomous unit, which has interfaces to interact with the rest of the system (see Figure I.14). The internals are hidden or only accessible via the interfaces.

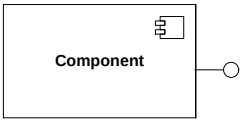


Figure I.14 — Component

I.4 State chart diagram

The state chart diagram is a graph that represents a state machine. States and various other types of vertices (pseudo states) in the state machine graph are rendered by appropriate state

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

and pseudo state symbols, while transitions are generally rendered by directed arcs that interconnect them (see Figure I.15).

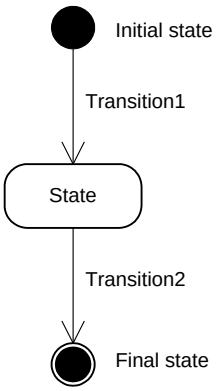


Figure I.15 — Elements of UML state chart diagrams

A **state** is shown as a rectangle with rounded corners. Optionally, it may have an attached name tab.

An **initial state** (pseudo state) is shown as a small solid filled circle.

A **final state** is shown as a circle surrounding a small solid filled circle (a bull's eye).

A **composite (super) state** is shown as a rectangle with rounded corners, containing two small ellipses (see Figure I.16). Such a state is composed from a set of sub-states, which in turn are connected by transitions. As an alternative syntax the super state is shown as a rectangle with rounded corners containing rectangles representing the substates.

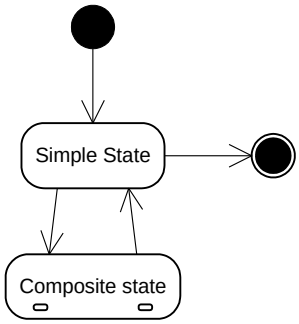


Figure I.16 — Example of UML state chart diagram

I.5 Use case diagram

A **use case diagram** is a class diagram for specifying required usages of a system. It contains actors, use cases and their relations. Figure I.17 shows the UML syntax that is used throughout this specification.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

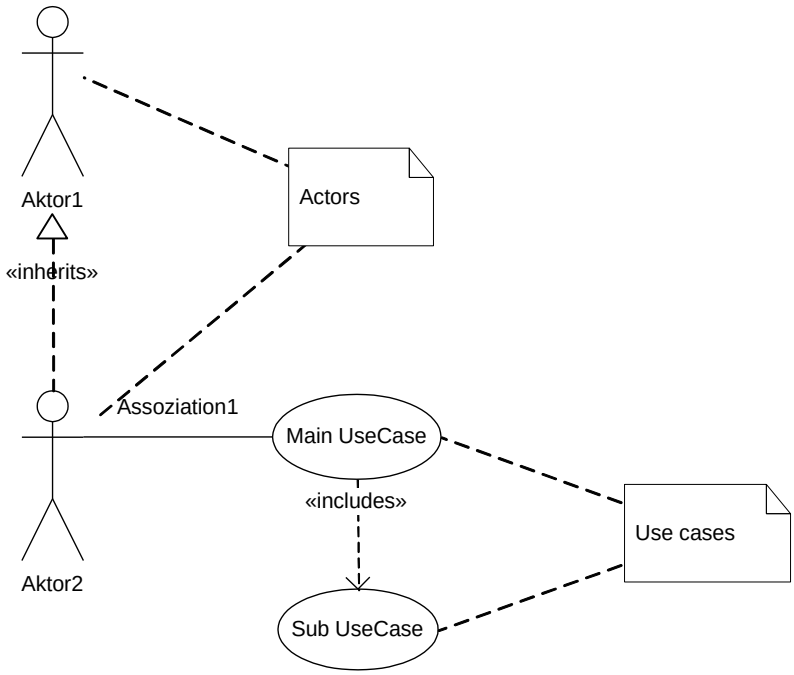


Figure I.17 — UML use case syntax

An **actor** is shown as person. It can represent a human being or an external other system interaction with the system which is specified.

A **use case** captures a functional requirement by way of describing the interaction between actor and the system. If a complex use case is composed from multiple simple use cases this is shown by ‘include’ relations.

An **inheritance** relation is shown as dashed line with a triangle at the parent, i.e. more general element. Inheritance between actors is used in this document to describe that one actor ‘inherits’ the permissions to execute certain use cases of another actor.

I.6 Sequence diagram

A diagram that depicts interactions by focusing on the sequence of messages between objects on the lifelines (see Figure I.18).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

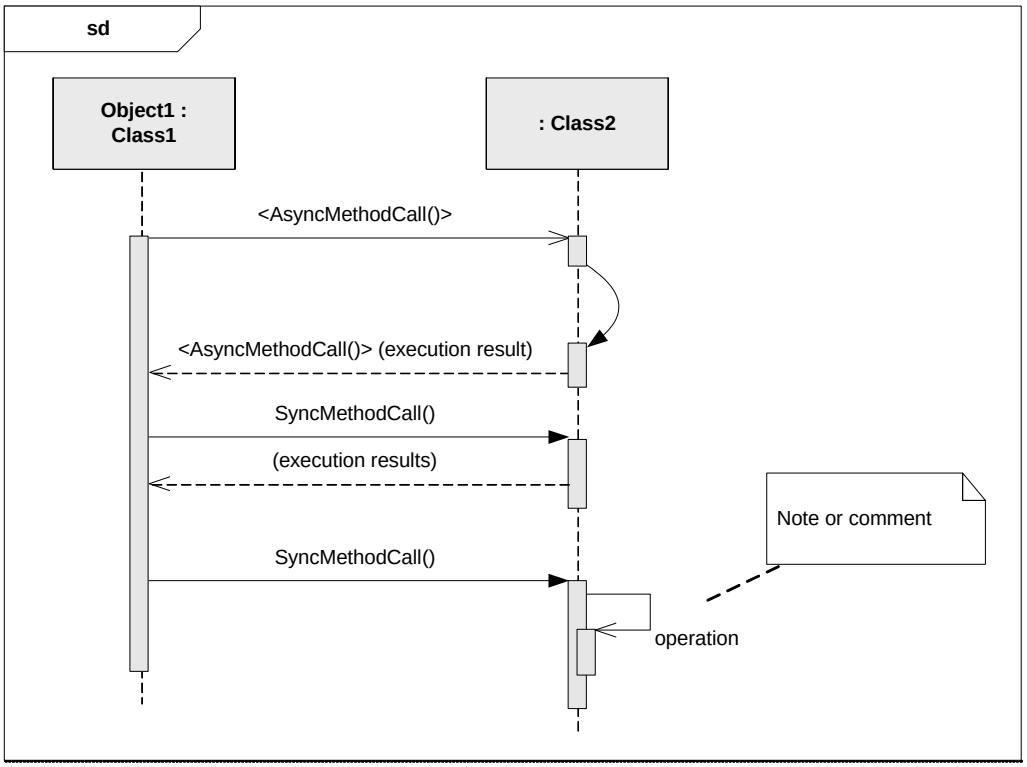


Figure I.18 — UML sequence diagram

Sequence diagrams are shown in a frame, which is a graphical boundary and shows type and name of the diagram (see Figure I.19).

Note: The diagram name may be referenced in other diagrams – see Figure I.29

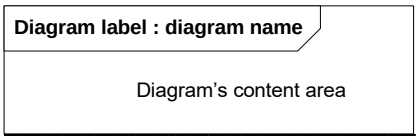


Figure I.19 — Empty UML sequence diagram frame

Object instances are represented by a vertical line (life line) as shown in Figure I.20. The object and class reference are denoted in the head. A rectangle on the life line indicates the activation of the object. The instance name of the object (in Figure I.20 "Object1") is shown only, when the distinction of different instances is relevant for understanding the diagram.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

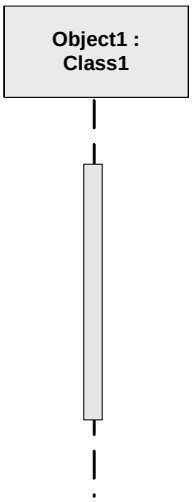


Figure I.20 — Object with life line and activation

A method call is represented by an arrow (see Figure I.21). In this specification a stick arrow is used to indicate a non-blocking method call, the method name is shown in angle brackets e.g. “<method()>”. A full arrow is used to depict the general occurrence of a blocking method call. The method name is shown as “method()”. In general no method arguments are shown. If a method argument is relevant for the sequence, it is shown with its name (in the example “argument1”). If the interface is relevant to understand the diagram (e.g. to identify the correct method), it may be shown preceding the method name, for example “interface1.method()”.

If there is no method defined (by FDT), the message shows that an activity is triggered. Such a message is labeled without parentheses, e.g. “trigger activity” or just “activity”.

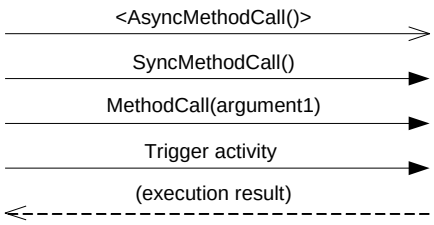


Figure I.21 — Method calls

The sequence of messages is defined by the order of messages starting from top of the diagram.

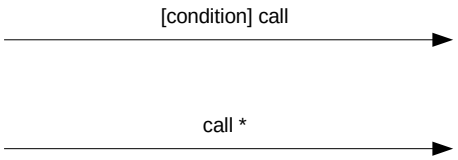


Figure I.22 — Modeling guarded call and multiple calls

Figure I.22 shows how calls are modeled, which are only called under certain conditions. Multiple calls are modeled with an appended ‘*’.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

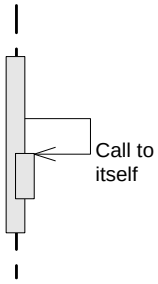


Figure I.23 — Call to itself

A call to itself indicates, that processing takes place in the system object (see Figure I.23).

A sequence diagram may be annotated by a state or by a constraint shown in an oval (see Figure I.24). This notation may be used to show two concepts: continuations and state invariants.

A continuation is shown at the beginning of a sequence. It shows a state or a constraint that must be valid in order to execute the sequence. This concept is used to show different possible behaviors depending on a state or condition.

A StateInvariant shows a state or a constraint which must be met in order for the sequence to be valid.

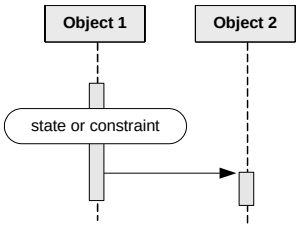


Figure I.24 — Continuation / StateInvariant

It is possible to describe alternative execution sequences in a sequence diagram.

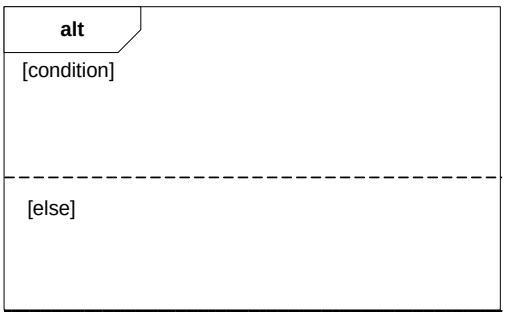


Figure I.25 — Alternative fragment

Alternatives are used for a mutually exclusive choice between two or more message sequences (see Figure I.25).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

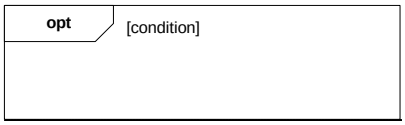


Figure I.26 — Option fragment

An option is used to model a simple "if then" statement (see Figure I.26).

Repetitive sequences are modeled with loop combination fragments as shown in Figure I.27.

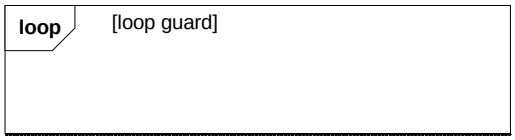


Figure I.27 — Loop combination fragment

In order to show the conditions for stopping execution of a loop, the break notation is used (see Figure I.28).

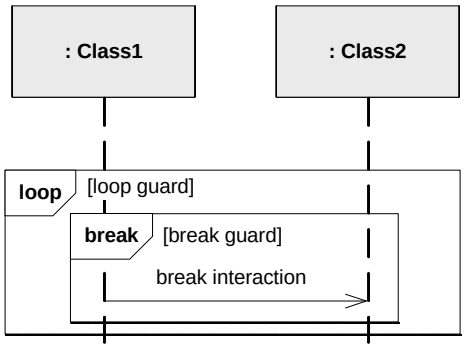


Figure I.28 — Break notation

Interaction references are used to refer to other sequence diagrams (see Figure I.29), which simplifies sequence diagrams and allows to reuse groups of coherent flows.

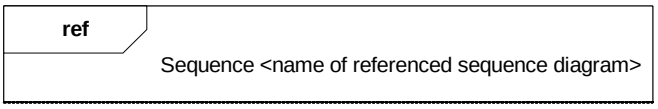


Figure I.29 — Sequence reference

I.7 Object diagram

Object diagrams provide a possibility to model instances of classes, associations, and attributes. A object diagram provides a snapshot of the system at runtime.

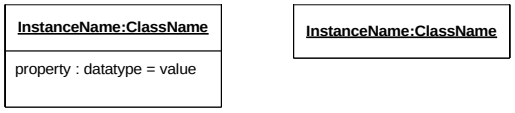


Figure I.30 — Objects

Figure I.30 shows two ways to model object instances.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

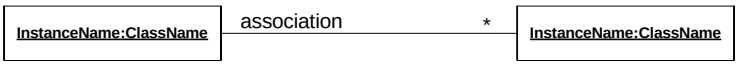


Figure I.31 — Object association

The association of two instances is shown in Figure I.31.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Annex J
(informative)
Physical Layer Examples

J.1 General

This annex provides examples for defined values for the PhysicalLayer element. It is recommended to use the same values for different fieldbuses, but it is not possible to provide a complete definition in this document. This list is provided for informational purposes only. For each fieldbus protocol the valid value range is defined in the respective FDT Protocol Annex document. When developing protocol support for a new fieldbus, existing protocol specifications for FDT3 should be considered.

J.2 Interbus S

D0D08238-B89C-11D9-AE7F-0000CB534BBC	LB ST
D0D08239-B89C-11D9-AE7F-0000CB534BBC	LB 2-wire
D0D0823A-B89C-11D9-AE7F-0000CB534BBC	LB Fiber Optics
D0D0823B-B89C-11D9-AE7F-0000CB534BBC	LB Inline
D0D0823C-B89C-11D9-AE7F-0000CB534BBC	LB Loop2
D0D0823D-B89C-11D9-AE7F-0000CB534BBC	LB Fieldline Modular
D0D0823E-B89C-11D9-AE7F-0000CB534BBC	LB Installations-Loop
D0D0823F-B89C-11D9-AE7F-0000CB534BBC	RB 2-wire
D0D08240-B89C-11D9-AE7F-0000CB534BBC	RB Fiber Optics
D0D08241-B89C-11D9-AE7F-0000CB534BBC	RB Fiber Optics HCS
D0D08242-B89C-11D9-AE7F-0000CB534BBC	RB Installation

J.3 PROFIBUS

036D1590-387B-11D4-86E1-00E0987270B9	IEC61158-2 (PROFIBUS PA)
036D1591-387B-11D4-86E1-00E0987270B9	RS485
036D1592-387B-11D4-86E1-00E0987270B9	Fiber

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

J.4 PROFINET

307DD808-C010-11DB-90E7-0002B3ECDCBE	10BASET	10 Mbit TwistedPair
307DD809-C010-11DB-90E7-0002B3ECDCBE	10BASETXHD	10 Mbit TwistedPair Half duplex
307DD80A-C010-11DB-90E7-0002B3ECDCBE	10BASETXFD	10 Mbit TwistedPair Full duplex
307DD80B-C010-11DB-90E7-0002B3ECDCBE	10BASEFLHD	10 Mbit Fiber-optic inter-repeater link Half duplex
307DD80C-C010-11DB-90E7-0002B3ECDCBE	10BASEFLFD	10 Mbit Fiber-optic inter-repeater link Full duplex
307DD80D-C010-11DB-90E7-0002B3ECDCBE	10BASEFXHD	10 Mbit X fiber over PMT, Half duplex
307DD80E-C010-11DB-90E7-0002B3ECDCBE	10BASEFXFD	10 Mbit X fiber over PMT, Full duplex
307DD80F-C010-11DB-90E7-0002B3ECDCBE	100BASETXHD	100 Mbit TwistedPair Half duplex
307DD810-C010-11DB-90E7-0002B3ECDCBE	100BASETXFD	100 Mbit TwistedPair Full duplex
307DD811-C010-11DB-90E7-0002B3ECDCBE	100BASEFXHD	100 Mbit X fiber over PMT, Half duplex
307DD812-C010-11DB-90E7-0002B3ECDCBE	100BASEFXFD	100 Mbit X fiber over PMT, Full duplex
307DD813-C010-11DB-90E7-0002B3ECDCBE	100BASELX10	100 Mbit Fiber-optic based on 2 single-mode fibers up to 10km
307DD814-C010-11DB-90E7-0002B3ECDCBE	100BASEPX10	100 Mbit Fiber-optic based on PONs up to 10km
307DD815-C010-11DB-90E7-0002B3ECDCBE	1000BASEXHD	PCS/PMA, unknown PMD, half duplex mode
307DD816-C010-11DB-90E7-0002B3ECDCBE	1000BASEXFD	PCS/PMA, unknown PMD, full duplex mode
307DD817-C010-11DB-90E7-0002B3ECDCBE	1000BASELXHD	Fiber over long-wavelength laser, half duplex mode
307DD818-C010-11DB-90E7-0002B3ECDCBE	1000BASELXFD	Fiber over long-wavelength laser, full duplex mode
307DD819-C010-11DB-90E7-0002B3ECDCBE	1000BASESXHD	Fiber over short-wavelength laser, half duplex mode
307DD81A-C010-11DB-90E7-0002B3ECDCBE	1000BASESXFD	Fiber over short-wavelength laser, full duplex mode
307DD81B-C010-11DB-90E7-0002B3ECDCBE	1000BASETHD	Four-pair Category 5 UTP, half duplex mode
307DD81C-C010-11DB-90E7-0002B3ECDCBE	1000BASETFD	Four-pair Category 5 UTP, half duplex mode
307DD81D-C010-11DB-90E7-0002B3ECDCBE	10GigBASEFX	10 Gbit X fiber over PMT

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Annex K
(informative)
Predefined SemanticIds

K.1 General

This annex provides an overview on pre-defined values for semantic ids and how they may be used. The ApplicationDomain for all these values is “FDT”.

Note: The ApplicationDomain “FDT” is reserved for semantic ids, which are defined by the FDT Group. Semantic ids defined by others shall use other ApplicationDomain values.

K.2 Data

The following SemanticIds are defined to be used with data (e.g. in IDeviceData and IInstanceData).

SemanticId	Explanation
slaveAddress (from FDT1.2.1)	The data contains the slaveAddress of the device.
busAddress (from FDT1.2.1)	The data contains the busAddress of the device.
busMasterConfigurationPart (from FDT1.2.1)	The data contains the busMasterConfiguration of the device.
productManufacturer	The data contains the name of the product manufacturer.

K.3 Images

The following SemanticIds are defined to be used with images (e.g. FdtBitmap and FdtIcon).

SemanticId	Explanation
conceptualImage	The bitmap/icon marked with this SemanticInfo contains a conceptual image (e.g. drawing).
photoRealisticImage	The bitmap/icon marked with this SemanticInfo contains a photography.

K.4 Documents

The following SemanticIds are defined to be used with documents (in IDtmInformation).

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

SemanticId	Explanation
eClass	The document (e.g. a tender specification) contains a document in eClass format.
ETIM	The document (e.g. a tender specification) contains a document in ETIM format.
BMEcat	The document (e.g. a tender specification) contains a document in BMEcat format.
PROlist	The document (e.g. a tender specification) contains a document in PROlist format.
DescriptionFile	The document (a 'Technical document') contains a protocol-specific device description file.
UserManual	The document (a 'Help' document) contains a device manual.
InstallationManual	The document (a 'Help' document) contains information for mounting and installation of the device.
NetworkConfigurationManual	The document (a 'Help' document) contains information for configuring the device communication.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Annex L
(informative)
Standard StaticFunctions

L.1 General

This annex defines standard Static Functions. These functions are defined as standard functions to define common support for specific use cases and in order to allow Frame Applications easily to recognize support for these use cases.

If a DTM intends to support the described use cases, it shall implement the standard functions as described.

L.2 StaticFunction GetDeviceStatus

Use case: Low effort monitoring of device status as defined by Namur recommendation NE107. Intended to be used for monitoring of many devices in parallel.

Table L.1 shows the information provided in the StaticFunctionDescription.

Table L.1 — StaticFunctionDescription for GetDeviceStatus

<table>
<tr><th>Property</th><th>Value</th><th>Comment</th></tr>
<tr><td>SemanticInfo</td><td><table><tr><th>Property</th><th>Value</th></tr><tr><td>Application domain</td><td>“FdtStaticFunction”</td></tr><tr><td>SemanticID</td><td>“GetDeviceStatusId”</td></tr></table></td><td></td></tr>
<tr><td>Descriptor</td><td>“Asynchronous operation to read current status of the device.”</td><td></td></tr>
<tr><td>FunctionId</td><td><…></td><td>DTM-specific value</td></tr>
<tr><td>InputParameters</td><td>null (no input parameters)</td><td>No input arguments</td></tr>
<tr><td>Label</td><td>“Read Device Status”</td><td></td></tr>
<tr><td>ProtocolIds</td><td><…></td><td>DTM-specific values,
All protocols supported by the DTM should have an implementation of this Static Function (possibly with different StaticFunctionProviders).</td></tr>
<tr><td>ReturnParameters</td><td>FunctionArgumentDescription:<table><tr><th>Property</th><th>Value</th></tr><tr><td>DataType</td><td>“Fdt.Dtm.DeviceStatus, Fdt.Datatypes, Version=1.1.0.0, Culture=neutral, PublicKeyToken=…”</td></tr><tr><td>DefaultValue</td><td>null (no default value)</td></tr><tr><td>Descriptor</td><td>“The current status of the device.”</td></tr><tr><td>Id</td><td>1</td></tr><tr><td>IsOptional</td><td>False</td></tr><tr><td>Label</td><td>“Device Status”</td></tr></table></td><td>The datatype will be described with the fully qualified datatype according to the respective FDT assembly.</td></tr>
</table>

Table L.2 shows the arguments for starting execution of the Static Function (only the functionId is provided).

Table L.2 — IStaticFunction2:BeginExecute arguments for GetDeviceStatus

Property	Value	Comment
FunctionId	<…>	DTM-specific value
Parameters	null (no input parameters)	No input arguments
…		

Possible exceptions specific for this Static Function are shown in Table L.3.

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

(If one of these exceptions occurs, it will be transported as the inner exception of Fdt.FdtOperationFailedException.)

Table L.3 — Exceptions for GetDeviceStatus

Exception	Condition
Fdt.FdtOperationFailedException	The operation failed.
Fdt.FdtDeviceTypeNotSupportedException	The type of the connected device is not supported by the Static Function.
Fdt.FdtCommunicationErrorException	One of the communication transactions fails with communication errors.
Fdt.FdtConnectRefusedException	The connect request has been refused.
Fdt.FdtInvalidUserPermissionsException	The operation is not allowed with the current user permissions.
Fdt.FdtConnectionAbortedException	The connection has been aborted.
Fdt.FdtInvalidReferenceException	The reference asyncResult is invalid.

L.3 StaticFunction GetProcessValue

Use case: This method returns the current value of a process variable of the device. If the device supports multiple process variables, the respective process variable is selected by the input argument of the method.

Only process values, which are described in the ProcessDataInfo of the DTM can be retrieved with this method.

Table L.4 — StaticFunctionDescription for GetProcessValue

<table>
<tr><th>Property</th><th>Value</th><th>Comment</th></tr>
<tr><td>ApplicationId</td><td><table><tr><th>Property</th><th>Value</th></tr><tr><td>Application domain</td><td>“FdtStaticFunction”</td></tr><tr><td>SemanticID</td><td>“GetProcessValueId”</td></tr></table></td><td></td></tr>
<tr><td>Descriptor</td><td>“Asynchronous operation to read current value of a process variable of the device.”</td><td></td></tr>
<tr><td>FunctionId</td><td><…></td><td>DTM-specific value</td></tr>
<tr><td>InputParameters</td><td>FunctionArgumentDescription:<table><tr><th>Property</th><th>Value</th></tr><tr><td>DataType</td><td>“System.String, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=…”</td></tr><tr><td>DefaultValue</td><td>null (no default value)</td></tr><tr><td>Descriptor</td><td>“Unique identifier of this IO signal within the ProcessDataInfo where it's contained.”</td></tr><tr><td>Id</td><td>1</td></tr><tr><td>IsOptional</td><td>False</td></tr><tr><td>Label</td><td>“IO Signal Id”</td></tr></table></td><td>Selector of the process variable. The Id value is provided by the DTM in the ProcessDataInfo (IProcessData.<GetProcessData>) .</td></tr>
<tr><td>Label</td><td>“Read Process Variable”</td><td></td></tr>
<tr><td>ProtocolIds</td><td><…></td><td>DTM-specific values,
All protocols supported by the DTM</td></tr>
<tr><td>ReturnParameters</td><td>FunctionArgumentDescription:<table><tr><th>Property</th><th>Value</th></tr><tr><td>DataType</td><td>“Fdt.Dtm.DataAccess.ResponseInfo, Fdt.Datatypes, Version=1.1.0.0, Culture=neutral, PublicKeyToken=…”</td></tr><tr><td>DefaultValue</td><td><no default value></td></tr><tr><td>Descriptor</td><td>“The current process value of the device.”</td></tr><tr><td>Id</td><td>2</td></tr><tr><td>IsOptional</td><td>False</td></tr><tr><td>Label</td><td>“Process Value”</td></tr></table></td><td>Provides the requested value together with additional information (e.g. Quality, TimeStamp, optionally ErrorInfo).</td></tr>
</table>

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Table L.5 — IStaticFunction2:BeginExecute arguments for GetDeviceStatus

Property	Value	Comment
FunctionId	<…>	DTM-specific value
Parameters	ArgumentId: 1 Value: <id of IO signal>	No input arguments
…		

Possible exceptions specific for this Static Function are shown in Table L.4.
(If one of these exceptions occurs, it will be transported as the inner exception of Fdt.FdtOperationFailedException.)

Table L.6 — Exceptions for GetProcessValue

Exception	Condition
Fdt.FdtOperationFailedException	The operation failed - e.g. because the operation is not supported.
Fdt.FdtDeviceTypeNotSupportedException	The type of the connected device is not supported by the Static Function
Fdt.FdtOperationCancelledException	CancelReadDataFromDevice(IAsyncResult)() was called.
Fdt.FdtCommunicationErrorException	One of the communication transactions fails with communication errors.
Fdt.FdtConnectRefusedException	The connect request has been refused.
Fdt.FdtInvalidUserPermissionsException	The operation is not allowed with the current user permissions.
Fdt.FdtLockDatasetException	The dataset can not be locked.
Fdt.FdtCommitTransactionFailedException	The commit transaction of the dataset fails.
Fdt.FdtConnectionAbortedException	The connection has been aborted.
Fdt.FdtInvalidReferenceException	Invalid asyncResult

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

Bibliography

[1] FDT V 1.2 – May 2003 – Addendum, PROFIBUS International

[2] FDT Specification V 1.2.1, FDT Group AISBL

[3] FDT2.1 Specification v1.01.000, FDT Group AISBL

[4] IEC 62453-1 Field Device Tool (FDT) interface specification – Part 1: Overview and Guidance

[5] IEC 62453-2 Field Device Tool (FDT) interface specification – Part 2: Concepts and Detailed Description

[6] FDT3.0 Styleguide for WebUI, FDT Group AISBL, (2020-06-04)

[7] .NET architectural components, Microsoft, https://docs.microsoft.com/en-us/dotnet/standard/components (2019-05-17)

[8] .NET Standard, Microsoft, https://docs.microsoft.com/en-us/dotnet/standard/net-standard (2019-05-16)

[9] .NET Core application deployment, Microsoft, https://docs.microsoft.com/en-us/dotnet/core/deploying/ (2019-05-16)

[10] NE107, Self-monitoring and diagnosis of field devices; Version: 01.03.2005; Namur

[11] Types Supported by the Data Contract Serializer, MSDN Library, .NET Framework 4, http://msdn.microsoft.com/en-us/library/ms731923.aspx (2010-06-30)

[12] Framework Design Guidelines: Conventions, Idioms, and Patterns for Reuseable .NET Libraries (Microsoft .NET Development); Krzysztof Cwalina, Brad Abrams, Addison-Wesley Longman

[13] OPC UA Specification, https://opcfoundation.org/developer-tools/specifications-unified-architecture

[14] OPC UA Information Model for FDT® Technology, FDT Group AISBL and OPC Foundation; (2016-11-22)

[15] Calling Asynchronous Methods Using IAsyncResult, MSDN Library, VS 9.0 Documentation, http://msdn.microsoft.com/en-us/library/ms228975.aspx; (2009-09-07)

[16] Managed Threading Best Practices, MSDN Library, .NET Framework 4, http://msdn.microsoft.com/en-us/library/1c9txz50(VS.100).aspx (2010-03-17)

[17] Handling and Raising Events, MSDN Library, .NET Framework 4.6 and 4.5, https://msdn.microsoft.com/en-us/library/edzehd2t.aspx (2010-03-18)

[18] 21 CFR Part 11 RIN 0910-AA29 Electronic Records; Electronic Signatures Part II, Final Rule, 62 CFR 13430, Mar. 20, 1997

[19] "XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition), http://www.w3.org/TR/2002/REC-xhtml1-20020801/"

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

[20] Authenticode, MSDN Library, Web Development, http://msdn.microsoft.com/en-us/library/ms537359%28VS.85%29.aspx (2016-12-15)

[21] Introduction to Code Signing, MSDN Library, https://msdn.microsoft.com/en-us/library/ms537361(v=vs.85).aspx (2017-11-30)

[22] How to get a Root Certificate update for Windows, https://support.microsoft.com/en-us/help/931125/how-to-get-a-root-certificate-update-for-windows (2017-11-30)

[23] Code Access Security, MSDN Library, .NET Framework Developer's Guide, http://msdn.microsoft.com/en-us/library/930b76w0%28VS.71%29.aspx (2010-06-25)

[24] XML Signature Syntax and Processing (Second Edition), W3C Recommendation 10 June 2008, http://www.w3.org/TR/xmldsig-core/ (2010-06-25)

[25] What Every Dev Must Know About Multithreaded Apps, Vance Morrison, MSDN Magazine, August 2005, https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Components.PostAttachments/00/10/67/06/52/MultiThreading.pdf (2018-09-12)

[26] KNOWNFOLDERID, MSDN library, Microsoft, http://msdn.microsoft.com/en-us/library/bb762584%28VS.85%29.aspx (2011-02-15)

[27] Environment.SpecialFolder, Microsoft, https://docs.microsoft.com/en-us/dotnet/api/system.environment.specialfolder?view=netframework-4.7.2 (2019-05-15)

[28] IEC Glossary, International Electrotechnical Commission, http://std.iec.ch/terms/terms.nsf/0/91A8B67C36ED9B16C125716900304B61?OpenDocument; (2011-02-17)

[29] Best Practice “Security techniques for Frame Applications”, FDT Group AISBL, document in preparation

[30] SignedXml Class, https://msdn.microsoft.com/en-us/library/system.security.cryptography.xml.signedxml(v=vs.110).aspx (2018-02-08)

[31] Technical Specification “FDT WebService Specification”, FDT Group AISBL, document in preparation

[32] ISO/IEC 29500-2:2012, Information technology -- Document description and processing languages -- Office Open XML File Formats -- Part 2: Open Packaging Conventions

[33] MSDN: Open Packaging Conventions Fundamentals, Microsoft, https://msdn.microsoft.com/en-us/library/windows/desktop/dd742818(v=vs.85).aspx (2019-05-16)

[34] LCID Structure, Microsoft, https://msdn.microsoft.com/en-us/library/cc233968.aspx (2019-05-16)

[35] Obtaining Certificates for DTM, Version 4.0 (document in development), FDT Group AISBL, document in preparation

[36] Creating localized NuGet packages, Microsoft, https://docs.microsoft.com/en-us/nuget/create-packages/creating-localized-packages (2020-02-13)

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.

[37] .NET Core RID Catalog, Microsoft, https://docs.microsoft.com/en-us/dotnet/core/rid-catalog (2020-03-18)

[38] Runtime IDs, Microsoft, https://github.com/dotnet/runtime/blob/master/src/libraries/pkg/Microsoft.NETCore.Platforms/runtime.json (2020-03-19)

Copyright 2020 by FDT Group, AISBL. All rights reserved. No portion of this document may be reproduced or redistributed without the express written consent of the FDT Group AISBL.