

# **FDT-Joint Interest Group Guideline**

## **FDT Interface Specification**

**Version 1.2.1**

Prepared by the FDT-JIG Working Group

Publisher:

FDT Joint Interest Group

[www.fdt-jig.org](http://www.fdt-jig.org)

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

## **FDT Preface**

### **Synopsis:**

This specification is an interface specification for developers of FDT 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 shall interoperate seamlessly.

### **Trademarks:**

Most computer and software brand names have trademarks or registered trademarks. The individual trademarks have not been listed here.

### **Required Runtime Environment:**

The implementation of the specified FDT components requires Microsoft Windows operating systems.

### **How to read this specification?**

You should read this document for your information to understand what FDT is dealing with and aiming for. Just use the inserted hyperlinks to navigate between terms and their definition.

### **Abstract:**

With the integration of fieldbusses into control systems, there are a few other tasks which must be performed. This applies to fieldbusses in general. Although there are fieldbus- and device-specific tools, there is no unified way to integrate those tools into higher-level system-wide planning- or engineering tools. In particular for use in extensive and heterogeneous control systems, typically in the area of the process industry, the unambiguous definition of engineering interfaces that are easy to use for all those involved, is of great importance.

### **Solution**

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 is in general open for all kind of fieldbusses and thus meets the requirements for integrating different kinds of devices into heterogeneous control systems.

### **Scope of the document**

The document is intended for developers who want to implement FDT components compatible to 1.2.1. For 1.2 compliant development previous versions of the specification must be used.

## Table of Contents

<b>1</b>	<b>INTRODUCTION.....</b>	<b>9</b>
1.1	Problem.....	9
1.2	State of the Art.....	9
1.3	Aims .....	10
1.4	Technological Orientation .....	11
1.5	Solution Concept.....	12
1.6	Migration to DTM .....	13
1.7	Scope of Performance .....	14
1.8	Audience.....	14
<b>2</b>	<b>FDT FUNDAMENTALS .....</b>	<b>15</b>
2.1	FDT Overview .....	15
2.2	Where FDT Fits.....	16
2.3	General FDT Architecture and Components .....	17
2.4	Overview of Objects and Interfaces.....	18
2.4.1	The Device Type Manager (DTM).....	18
2.4.2	The Block Type Manager (BTM) .....	19
2.4.3	The FDT Frame Application .....	20
2.5	Synchronization and Serialization Issues .....	21
2.6	Parameter interchange via XML .....	22
2.6.1	Examples of usage .....	23
2.7	Persistent Storage Story .....	25
2.7.1	Persistence Overview.....	25
2.7.2	Persistence Interfaces .....	26
2.8	Basic features of a session model.....	26
2.9	Basic Operation phases.....	26
2.9.1	Roles and Access Rights .....	27
2.9.2	Operation Phases.....	27
2.10	Abstract FDT Object Model .....	28
2.11	Fieldbus independent Integration .....	32
2.12	Scanning and DTM Assignment .....	32
<b>3</b>	<b>FDT VERSION INTEROPERABILITY GUIDE.....</b>	<b>33</b>
3.1	Abstract.....	33
3.2	Introduction .....	33
3.3	Component Interoperability.....	34
3.4	FDT Type Library.....	35
3.5	DTM and Device Versions .....	35
3.6	Persistence .....	35
3.7	Nested Communication .....	36
3.7.1	Data Exchange .....	36
3.7.2	Communication Channel Upgrade .....	36
3.7.3	Scenarios.....	36
3.7.4	OnAddChild .....	37
3.8	Implementation Hints .....	37
3.8.1	Interfaces.....	37
3.8.2	Persistence .....	37
<b>4</b>	<b>FDT INTERFACE SPECIFICATION.....</b>	<b>38</b>
4.1	Overview of the FDT Interfaces .....	38
4.2	FDT Objects.....	39
4.2.1	FDT Object Model .....	39
4.2.2	DTM State Machine.....	42
4.3	Device Type Manager.....	46
4.3.1	Interface IDtm .....	46
4.3.2	Interface IDtm2 .....	56
4.3.3	Interface IDtmActiveXInformation.....	58

4.3.4	Interface IDtmApplication .....	59
4.3.5	Interface IDtmChannel .....	61
4.3.6	Interface IDtmDocumentation.....	62
4.3.7	Interface IDtmDiagnosis .....	63
4.3.8	Interface IDtmImportExport .....	65
4.3.9	Interface IDtmInformation.....	67
4.3.10	Interface IDtmInformation2.....	68
4.3.11	Interface IDtmOnlineDiagnosis.....	69
4.3.12	Interface IDtmOnlineParameter.....	70
4.3.13	Interface IDtmParameter .....	73
4.3.14	Interface IFdtCommunicationEvents .....	74
4.3.15	Interface IFdtCommunicationEvents2 .....	77
4.3.16	Interface IFdtEvents .....	78
4.3.17	Interface IDtmHardwareIdentification.....	81
4.3.18	Interface IDtmSingleDeviceDataAccess .....	83
4.3.19	Interface IDtmSingleInstanceDataAccess .....	86
4.4	DTM ActiveXControl .....	89
4.4.1	Interface IDtmActiveXControl .....	89
4.5	FDT Channel.....	91
4.5.1	Interface IFdtChannel.....	91
4.5.2	Interface IFdtChannelActiveXInformation .....	94
4.5.3	Interface IFdtCommunication .....	96
4.5.4	Interface IFdtChannelSubTopology.....	103
4.5.5	Interface IFdtChannelSubTopology2.....	107
4.5.6	Interface IFdtChannelScan.....	108
4.5.7	Interface IFdtFunctionBlockData .....	109
4.6	FDT Channel ActiveXControl.....	112
4.6.1	Interface IFdtChannelActiveXControl.....	112
4.6.2	Interface IFdtChannelActiveXControl2.....	113
4.7	Block Type Manager.....	114
4.7.1	Interface IBtm .....	114
4.7.2	Interface IBtmInformation .....	116
4.7.3	Interface IBtmParameter .....	116
4.8	BTM ActiveXControl .....	117
4.8.1	Interface IBtmActiveXControl .....	117
4.9	Frame Application .....	119
4.9.1	Interface IDtmEvents.....	119
4.9.2	Interface IDtmEvents2.....	128
4.9.3	Interface IDtmScanEvents.....	129
4.9.4	Interface IDtmAuditTrailEvents.....	131
4.9.5	Interface IFdtActiveX .....	133
4.9.6	Interface IFdtActiveX2 .....	134
4.9.7	Interface IFdtBulkData.....	138
4.9.8	Interface IFdtContainer.....	140
4.9.9	Interface IFdtDialog .....	143
4.9.10	Interface IFdtTopology .....	144
4.9.11	Interface IDtmRedundancyEvents.....	149
4.9.12	Interface IDtmSingleDeviceDataAccessEvents .....	151
4.9.13	Interface IDtmSingleInstanceDataAccessEvents .....	154
4.9.14	Interface IFdtBtmTopology .....	155
4.10	General Concepts .....	157
4.10.1	Task Related FDT Interfaces .....	157
4.10.2	Return Values of Interface Methods.....	158
4.10.3	Dual Interfaces .....	159
4.10.4	Unicode .....	159
4.10.5	Asynchronous vs. Synchronous Behavior.....	159
4.10.6	ProglDs .....	160
4.10.7	Slave Redundancy .....	160
4.10.8	Field Bus Scanning and DTM Assignment.....	163

<b>5</b>	<b>FDT SESSION MODEL AND USE CASES.....</b>	<b>167</b>
5.1	Actors.....	169
5.2	Use cases .....	171
5.2.1	Observation .....	171
5.2.2	Operation.....	172
5.2.3	Maintenance.....	180
5.2.4	Planning.....	189
5.2.5	OEM Service .....	197
5.2.6	Administrator .....	198
5.3	DTM Use Case Realization .....	199
5.4	Frame Application Use Case Realization .....	204
<b>6</b>	<b>FDT SEQUENCE CHARTS .....</b>	<b>207</b>
6.1	DTM Peer To Peer Communication.....	207
6.1.1	Establish a Connection between DTM and Device .....	207
6.1.2	Asynchronous Connect .....	207
6.1.3	Asynchronous Disconnect.....	208
6.1.4	Asynchronous Transaction.....	209
6.2	Nested Communication .....	210
6.2.1	Generate Systemtopology.....	210
6.2.2	Establish a Connection between DTM and Device .....	213
6.2.3	Asynchronous Transaction.....	214
6.3	Topology Scan .....	215
6.3.1	Scan Network .....	215
6.3.2	Cancel Topology Scan .....	216
6.3.3	Provisional Scan Result Notifications.....	216
6.3.4	Scan for Communication Hardware .....	217
6.3.5	Manufacturer specific Device Identification .....	218
6.4	Registration of Protocol Specific FDT Schemas.....	220
6.5	Configuration of a Fieldbus Master.....	222
6.6	Starting and Releasing Applications.....	224
6.7	Channel Access.....	225
6.8	DCS Channel Assignment.....	226
6.9	Printing of DTM Specific Documents.....	230
6.10	Printing of Frame Application Specific Documents.....	231
6.11	Propagation of Changes .....	232
6.12	Locking.....	233
6.12.1	Locking for Non Synchronized DTMs.....	233
6.12.2	Locking for Synchronized DTMs .....	235
6.13	Instantiation and Release .....	237
6.13.1	Instantiation of a New DTM.....	237
6.13.2	Instantiation of an Existing DTM.....	237
6.13.3	Instantiation of a DTM User Interface.....	238
6.13.4	Release of a DTM User Interface.....	239
6.14	Persistent Storage of a DTM .....	240
6.14.1	State machine of instance data .....	240
6.14.2	Saving Instance Data of a DTM .....	241
6.14.3	Reload of a DTM Object for Another Instance .....	242
6.14.4	Copy and Versioning of a DTM Instance .....	243
6.15	Audit Trail.....	244
6.16	Comparison of Two Instance Data Sets .....	245
6.16.1	Comparison Without User Interface .....	245
6.16.2	Comparison With User Interface .....	246
6.17	Failsafe Data Access .....	247
6.18	Set or Modify Device Address With User Interface .....	248
6.19	Sets or Modifies Known Device Addresses Without User Interface.....	249
6.20	Display or Modify All Child Device Addresses With User Interface.....	250
6.21	Device Initiated Data Transfer .....	251
6.22	Starting and Releasing DTM User Interface in Modal Dialog.....	252

6.23 Parent Component Handling Redundant Slave.....	253
6.24 Initialization of a Channel ActiveX Control.....	255
6.24.1 Supports IFdtChannelActiveXControl2 .....	255
6.24.2 Does Not Support IFdtChannelActiveXControl2 .....	255
6.25 DTM Upgrade .....	256
6.25.1 Saving Data from a DTM to be Upgraded.....	256
6.25.2 Loading Data in the Replacement DTM.....	258
6.26 Usage of IDtmSingleDeviceDataAccess::ReadRequest / Write Request .....	259
6.27 Instantiation of DTM and BTM .....	260
<b>7 INSTALLATION ISSUES.....</b>	<b>262</b>
7.1 FDT Registry and Device Information.....	262
7.1.1 Visibility of business objects of a DTM.....	262
7.1.2 Component Categories .....	262
7.1.3 Registry Entries .....	263
7.1.4 Installation Issues .....	263
7.1.5 Microsoft's Standard Component Categories Manager .....	264
7.1.6 Building a Frame Application-Database of Supported Devices .....	264
7.1.7 DTM Registration.....	264
<b>8 DESCRIPTION OF DATA TYPES, PARAMETERS AND STRUCTURES.....</b>	<b>266</b>
8.1 Ids .....	266
8.2 Data Type Definitions .....	267
<b>9 GLOSSARY OF TERMS.....</b>	<b>268</b>
<b>10 LITERATURE.....</b>	<b>270</b>
<b>11 APPENDIX – FDT IDL .....</b>	<b>271</b>
<b>12 APPENDIX – FDT XML SCHEMAS .....</b>	<b>287</b>
12.1 FDTDataTypesSchema .....	287
12.2 FDTApplicationIdSchema .....	302
12.3 FDTUserInformationSchema .....	303
12.4 DTMInformationSchema .....	305
12.5 DTMFunctionCallSchema .....	309
12.6 DTMPParameterSchema .....	310
12.7 DTMDocumentationSchema.....	319
12.8 DTMProtocolsSchema .....	322
12.9 DTMSystemTagListSchema .....	323
12.10 DTMAuditTrailSchema.....	325
12.11 DTMDeviceStatusSchema .....	327
12.12 DTMFunctionsSchema .....	329
12.13 DTMChannelFunctionsSchema .....	333
12.14 DTMOnlineCompareSchema.....	336
12.15 FDTFailSafeDataSchema .....	337
12.16 DTMTopologyScanSchema .....	338
12.17 FDTOperationPhaseSchema.....	339
12.18 DTMInitSchema .....	340
12.19 FDTUserMessageSchema.....	341
12.20 DTMInfoListSchema.....	342
12.21 FDTTopologyImportExportSchema .....	343
12.22 DTMDeviceListSchema .....	347
12.23 DTMSystemGuiLabelSchema.....	350
12.24 DTMStateSchema.....	350
12.25 DTMEnvironmentSchema.....	351
12.26 FDTConnectResponseSchema .....	351
12.27 TypeRequestSchema .....	352
12.28 FDTScanRequestSchema .....	353

12.29	FDTxxxIdentSchema .....	354
12.30	FDTxxxDeviceTypIdentSchema .....	355
12.31	FDTxxxScanIdentSchema .....	355
12.32	DTMIdentSchema .....	355
12.33	DTMScanIdentSchema .....	356
12.34	DTMDeviceTypIdentSchema .....	359
12.35	DTMItemListSchema .....	361
12.36	BtmDataTypesSchema .....	366
12.37	BtmInformationSchema .....	368
12.38	BtmParameterSchema .....	369
12.39	BtmInitSchema .....	371
12.40	BtmInfoListSchema .....	372
<b>13</b>	<b>APPENDIX – FDT XML STYLES .....</b>	<b>373</b>
13.1	Documentation .....	373
<b>14</b>	<b>APPENDIX – FDT XSL TRANSFORMATION.....</b>	<b>378</b>
14.1	Identification Transformation .....	378
<b>15</b>	<b>APPENDIX – CHANNEL SCHEMA.....</b>	<b>380</b>
15.1	FDTBasicChannelParameterSchema .....	380
15.2	Template for Channel Schema .....	381
<b>16</b>	<b>APPENDIX – UPDATE HINTS .....</b>	<b>383</b>



# 1 Introduction

In this chapter, the motivation, the aim, and the solution for the FDT (Field Device Tool) concept are described.

## 1.1 Problem

In process automation, a control system often comprises more than 10,000 binary and analog input/output signals. When a fieldbus is used, these signals are transmitted via the bus. To this end, the field devices are connected directly to the bus or measured via remote I/O. More than 100 different field device types from various device manufacturers are frequently in use.

The devices are configured and parameterized for each task. The device-specific properties and settings must be taken into consideration when configuring the fieldbus coupler and the bus communication, and the devices must be made known to the control system. Input and output signals as well as function blocks provided by devices must be created and integrated into the function planning of the control system.

## 1.2 State of the Art

The large number of different device types and suppliers within a control system project makes the configuration task difficult and time-consuming today. Different tools must be mastered and data must be exchanged between these tools. The data exchange is not standardized. Therefore, data conversions are often necessary, requiring detailed specialist knowledge. In the end, the consistency of data, documentation and configurations can be guaranteed by an intensive system test only.

The central workplace for service and diagnostic tasks in the control system does neither fully cover the functional capabilities of the fieldbus devices nor can the different device-specific tools be integrated into the system's software tools. Typically, device-specific tools can only be connected directly to a fieldbus line or directly to the field device.

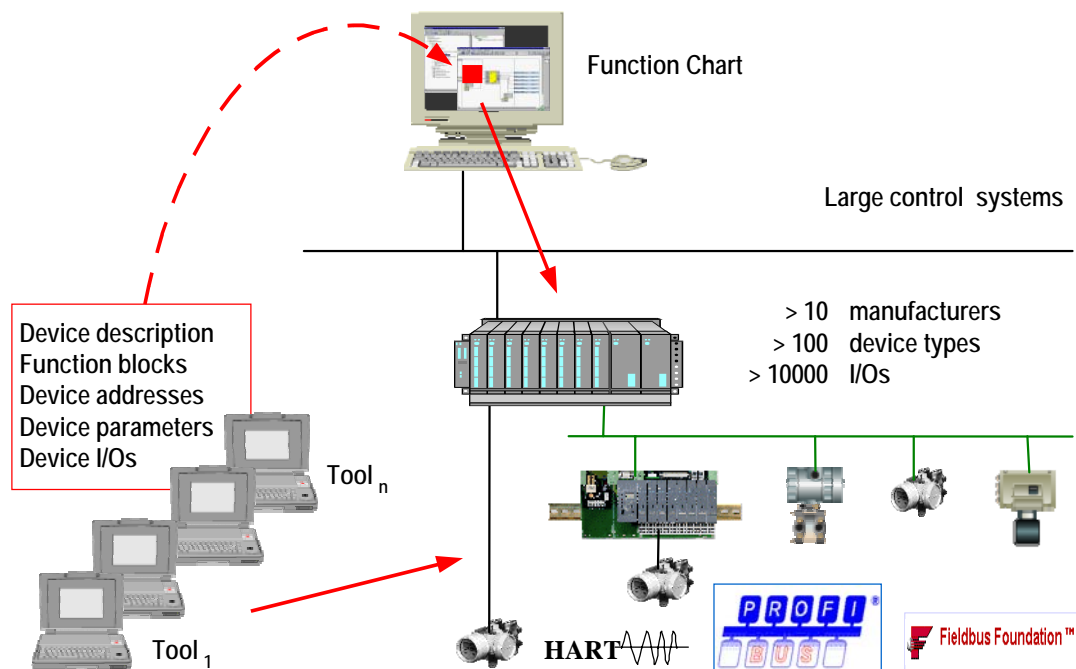


Figure 1: Different tools and multiple data input have determined field device integration to date

## 1.3 Aims

In order to maintain the continuity and operational reliability of process control technology, it is necessary to fully integrate fieldbus devices as a sub-component of process automation. The process control systems must provide the communication path from a central engineering- or operator-workplace via the system and fieldbusses to the individual field devices.

The main aims here are:

- Central workplace for planning, diagnostics and service with direct access to all field devices
- Integrated, consistent configuration and documentation of the process control system, the fieldbusses and devices
- Organization of common data for the process control system and the field devices
- Central data management and data security
- Simple, fast integration of different device types into the process control system

The integration of the field device technology into the engineering systems of process control technology is not only to be limited to a small, generally valid set of configuration, service and diagnostic functions - that would mean integration of the PROFILE definitions as a basic definition for field devices. Instead, the integration is to result in the individual device properties, characteristics and special features of the different device types being supported. The planning and service tools provided by the device manufacturer are to be integrated as device-specific software components into the engineering system. The device manufacturer defines the configuration, service and diagnostic functions for his devices himself and also designs the appearance of his devices in the engineering environment of the process control system.

It must then be possible for these components to be integrated into the engineering systems of all control technology suppliers. This reduces the costs for the device manufacturer, as he only needs to offer one standardized software component with all configuration, service and diagnostic functions for an intelligent field device. The frequent project-specific or control system-specific adaptations, which have to be developed and maintained over and over for one device type, are to be eliminated as a result of a standardized component technology.

The control system manufacturer has to implement the defined interfaces for the integration of all fieldbus devices only once. Manufacturer-specific and/or device-specific implementations and their maintenance are eliminated.

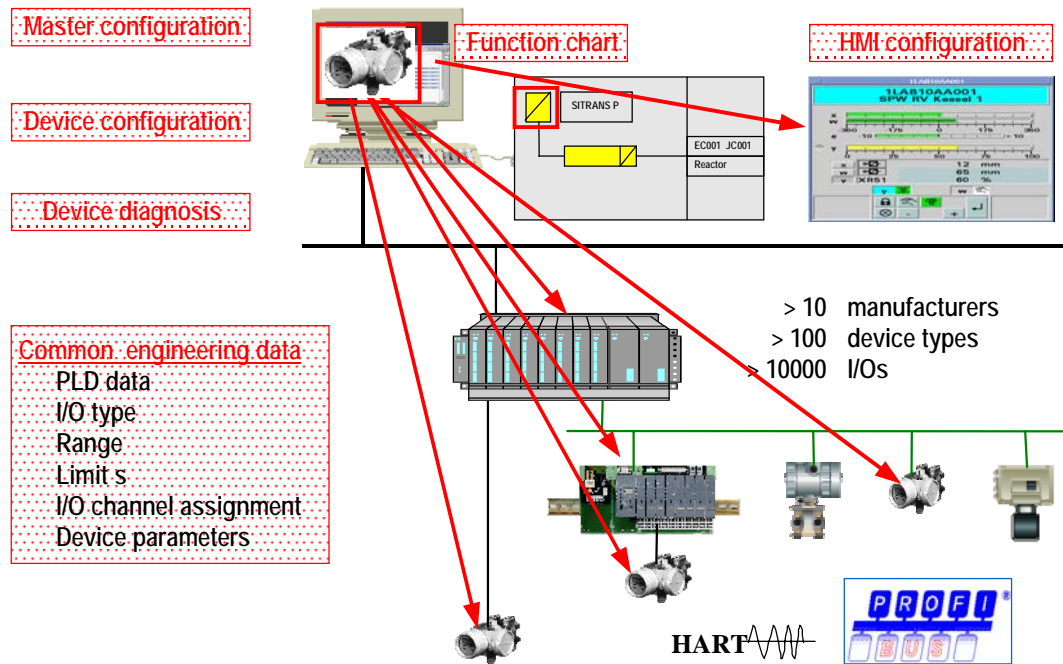


Figure 2: The potential of the field bus technology cannot be used until the field bus has been homogeneously integrated into the engineering systems.

## 1.4 Technological Orientation

Using the combination of a software component with a hardware component, like the driver software for a printer, the "Plug & Play" principle is also being introduced in fieldbus technology.

Moreover, as Microsoft Windows has established itself as the de-facto industry standard, there are few alternatives when choosing the operating system. The ActiveX technology introduced by Microsoft makes it possible to define interfaces which contain not only data but also functions. These possibilities have already been successfully used in conjunction with OPC (OLE for Process Control) definition. Within FDT only interfaces in ActiveX technology are specified for the engineering components of field devices. If the engineering system implements the corresponding interfaces, the ActiveX technology provides the automatic integration of the components and takes care of the interaction between the engineering system and the software components of the devices. Furthermore the FDT interface specification allows the integration of device components with integrated user interfaces as well as the embedding of ActiveX controls provided by the device component for special engineering tasks. The underlying object model is based on a client-server architecture that is easily extensible for future functionality. The data exchange between the devices' software components and the engineering system is performed by means of XML (eXtensible Markup Language) which also allows for easily extending the content of the information that is exchanged in later versions of FDT. The implementation of FDT's client server architecture is based on Microsoft COM.

## 1.5 Solution Concept

The FDT concept defines the interfaces between device-specific software components provided by the device supplier and the engineering tool of the control system manufacturer. The device-specific software component is called DTM (Device Type Manager).

The FDT concept can apply to any other application for handling field devices. However, the focus of the current version of FDT lies on engineering, commissioning, diagnostics and documentation of fieldbus-based control systems. Basic functionality is defined to achieve Audit Trail for applications like Asset Management.

The **Device Type Managers** are supplied by the device manufacturer together with the device. The following properties are characteristic for the DTM:

- It is generally no standalone tool
- ActiveX interfaces defined by the FDT-Spec.
- All rules of the device known
- All user dialogs contained
- User interface (multilingual including help system)
- Parameter validity check (also depending on other device-specific parameters)
- Automatic generation of dependent parameters
- Definition of the processing sequences of complex calibration, matching and setting procedures for high-quality field devices
- Reading and writing of parameters from/to the field device
- Diagnostic functions customized for the device
- Provision of the type-specific data for establishment of communication
- Provision of device/instance-specific data e.g. to be used in function planning
- Device / instance specific documentation
- No direct connection to any other device
- No information on the engineering environment
- Support for one or more device types

The quantity of functions (optimized by the device manufacturer for its device) listed here depends on the functional capabilities of the device. A DTM covers at least one field device. DTMs can, however, also cover device families (for example, pressure transducers), for example on the basis of Profiles or the entire palette of a manufacturer. Communication (via the various bus systems of a control system) and data management are handled via the interfaces of the engineering tool. Within the framework of overall system planning or plant management, a DTM must always be integrated into the appropriate engineering tool. Parallel standalone operation may be implemented in special cases for example when migrating from a standalone tool to a DTM. For reasons of data consistency, parallel operation of standalone solutions and DTMs running in the system's engineering tool accessing the same devices are not intended. Standalone operation may typically be used for testing purposes in a plant's workshop.

A DTM is installed as a component of an engineering tool or any other application that manages the device instances, provides the communication mechanisms and commissions the associated component with device-specific tasks. In the following, those applications are referred as 'Frame Applications'.

The following requirements apply to Frame Applications:

- No device-specific knowledge necessary
- Manages all device instances and stores instance data
- Creates the device communication and connection (tool routing)
- Guarantees system-wide consistent configuration
- Makes multi-user and server/client operation possible
- Takes care of data versioning

## **1.6 Migration to DTM**

Reflecting the current situation, there are a lot of different field devices ranging from simple I/O sensors to complex, modular Remote-I/Os or drives. According to their complexity, the devices can be divided into four categories:

- A: Simple devices that communicate only cyclically, for example a light barrier
- B: Adjustable devices with fixed hardware and software, for example a pressure transducer
- C: Adjustable devices with modular hardware but fixed software blocks, for example remote I/O
- D: Adjustable devices with modular hardware and programmable software blocks, for example a complex servo-drive

These different devices come with different kinds of descriptions of their capabilities or even their own configuration tools depending on the functionality the devices provide. With FDT all these devices can be integrated into Frame Applications via DTMs in a unified way even if the device manufacturer doesn't see his primary task in developing a DTM.

For instance, simple devices of categories A and B may be sufficiently described by already existing device descriptions or files containing information about the communication capabilities. It is possible to develop 'generic DTMs' that can interpret these device descriptions and make the contained information and functions available for the system and its user. Once a generic DTM for a specific device description is developed, all devices supporting this description can be integrated using the same DTM.

On the other hand, for devices of categories C and D there may be already existing standalone tools. FDT provides the openness to equip these tools with the FDT-interfaces and to build DTMs out of existing standalone tools. That way, the device manufacturer's investments can be protected. DTMs that are equipped with external tools are seen as a way to migrate towards FDT. The final goal of such a migration should be a DTM, which provides a well integrated ActiveX-based GUI.

In the long run, each device manufacturer has the freedom to choose from the following options for existing or new devices:

- Stay with the generic solutions based on device descriptions,
- Offer a DTM that is maintained as a standalone tool in parallel,
- Build a new DTM from scratch in order to introduce new features for handling the device.

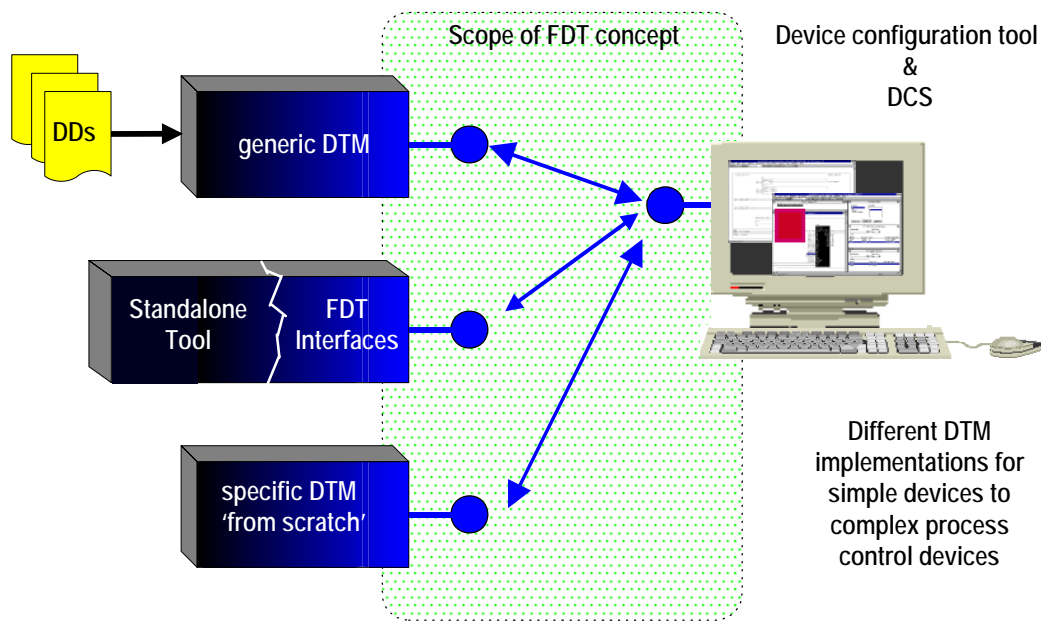


Figure 3: DTM - Implementations

## 1.7 Scope of Performance

The FDT concept does not claim to provide a solution for all engineering tasks and the associated tools. Therefore, the engineering subjects "electrical wiring planning, mechanical planning, etc." or the plant management subjects such as "maintenance, optimization, archiving, etc." do not form part of the current performance range of FDT. Some of these aspects may be included with future versions of FDT.

The scope of this version of the FDT concept includes the engineering and commissioning of field devices. Functionality for documentation and audit trail support is also considered. Examples and use cases are mainly given for actors, sensors and remote I/Os. Of course, FDT is prepared to be used for devices like drives, analyzers, recorders, etc. as well.

The FDT interfaces are designed in a way to support all significant fieldbus protocols. The main focus of this version of FDT does not lie on any specific fieldbus protocol. Due to the scalable structure of FDT it will be easy to add new schemas in order to extend the scope to new protocols.

The FDT concept allows a device manufacturer to provide his device specific functionality within an engineering system. So the main capacity of FDT depends on the functionality of the devices and the according applications.

## 1.8 Audience

This specification is intended as reference material for developers of FDT compliant Frame Applications and DTMs. It is assumed that the reader is familiar with Microsoft ActiveX technology and the needs of the Process Control industry or of the field devices.

This specification is intended to facilitate development of DTMs and Frame Applications in the language of choice that supports Microsoft ActiveX. Therefore, the developer of the respective component is expected to be familiar with the technology required for the specific component.

Remember, FDT is a client/server architecture implemented in a first step with Microsoft COM.

## 2 FDT Fundamentals

This chapter introduces the FDT model and covers topics which are specific to the requirements of field device integration.

### 2.1 FDT Overview

This specification describes the FDT Objects and their interfaces implemented by the Frame Application and the device specific applications called Device Type Manager (DTM).

DTMs can connect to monolithic Frame Applications or to Frame Applications made of different components provided by one or more vendors. Vice versa a Frame Application can support the integration of device specific DTMs of different vendors.

DTMs act as servers for device information and functionality. Different vendors may provide DTM Servers. Vendor-supplied code determines the device functionality and data to which the Frame Application has access.

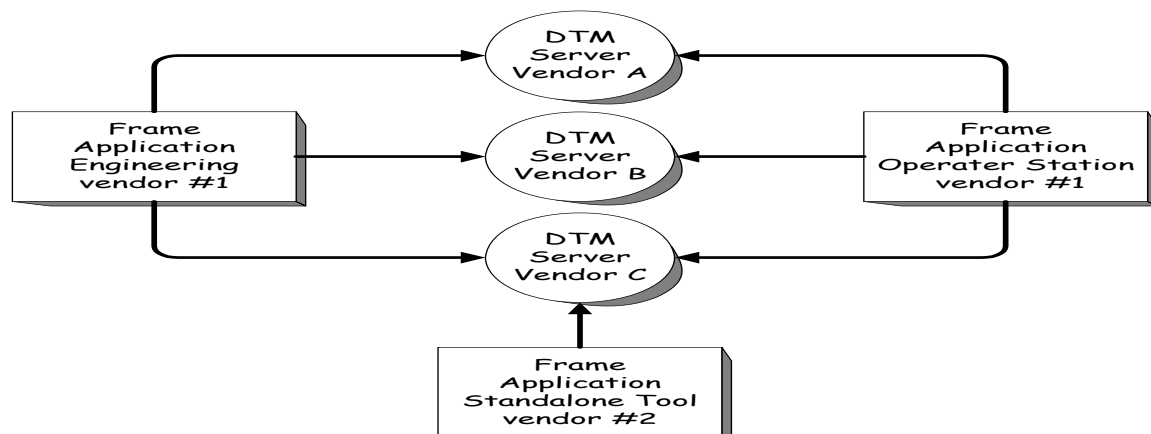
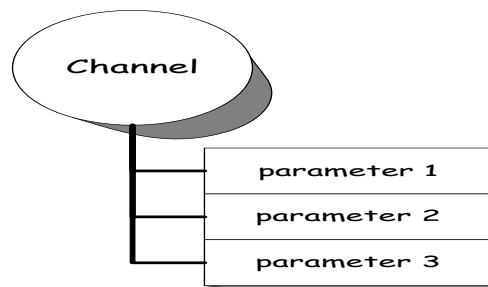


Figure 2-1 General FDT Client/Server Relationship

At a high level, a DTM is comprised of several objects: the server object and channel objects. This model represents a view on a device from an external application. There is the device itself that contains a certain set of functionality. To access the device's functionality, data must be exchanged between device and environment via communication channels. These channels may be identical to I/O connections of a Remote-I/O or to different process values measured by a transmitter and communicated via the fieldbus.

According to this model, the DTM server object maintains the functionality of the device. The FDT channel objects maintain information about the channels or the I/O data of the device, respectively. Furthermore, the FDT channel objects provide the mechanism for data exchange between DTM and Frame Application.

Within each channel the DTM can define one or more fieldbus specific parameters that give information about the channel like data types, ranges, alarms etc.



*Figure 2-2 - Channel/Parameter Relationship*

These fieldbus specific parameters contain all information a Frame Application needs for the integration of the I/O data of a device. In general, channels carrying the parameters to describe the I/O data are not the data sources - they are just representations for them. These parameters should be thought of as simply specifying the address of the data, not as the actual source of the data that the address references.

Furthermore a channel can carry the functionality for communicating via the channel with a secondary communication protocol. Such a channel is called 'Gateway Channel' and will be described in detail at the chapter about 'Nested Communication'.

Detail information about the FDT object model can be found in chapter 4.2.

## **2.2 Where FDT Fits**

Although FDT is primarily designed to control the functionality of a device and for accessing data to configure parts of the control system, FDT interfaces can be used in many places within an application. At the lowest level they can get raw data from the devices into a SCADA or DCS to configure the bus master. At a higher level the Frame Application can start a device specific diagnosis application via the DTM. The architecture and design makes it possible to build and to integrate scalable DTMs, where the functionality depends on the capabilities of the device.

The scalability of DTMs will be explained later on.



## 2.3 General FDT Architecture and Components

FDT is a specification of interfaces to facilitate the interaction between a device-specific application and a FDT Frame Application. This is shown below.

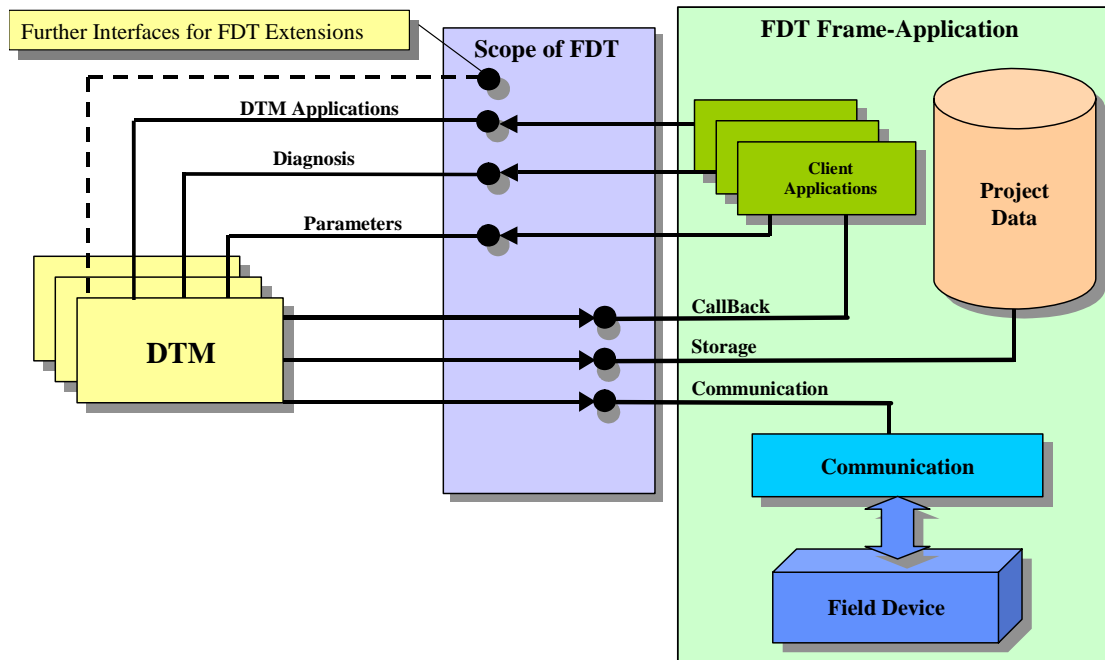


Figure 2-3 - The FDT Interfaces

As a runtime environment, a DTM needs a so-called FDT Frame Application. This Frame Application must provide the interfaces as defined in the FDT specification. Typically, the FDT-Frame Application comprises client applications that use DTMs, some kind of database for persistent storage of device/DTM data, and a communication link to the field devices. FDT-Frame Applications may be represented by applications like engineering tools for control systems (probably the complete control system) or standalone tools for device configuration. These applications are called 'Frame Applications'. Throughout this document the terms 'FDT-Container' and 'Frame Application' are used as synonyms.

Client applications are seen to be single applications focusing on special aspects like configuration, observation, channel assignment etc. and using the functionality provided by the DTM who is the server.

**The FDT Specification specifies COM interfaces (what the interfaces are), not the implementation (not the how of the implementation) of those interfaces.** It specifies the behavior that the interfaces are expected to provide to client applications that use them. The FDT-Specification neither specifies the implementation of DTMs nor the implementation of Frame Application.

Included are descriptions of architectures and interfaces which seemed most appropriate for those architectures. Like all COM implementations, the architecture of FDT is a client-server model where DTMs are the Server components managed by the Frame Application.

## 2.4 Overview of Objects and Interfaces

### 2.4.1 The Device Type Manager (DTM)

There are different types of fieldbus devices installed on a plant. Therefore, you need one or several DeviceTypeManagers (DTMs) to handle these different devices. Fieldbus device manufacturers deliver the DeviceTypeManagers. They are installed in the system, so that the system can be dynamically extended by installing new DTMs for new fieldbus devices.

It depends on the software design of the DTMs, whether they are responsible for one device type or a group of device types. It is possible to implement even one very powerful DTM for a group of targeted device types.

Usually, one DTM handles one device type and knows everything about its specific parameters, behavior, and limitations.

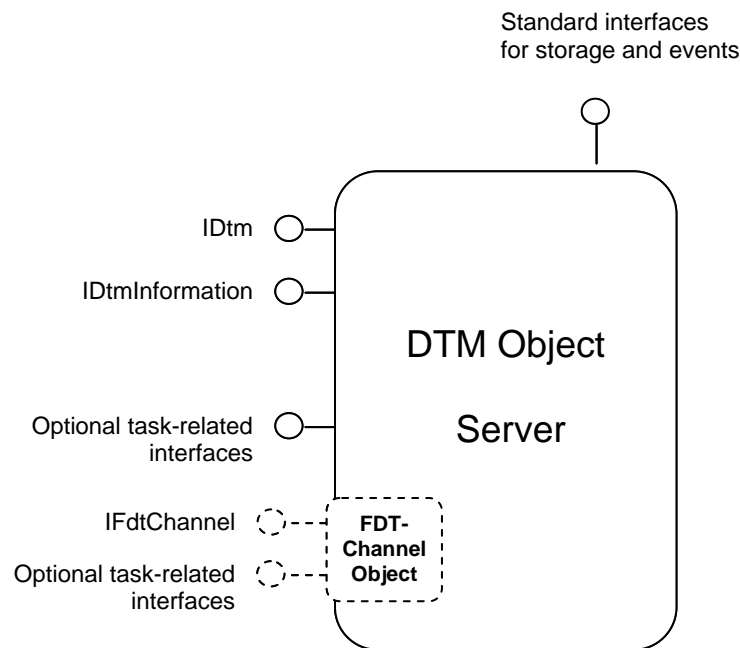


Figure 2-4 - The DTM Interfaces

The interfaces IDtm and IDtmInformation have to be implemented by each DTM. These interfaces provide the base functions and information for controlling a DTM. From a Frame Application's point of view, all task-related interfaces for the interaction with the device functionality are available via these interfaces. Which task related interfaces are provided depends on the capability of the DTM and the according device. Each interface will be described in detail at its own chapter.

In order to represent the device's I/O connections or process values accessible for data exchange with the Frame Application, the DTM can implement an FDT-Channel object (Process Channel). This object implements at least the IFdtChannel interface that gives access to all parameters of the channel which describes the channel itself.

If the device provides communication functionality, like a fieldbus adapter or a gateway, the FDT-Channel object must implement further interfaces for the communication via this channel (Communication Channel). Each interface of the FDT-Channel object will be described in detail at its own chapter.

## 2.4.2 The Block Type Manager (BTM)

The BTMs are software objects, which can be used to represent the modularity inside the device. The BTM acts in the similar manner as a DTM: e.g., it follows the state machine and the persistent mechanism of a DTM. Furthermore the BTM implements similar interfaces to those specified for a DTM. Block specific schemas replace device related XML schemas to provide block information. For example, [BtmInformationSchema](#) replaces [DTMInformationSchema](#).

The standard FDT topology mechanism and interfaces are used to assign a BTM to a DTM. If the DTM has child BTMs, it acts as a gateway. Mechanisms of nested communication are used for communication between the BTM and the DTM.

The DTM is the root element for complete device representation. It is the starting point to collect the information from the BTMs belonging to the same device. The internal device structure is represented by this topology. See figure 2-5.

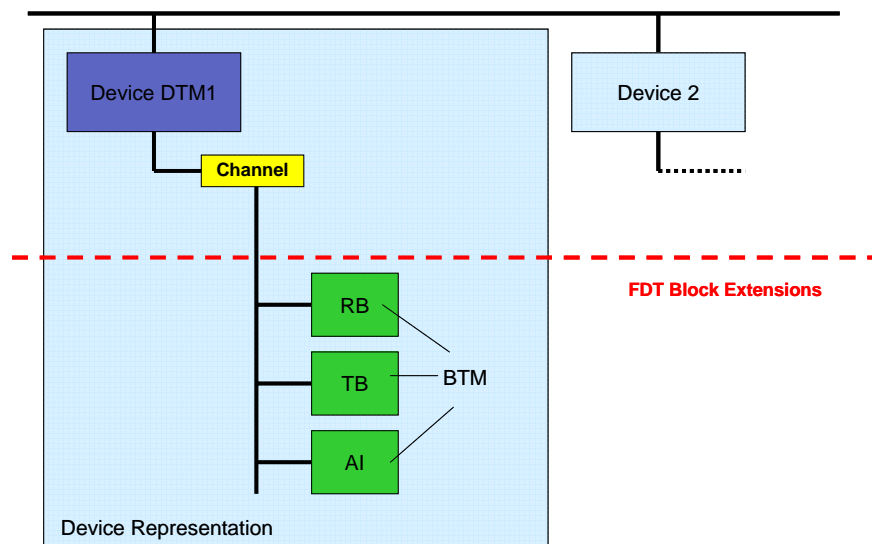


Figure 2-5 - Example of Device Architecture and Components

The BTM concept is protocol independent and can be applied to different protocols.

### 2.4.3 The FDT Frame Application

The FDT Frame Application provides the complete functionality to manage data, to communicate with the device and to embed DTMs. So it depends on the environment and the tasks whether a Frame Application is an engineering tool, a standalone tool or a web page. A DTM must be independent of the environment it is running in. So all environment-specific tasks must be handled by the Frame Application.

In most cases, engineering tools are based on Database Management Systems (RDBMS, OODBMS).

Due to this, no DTM should implement transaction strategies.

It depends on the data management of each Frame Application whether transactions are used or not. All aspects of data management are encapsulated within the Frame Application. It is not a matter of any DTM.

The standard storage interfaces encapsulate the storage mechanism of the Frame Application. This can be a simple file system or in case of engineering tool the database provided by the DCS system manufacturer.

The data a DTM stores via this interfaces are DTM-specific and are not available for other applications. It is up to the DTM which data it stores but each DTM has to guarantee that it can represent each stored device instance by loading these data

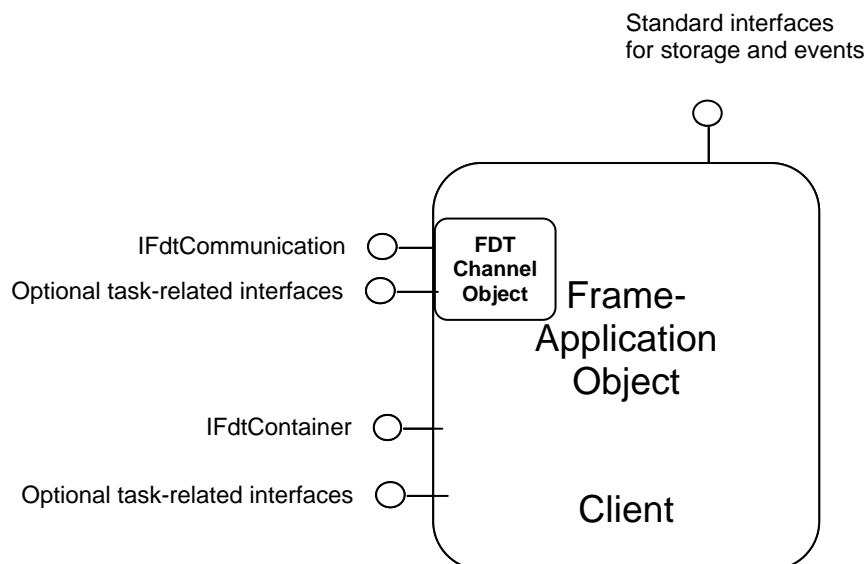


Figure 2-5 - The Frame Application Interfaces

From a DTM point of view, all task-related interfaces for the interaction with Frame Application are available via the main interface IFdtContainer of the Frame Application. Which task-related interfaces are provided depends on the capability of the Frame Application. Each interface will be described in detail at its own chapter.

In a complex plant environment, there are also complex communication networks for linking the process devices. No DTM should need any information about the topology of a system network. So it is up to the Frame Application to organize the routing for accessing a device. The Frame Application has to provide in each case a peer-to-peer connection (physical or logical). So its up to the Frame Application to manage the multi user access to a device.

To represent its communication capabilities a Frame Application can implement FDT-Channel objects. The Frame Application's channels represent the gateway from the FDT-specific to the Frame Application-specific communication. At least the interface IFdtCommunication must be implemented for a channel of

a Frame Application. On the Frame Application's side the communication channel can be a PC I/O board or engineering topology with processing units and proprietary bus systems.

IFdtCommunication always provides the communication functionality for DTMs to access their fieldbus devices. All actions that belong to the physical fieldbus have to be done by using this interface.

Each DTM can communicate with each FDT- communication channel provided that the FDT-Channel supports the appropriate communication protocol. The association of a DTM with a specific FDT-Channel is done by configuration. The topology information for configuration is stored in the Frame Application's database via the IFdtTopology interface.

A FDT- communication channel must be able to handle several connections to the same device and can be responsible for connections to different devices. The component guarantees that a link to a device is established as a peer-to-peer connection. The unique peer-to-peer connection is necessary for the asynchronous communication especially the management of the invoke IDs. Only for the peer-to-peer connection, the DTM has to guarantee that the used invoke IDs are unique for all of its communication processes (e.g. configuration and observation in parallel).

In general one and the same DTM can communicate with devices of the same type attached to different fieldbus interfaces using the appropriate FDT- communication channels.

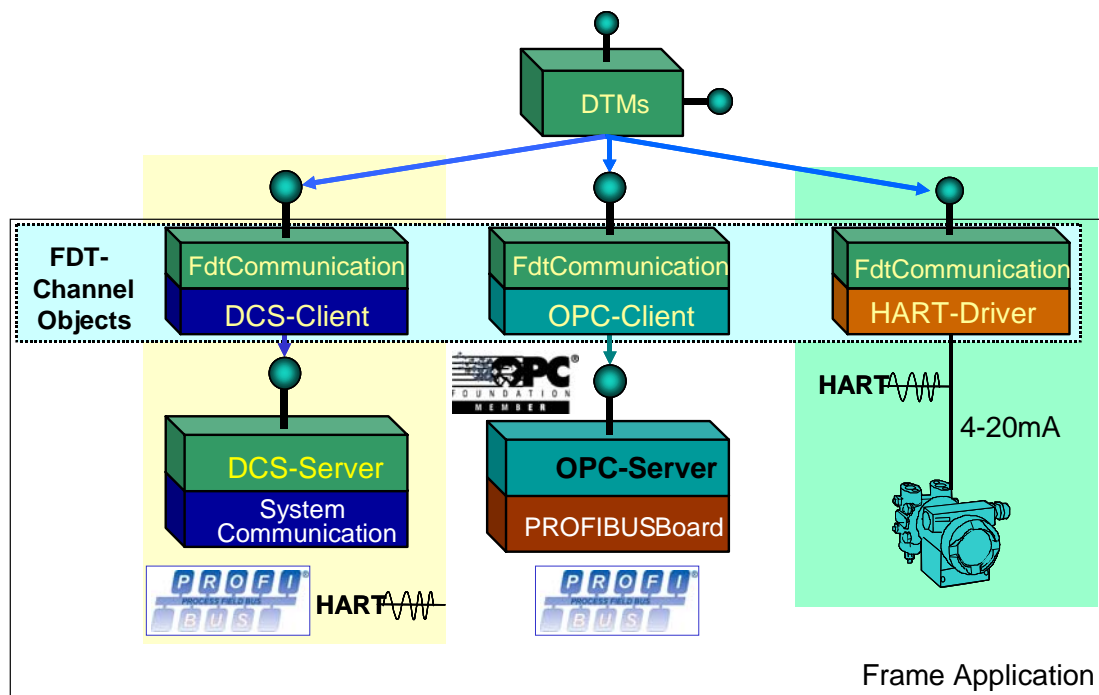


Figure 2-6 - The FDT Communication Layer

The general functionality of the FDT-Channels is encapsulated within a channel object and allows communicating with master and slave devices or following software components. An engineering tool for example may configure its fieldbus master by its own manufacturer specific means.

## 2.5 Synchronization and Serialization Issues

On one hand we mean by 'synchronization' the ability of a client to read or write values and attributes in a single transaction. For example, most applications want to ensure that value, unit, and limits of a particular measuring channel are in 'sync'. This is ensured in FDT via XML documents for data transfer. All elements of a document are transferred within a single transaction.

In general, DTMs should try to preserve synchronization of data items and attributes that are read or written in a single operation. Synchronization of items read or written individually in separate operations is not required.

On the other hand we mean by 'synchronization' the additional handshaking between the Frame Application and the DTMs to signal such states as 'ready' and 'parameter changed'. This handshaking is especially necessary within a multi-user environment to synchronize DTMs among each other and with the Frame Application according to the current application context. Each DTM has to take some simple rules into account to assure that the Frame Application can do this synchronization. Many of these issues will be clarified in the detailed descriptions of the methods and the according sequence charts below.

## 2.6 Parameter interchange via XML

The purpose of the parameter interchange via XML is to provide a way to exchange information between Frame Application and DTMs. Typically, in process control systems, multiple client applications like observing, channel assignment, or master configuration, need information about the configuration of a device.

XML is not meant to replace proprietary formats; it is meant to provide access to data that is stored in a proprietary format. Data should be stored locally in the fashion that makes the most sense. XML provides an extendable standard to connect FDT components. The data exchange is done via XML documents. Within these documents XML tags are used to delimit pieces of data. XML leaves the interpretation of the data to the application that reads it. To get a common understanding of the exchanged data FDT uses XML schemas for validation. For the data access are standardized tools like the DOM (W3C's Document Object Model) available.

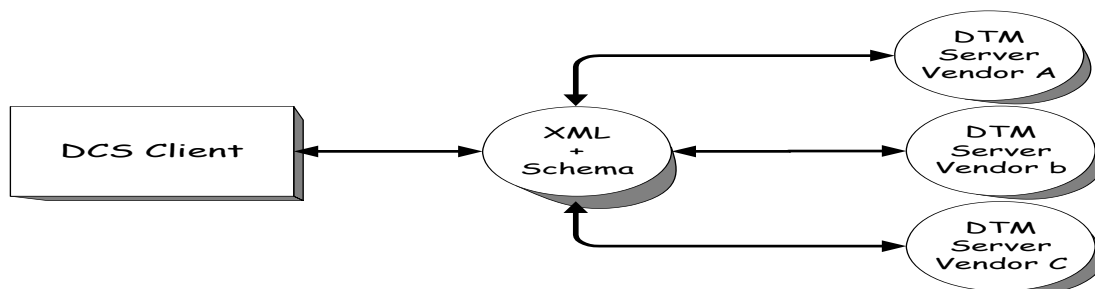


Figure 2-7 FDT Client/Server Relationship via XML

XML schemas are valid XML syntax themselves and are used to validate XML data. They allow the validation of the document structure and the data types of the elements

W3C's Document Object Model (DOM) is a standard internal representation of the document structure. It aims to make it easy for programmers to access components and delete, add, or edit their content, attributes and style. In essence, the DOM makes it possible for programmers to write applications that work properly on all browsers and servers, and on all platforms. While programmers may need to use different programming languages, they do not need to change their programming model.

An XML parser usually generates the DOM. Microsoft provides such an XML parser.. So the DOM API is free accessible in VC++, Visual Basic and VBScript.

The parsing of XML documents with XML schemas guarantees a valid DOM with well-defined elements.

The FDT developer should always work with the DOM because

- The XML Parser generates the DOM from the transferred XML data
- The schemas guarantee a valid DOM with well defined elements
- The DOM supplies standard tree- and collection-methods for data access
- The DOM can generate the XML data for the data transfer

Note:

In case information is shared across multiple clients it is required to ensure that the configuration information remains consistent across multiple clients by informing all DTMs which have a reference to the same data set about changed data. Example: more than one DTM for the same device on different working stations.

## 2.6.1 Examples of usage

Parameter interchange between DTM and Frame Application is done via XML document. Object oriented access to data is provided when using XML parser that generates an in-memory representation of the XML data (e.g. a DOM). Instance data to be stored (persistence) can also be handled as XML document to simplify the DTM development and to have a homogeneous data handling within a DTM. But also if the data are stored as XML the content of this data is only known by DTM.

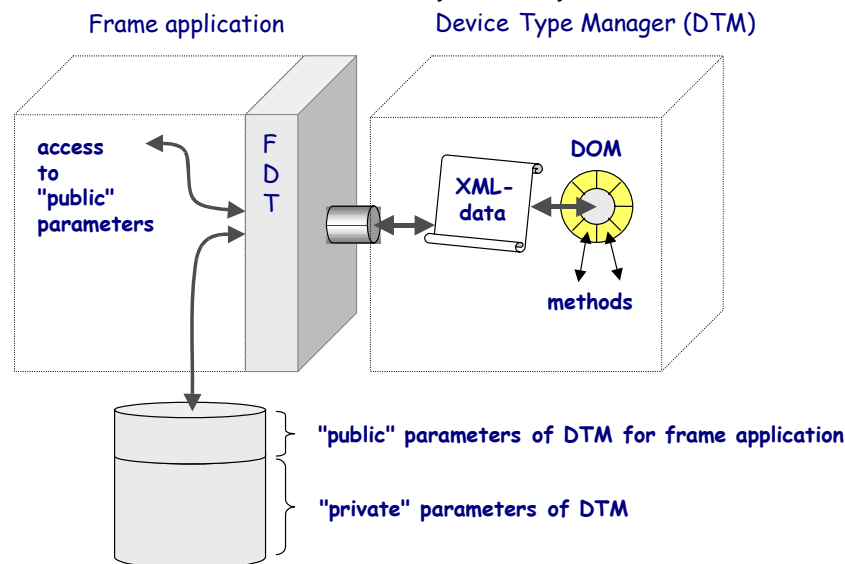


Figure 2-7-1 Data access and storage

The XML document for Communication includes device data and the necessary information for routing to establish peer-to-peer connection between DTM and field device.

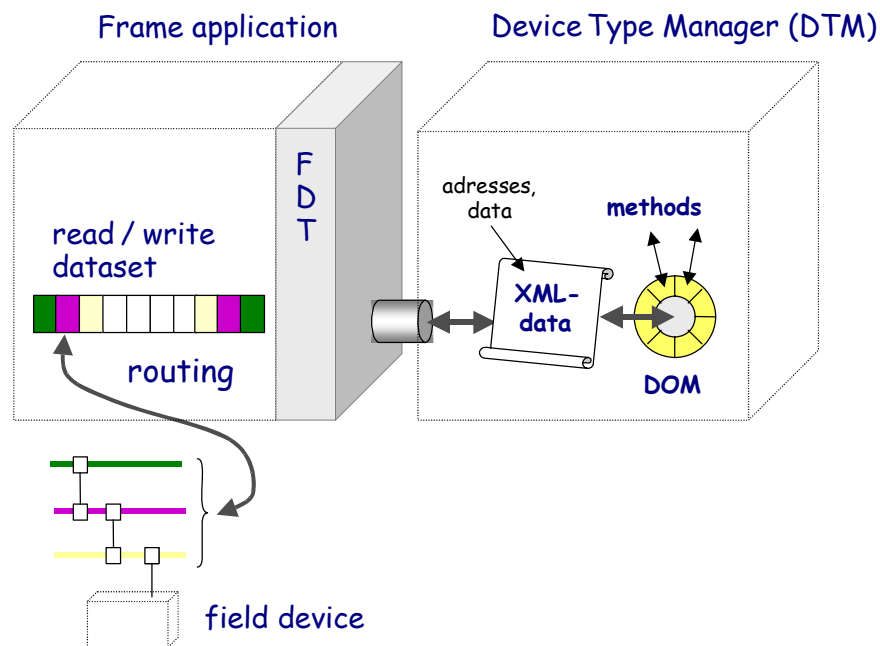


Figure 2-7-2 Communication

Here the DTM only knows the address information for a peer-to-peer connection. The Frame Application adds during runtime all necessary routing information.

For documentation of field devices within the project documentation and field device specific documentation XML is used in conjunction with XML style sheets (XSL) for layout. A default style sheet is supported by the Frame Application.

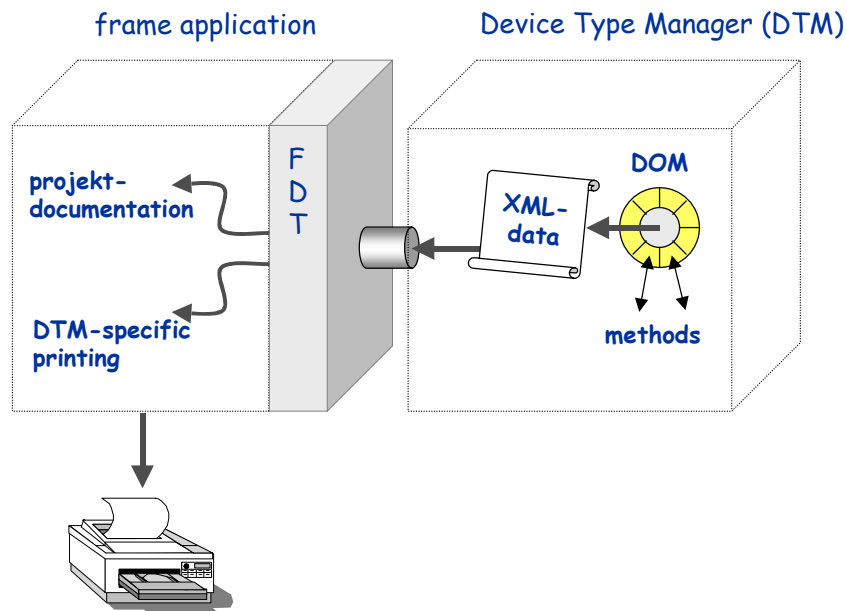


Figure 2-7-3 Documentation

In case of failsafe field devices the instance data that are stored in the controller (e.g. plc) after upload from the field device have to be verified by the DTM. Interchange format of this data is also an XML document.



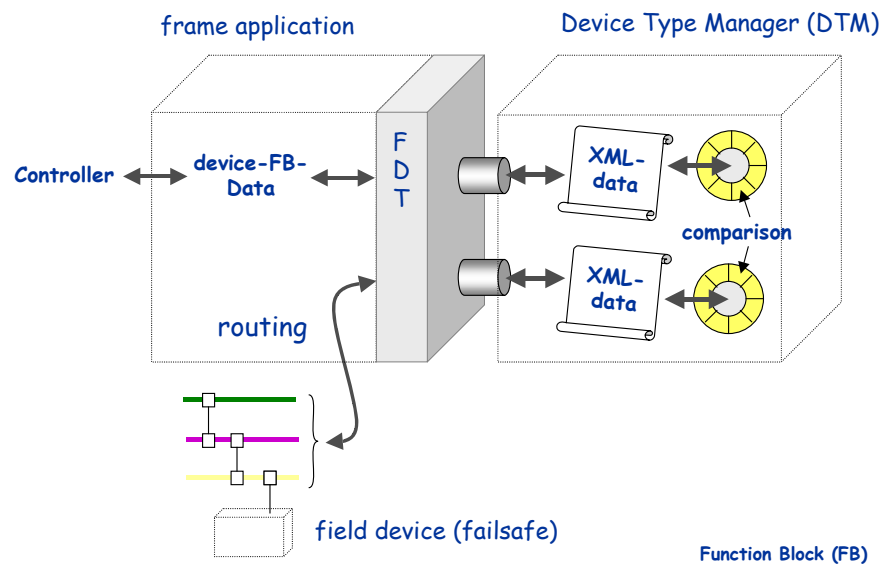


Figure 2-7-4 Parameter verification in case of failsafe devices

## 2.7 Persistent Storage Story

### 2.7.1 Persistence Overview

Basically, two kinds of data can be seen: instance-related and non-instance-related. Instance-related belong to the DTM itself, non-instance-related belong to a project or are global (e.g. libraries).

The following types of persistence requirements for a DTM can be seen:

- A typical DTM without local data storage must be able to save and restore its instance-related data within the project database of the Frame Application. Therefore it can use the standard IPersistXXX mechanism.
- A DTM using additional own, local storage must be able to store a reference to its instance-related data within the project database of the Frame Application. Such a reference allows the DTM to find the requested data in its own private storage. So the instance-related as well as the non-instance related data can be stored in a proprietary way using the file system path of the Frame Application provided by the bulk data interface. But a DTM that uses a private storage mechanism has to guarantee the data consistence for multi user and multi client data access. For import and export these DTM must provide data of its private storage via a separate interface to the Frame Application. It is not allowed for a DTM to install private data base systems.

A Frame Application must be able to store DTM's private data and all device parameters. Such a storage component, e.g. a database of an engineering tool, has to handle the locking and concurrent access to data by DTMs and Frame Application. The notification for synchronization is done via the interface IFdtContainer.

To use the storage component of the Frame Application, a DTM has to implement the standard COM-interfaces IPersistPropertyBag and IPersistStreamInit. It is not determined how the DTM performs the storage or which kind of private data of a DTM is stored.

The Frame Application requests the storage of private data of a DTM. A DTM must be able to re-establish its complete state when this is requested by the Frame Application. This is done by calling the function IPersistXXX::Load of the DTM. Reload of a DTM object by calling IPersistXXX::Load several times may

not be supported by all DTM suppliers. With an IPersistXXX:Save request a DTM must store its private data within the Frame Application. A DTM object for a new instance must be initialized if the IPersistXXX:InitNew method is called by the Frame Application.

DTMs using additional own data storage must provide all data which are necessary for commissioning via the IPersistXXX interface. Private data not provided via the IPersistXXX interface must be offered to the Frame Application for import and export via the IDtmImportExport interface. Also an IStream object is used to store and retrieve such import and export data.

Remarks:

- In order to simplify the DTM development, it is up to a DTM to implement one of the defined persistent interfaces (IPersistStreamInit or IPersistPropertyBag) according to the Microsoft standard. The Frame Application must be able to handle both.
- References to DTMs do not belong to the instance data of a DTM. A DTM must not store any references to other DTMs. A DTM can get information concerning its parents or childs via [IFdtTopology::GetParentNodes\(\)](#) and [IFdtTopology::GetChildNotes\(\)](#).
- DTM should report data load errors via standard COM error mechanism (HRESULT not equal to S\_OK). Optionally, DTM can write further human readable error information to standard COM global error info (Win32 SetLastError method).

## 2.7.2 Persistence Interfaces

For detailed information about IPersistStreamInit and IPersistPropertyBag please refer to the standard Microsoft documentation like MSDN.

## ***2.8 Basic features of a session model***

The Frame Application has to implement a session model for multi-user support, for safe data management and for data consistency.

Typically a session starts at the start of a DTM on the user interface of the Frame Application and is closed at the termination of the DTM. If there is a second DTM started by the user within the Frame Application, it creates a separate session.

All data objects that are opened inside such a DTM are registered in its session. At the end of the session all modified data of the session have to be stored synchronously (if the user wants to save modifications).

A data object is locked if a DTM opens it with write access. While data are locked by a DTM, other DTMs have only read access, but no write access.

A session can be created with or without the right for locking data within the session. If it is created without that right, DTMs are not allowed to lock data in this session. The Frame Application creates the session without that right, if the DTM is started with an function Id or a user role which does not allow modification of data. In all other cases it opens the session with the right for locking data.

## ***2.9 Basic Operation phases***

## 2.9.1 Roles and Access Rights

When a DTM is started by the Frame Application, the user role is passed to the DTM, which may restrict the parameter access according to the role.

Also the availability of functions and appearance of user interfaces may be adapted.

Examples:

An observer will not get access to calibration functions.

An observer can not modify parameters.

See chapter 5.1 (Actors)

## 2.9.2 Operation Phases

If a Frame Application requests available functions from the DTM (GetFunctions()), it passes the operation phase, which can be used by a DTM to adapt the availability of functions and the appearance of the user interface.

There are five operation phases:

- **Engineering**  
Planning and configuring of a plant,  
no online access to the plant
- **Commissioning, workshop**  
Installing the plant and the devices,  
Scanning the network to verify a planned network,  
Downloading project data into the plant,  
Programming, diagnosis of the devices,  
Adjusting parameters
- **Runtime**  
The plant is completely commissioned and running.  
Very restricted access to configuration and parameterization data.  
Scan to verify the network  
Replacing defect devices.  
Reading process values and diagnosis information.
- **Service**  
This operation phase is used for service tool Frame Applications.  
Scan to create a topology. Scan to verify a network. All service related operations. Unrestricted access to the device.
- **Not supported**  
The application doesn't support the operation phases defined above.  
A DTM must offer all functions if the Frame Application passes "not supported".

The following table describes the usage of operation phases in different Frame Application types.

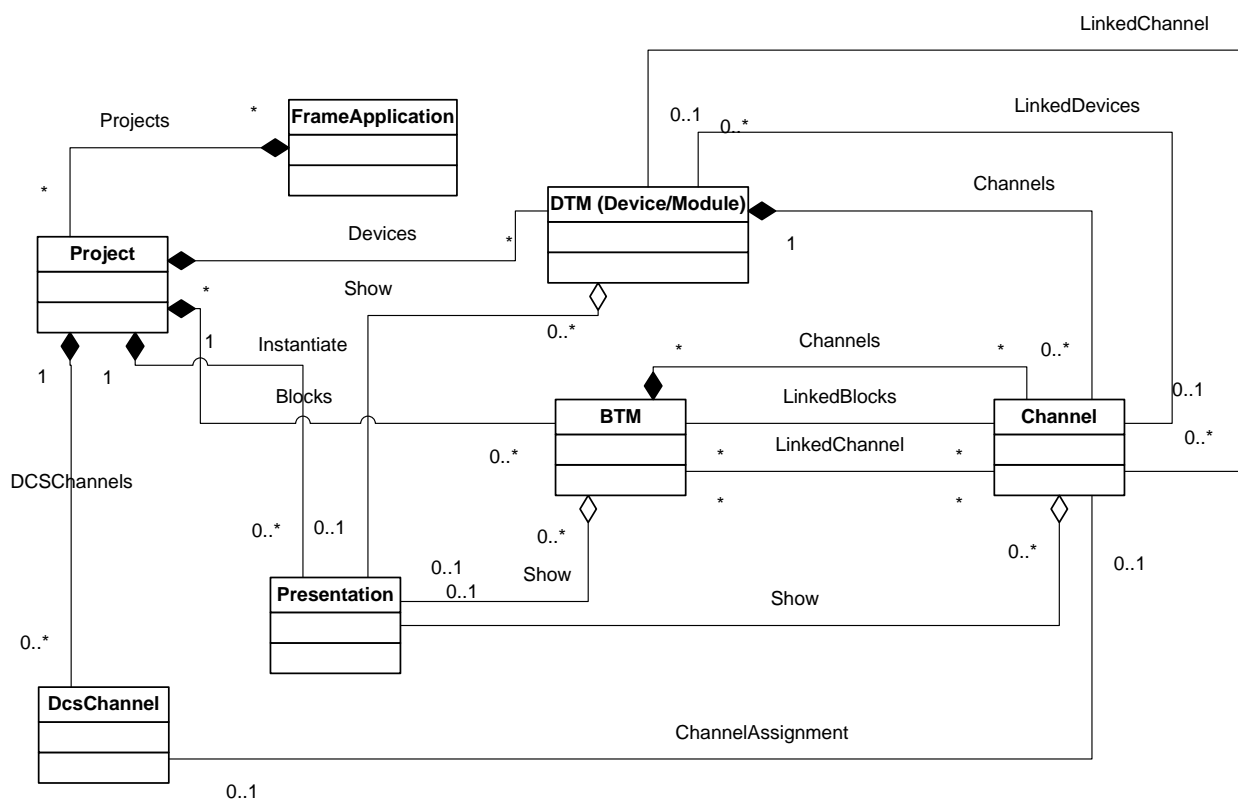
System Frame Application	Service tool Frame Application	Other Frame Applications
Engineering	Service	notSupported
Commissioning		
Runtime		

**Table 1.** Operation phases

For example the DTM allows the maintenance actor during commissioning phase the complete online parameterization, but during runtime phase it doesn't allow it or it allows it only for some of its parameters

## 2.10 Abstract FDT Object Model

This is the view on the device itself to integrate the I/O structure for Frame Application tasks like channel assignment and controller configuration. An object model of DTMs, BTMs and FDTChannels serves the access to this information. Only DTM, BTM, Channel and presentation objects are implemented as COM objects. Information about the Frame Application, project or DCS channels is provided using XML documents. Also information about functionality and properties of the DTM, BTM and channel objects is available via XML documents and exchanged via COM interfaces of these objects.

*Figure 4-1 – FDT Objects-Device related*

Object	Description
FrameApplication	<p>The FrameApplication is a logical object to represent an environment like an engineering system or a standalone tool. It controls the lifetime of DTM-instances within the project. A Frame Application can handle several projects</p>
Project	<p>The Project is a logical object to describe the management and controls at least the lifetime of device-instances within a Frame Application. That means at least the management of instance data sets within a database or file system. The Project belongs to the Frame Application.</p>
DTM	<p>Each device is represented by a DTM object. This object is the starting point for navigation to the finer parts of the device like modules and channels.</p> <p>A device can be subdivided into modules and sub modules. A module is either a hardware module or a software module. Hardware modules are plugged into physical slots of the device. For example, an I/O module can be plugged into a Remote I/O device.</p> <p>A module can also be a software module. Software modules represent static or configurable substructures of a device like a closed loop module. These SW modules in general are device specific and can not be moved between devices.</p> <p>A DTM may represent different types of devices, e.g. field devices, communication devices and gateway devices.</p> <ul style="list-style-type: none"> <li>• A Device-DTM represents a 'normal' field device that uses a Communication-Channel to communicate with the related field device</li> <li>• A Communication-DTM represents a communication device that provides communication capabilities via Communication-Channels (in sense of FDT) but does not needs any communication capabilities of a parent DTM</li> <li>• A Gateway-DTM is a Communication-DTM that provides communication capabilities via Communication-Channels (in sense of FDT) and requires communication capabilities of a parent DTM</li> </ul>
BTM	<p>A BTM is used to represent flexible software objects, like function blocks. These software blocks are more flexible than Software modules, for instance they may be moved between devices. Also a protocol may require modeling of device structures as blocks.</p>
FdtChannel	<p>FdtChannel-Object could behave in 2 ways: As a 'Communication-Channel' and a 'Process-Channel'.</p> <ul style="list-style-type: none"> <li>• Communication-Channel is an object that provides access to communication infrastructure. It is used by Device-DTM or</li> </ul>

	<p>Gateway-DTM as a service provider for communication.</p> <ul style="list-style-type: none"> <li>• Process-Channel represents a single process value, its optional state information, and its assigned ranges and alarms. A channel either belongs to a device or a module DTM.</li> </ul> <p>An FdtChannel-Object could implement both behaviors.</p>
Presentation	<p>Presentation objects represent a (visual) user interface. A presentation object can be an ActiveX control provides by a DTM or it can be encapsulated in case of monolithic DTMs The Presentation object belongs to the DTM</p>
DcsChannel	<p>The DcsChannel is a logical object representing a part of a DCS function. To make the cyclic I/O data available within a Frame Application there must be an association between the I/O function blocks and the process values of a device. The inputs and outputs of I/O function blocks are represented by DcsChannels, the process value by an FdtChannel and the association is called channel assignment.</p> <p>A DcsChannel belongs to the Frame Application.</p>

All the associations shown in the object model above are listed in the table below.

Name	Description
Devices	<p>Within a project it is important to know all field devices. This association enumerates all field devices in the project.</p> <p>It also forces DTMs with private instance database to collaborate with the project for creation and removal of field devices.</p> <p>The removal of the project forces all DTM instances to be removed.</p> <p>A DTM instance (field device) cannot be part of more than one project.</p>
Channels	<p>A channel is part of a DTM. The creation of channel instances is controlled by a DTM on the next higher level. This can be a DTM for a device or a module</p>
Show	<p>The instances of presentation objects are associated to the instance of their DTM business object by this relationship.</p> <p>This association belongs to the type of application.</p>
Instantiate	<p>This association shows for ActiveX controls that the presentation objects of a DTM are at least instantiated and embedded by the Frame Application.</p>
Channel-Assignment	<p>To make the cyclic I/O data available within a Frame Application there must be an association between the I/O function blocks and the process values of a device. The I/O function blocks are represented by a DcsChannel, the process value by an FdtChannel and the association is called channel assignment</p> <p>The Frame Application (project) is responsible for handling this association.</p>
LinkedDevice	<p>The topology of field devices is shown with this association. Within the topology tree a channel object is the parent node for a device. The association shows the connection from a channel to a device. Linked devices are available via <a href="#">GetChildNodes()</a></p> <p>The Frame Application (project) is responsible for handling this association.</p>
LinkedChannel	<p>The topology of field devices is shown with this association. Within the topology tree a DTM object as proxy for a device is the child node of a channel. The association shows the connection from a device to a channel. The channel is available via <a href="#">GetParentNodes()</a></p> <p>The Frame Application (project) is responsible for handling this association.</p>

## ***2.11 Fieldbus independent Integration***

The fieldbus independent integration of DTMs into a Frame Application is realized by bus category ids defined within the protocol specific annex. These bus category ids are defined as UUIDs.

For validation of topology during system planning runtime information of a DTM must be used. The same category ids are used as parameters of interface methods and within the XML documents used for the interaction between DTMs and Frame Application. The bus information is specified within the XML schemas like [DTMParameterSchema](#) or [FDTHARTChannelParameterSchema](#) and is available via the according bus independent interfaces. The information about the supported fieldbusses can be used to validate the bus topology or a connection during communication.

For further fieldbusses the definition of category ids can be extended complement on another fieldbus specific XML schema. The fieldbus independent FDT schemas handle the category ids just as an identifier within an attribute and each FDT component can use this information and the parameter of interface methods for validation.

Due to this mechanism the Frame Application must only know the category ids of its direct connected busses. For lower sub-topologies it just needs the contents of the category attribute within the XML documents for validation. Should the occasion arise, it is up to the DTM developer to define new category ids for proprietary bus systems and to install his DTM with this category id according to the FDT installation requirements.

## ***2.12 Scanning and DTM Assignment***

A field bus topology can be scanned to get a life list containing a list of connected physical devices.

Based on the online identification information of each found physical device, a Frame Application is able to identify proper DTMDeviceTypes, which can be instantiated in the network topology.

This assignment can be done by the Frame Application without knowledge of the field bus specific identification.

Refer to chapter 4.10.8 for more information about the general concept.



## 3 FDT Version Interoperability Guide

### 3.1 Abstract

FDT is a component based standard, which is constantly enhanced to new improved versions. Control system environments typically run for 10-15 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 chapter *FDT Version Interoperability Guide* goes further into this question and provides solutions.

### 3.2 Introduction

This chapter mainly deals with two topics concerning FDT version interoperability:

#### 1. *Persistence*

New versions of FDT components (Frame Applications and DTMs) must be able to load project data of older versions.

#### 2. *Component Interoperability*

Components of different FDT versions must interoperate properly. DTMs of newer FDT versions must run in older Frame Applications and vice versa. Communication must 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 results 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. schemas 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 release and a build number (e.g. 1.2.0.3 where 1 is the major, 2 is the minor number, 0 the release and 3 the build number). Compatibility is defined slightly different with respect to the major number:

- Compatibility between FDT components with the same FDT major version number is assured by the specification.

- Compatibility between FDT components with different FDT minor, release and build version numbers can be achieved by extra code paths. FDT will provide needed information and mechanisms to support full compatibility at this level. The minor or release number is increased if new functionality, dependent of it's appropriate importance, is added to the FDT specification. The build number is increased if a bugfix within the specification or the type library is necessary.

### 3.3 Component Interoperability

Interoperability of Frame Application and DTM components is shown in the following table:

	Frame Application, FDT Version 1.2	Frame Application, FDT Version 1.2.1	Frame Application, FDT Version ...	Frame Application, FDT Version 1.9	Frame Application, FDT Version 2.0
DTM, FDT Version 1.2	✓	✓	✓	✓	O
DTM, FDT Version 1.2.1	✓	✓	✓	✓	O
DTM, FDT Version ...	✓	✓	✓	✓	O
DTM, FDT Version 1.9	✓	✓	✓	✓	O
DTM, FDT Version 2.0	O	O	O	O	✓

**Table 1.** Interoperability between components of different versions (O – optional      ✓ – assured)

The FDT specification ensures that DTMs and Frame Applications of the same major version cooperate regardless of minor version, e.g. 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 version (than the Frame Application) may not function within a Frame Application with lower major 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. In order to work with the 'old' DTM the Frame Application needs to provide both interface sets (see section 'Design Rules'). In this situation the Frame Application behaves like an 'old' FDT version toward the DTM (old version number, old XML schemas).

To assure that containers detect only compatible installed DTMs for each FDT major version a separate Category ID will be defined within the FDT specification. This assures that a Frame Application is able to detect installed DTMs for a specific FDT major versions. DTMs supporting FDT version 1.2.1 or higher version must implement the IDtm2 interface. A Frame Application supporting FDT version 1.2.1 or higher versions must check after instantiation of a DTM instance if this DTM supports the IDtm2 interface. In this case the Frame Application must use this interface instead of using IDtm.

Version information is provided by the DTM within its information document, the version of the Frame Application is set during `IDtm2::Environment2()`. A DTM supporting FDT version 1.2.1 or higher version called using `IDtm::Environment` must assume a Frame Application supporting FDT version 1.2.

The Frame Application will always provide the schemas according to the FDT version it publishes in `IDtm2::Environment2()`. For example even DTMs based on version 1.2 will find newer schemas in the schema path if the Frame Application supports a newer version of FDT.

### **3.4 FDT Type Library**

COM type libraries can only use a two-part version number. If only the minor version number increases, all the features of the previous type library must be supported in a compatible way. If the major version number changes, code that compiled against the type library must be recompiled. The version number of the type library may differ from the version number of the application.

The four-part FDT version number is mapped to the two-part type library version using a COM minor version consisting of 5 digits. The first digit represents the FDT minor version, the second and third the FDT release version and the fourth and fifth digit represent the FDT build number (e.g., FDT 1.2.0.1 is represented as type library version 1.20001, FDT version 1.2.1.0 as type library version 1.20100).

According to COM rules a new FDT COM type library will be downward compatible to a previous version of the same major FDT version.

The FDT version (major, minor, release and build) is used also as version number of the `fdt100.dll` file. This assures that a new version replaces an older version during installation.

### **3.5 DTM and Device Versions**

When providing a new DTM for a new major FDT version, it is highly recommended to use a new `ClassID` and `ProgID`. A DTM handling a new major FDT version requires that the container has installed an appropriate new COM type library, typically incompatible to the previous major FDT version. Only this new `ClassID` and `ProgID` should register for the category ID according to the appropriate FDT major version.

If `ClassID` and `ProgID` of a DTM have changed, FDT will provide a mechanism to upgrade to the newer DTM (see also chapter 3.6 and 6.25).

Within its device catalog information (as `DTMDeviceType` elements in the `IDtmInformation` document) a new DTM version is able to provide the same list or a subset of the devices of the old DTM version. In this case a FDT Frame Application will handle the two versions of a DTM as any other DTMs able to handle the same field device as two distinguished DTMs.

A DTM of a higher FDT version which does not use a new `ClassID` and `ProgID` must support at least all `DTMDeviceTypes` of that were supported by previous versions of this DTM.

### **3.6 Persistence**

If the version of the Frame Application or the version of the DTM change, persistence (loading of stored projects) can be a problem.

A Frame Application must be able to load old datasets and to provide the unchanged dataset to the DTM even the DTM version has changed. A DTM must be able to load old datasets as long as the `ClassID` and the `ProgID` of the DTM do not change. If `ClassID` and `ProgID` of a DTM have changed, FDT will provide a mechanism to upgrade to the newer DTM. By this way it is possible to load an existing old dataset of a DTM into another DTM object. The new DTM is responsible to convert the dataset accordingly.

The new DTM provides information (via the [IDtmInformation::GetInformation\(\)](#) method) what DTM datasets it can load (compatibility of dataset). It must support all device types of the old DTM. If the Frame Application recognizes the new DTM, it can inform the user (e.g. "A new compatible DTM was installed, do you want to use the new version instead of the old version?"). If the user confirms the new DTM object is instantiated instead of the old and it loads and converts the old dataset.

## **3.7 Nested Communication**

Since DTMs can be chained with respect to the FDT topology, they need to communicate properly. Because the involved DTMs may support different FDT versions the following restrictions must be considered.

### **3.7.1 Data Exchange**

An XML document exchanged via IFdtCommunication and IFdtCommunicationEvents can contain version dependent attributes and elements. Unlike attributes of a newer version, which can be defined and handled as optional, new elements in a higher version would not be recognized by the parent component of an older version and can produce unpredictable behaviour in such a component.

Therefore a DTM can only use communication documents according to the FDT version of its parent component. This version information of the parent component must be provided within the [IFdtCommunicationEvents2::OnConnectResponse2\(\)](#). A DTM can then use only communication documents compatible to this FDT version.

### **3.7.2 Communication Channel Upgrade**

A Communication Channel must support the older FDT minor versions if it is upgraded to a higher minor version. This is due to the fact that some of its already existing children may not support the higher minor version interfaces.

### **3.7.3 Scenarios**

#### **3.7.3.1 Parent Component Version 1.2**

A DTM of FDT version higher than 1.2 will not get version information from the communication channel because [IFdtCommunicationEvents::OnConnectResponse\(\)](#) is called. In this case such a DTM can only use FDT1.2 compatible communication documents.

#### **3.7.3.2 Parent Component newer than Version 1.2**

A DTM of FDT version higher than 1.2 will get version information from the communication channel because [IFdtCommunicationEvents2::OnConnectResponse2\(\)](#) is called by the parent component. Such a DTM must use communication documents compatible to the FDT version provided by the parent component.

#### **3.7.3.3 DTM Version 1.2**

The DTM does not expose the interface IFdtCommunicationEvents2. A parent component higher than 1.2 must call [IFdtCommunicationEvents::OnConnectResponse\(\)](#).

### 3.7.3.4 Nested Communication

Within nested communication the version number returned by a Gateway-DTM depends on the functionality of its parent component.

If the Gateway-DTM is at a higher version number than the parent component then the Gateway-DTM :

- can return the same version as the parent and provide the functionality according to this version or
- can emulate the higher functionality if possible. Then the Gateway-DTM must guarantee that all required functionality is provided by its parent.

Examples:

1. A DTM of a Profibus – HART Remote I/O can support HART FDT communication compatible to FDT version 1.2.1 although its parent component supports only FDT version 1.2. In this case the Remote I/O DTM must be able to map all FDT 1.2.1 HART transaction requests (e.g. HART burst mode) to appropriate FDT 1.2 profibus compatible transaction requests
2. A Multiplexer DTM of version 1.2.1 with a parent component supporting only FDT version 1.2 may not be able to provide functionality of the higher FDT version (e.g. because it depends on device initiated data transfer).

### 3.7.4 OnAddChild

If a DTM of FDT version 1.2.1 or higher is to be connected to a 1.2 communication channel, a Frame Application has to use the first BusInformation element in the DtmParameter document of the DTM (, regarded as primary protocol), to verify if the DTM is supported by the communication channel. This is a must because communication channels of FDT version 1.2 are unable to retrieve the additional BusInformation elements.

## 3.8 Implementation Hints

### 3.8.1 Interfaces

Mandatory interfaces defined in a new FDT minor version may not be defined in older versions. Therefore components designed for the older FDT version will not implement these interfaces. Although the system runs under the newer FDT version defined by the Frame Application the components need to be aware that an interface may not be supported if the server component is based on a lower FDT version. Example: A DTM for FDT version 1.2.1 must not rely on the interface IDtm2. Although this interface may be defined mandatory for a DTM with FDT version 1.2.1, it must be possible to connect the DTM to an other DTM with FDT version 1.2 which supports only IDtm.

### 3.8.2 Persistence

A DTM should add version information to its dataset, e.g. as a property of the propertybag.

## 4 FDT Interface Specification

### ***4.1 Overview of the FDT Interfaces***

The FDT interface specification includes the following objects:

- DTM
- BTM
- Presentation Objects
  - DTMAbstractControl
  - BTMAbstractControl
  - FdtChannelAbstractControl
- FdtChannel
- Frame Application

The behavior of these objects and their interfaces are described in detail in this chapter. Developers building FDT objects for DTMs or parts of Frame Application like storage or communication objects must implement the functionality defined in this chapter.

This chapter also references and defines expected behavior of both standard COM interfaces and FDT specific interfaces that FDT compliant objects must implement.

## 4.2 FDT Objects

### 4.2.1 FDT Object Model

These FDT Objects and at least the interfaces represent the tasks for the integration of a field-device-application into a Frame Application. All interfaces of a DTM, of a BTM, of a channel as well as the interfaces of the container are implemented by one COM object so that a client can access them by calling QueryInterface on one of these interfaces of a server object. So a client is able to detect availability of optional interfaces of each object during runtime.

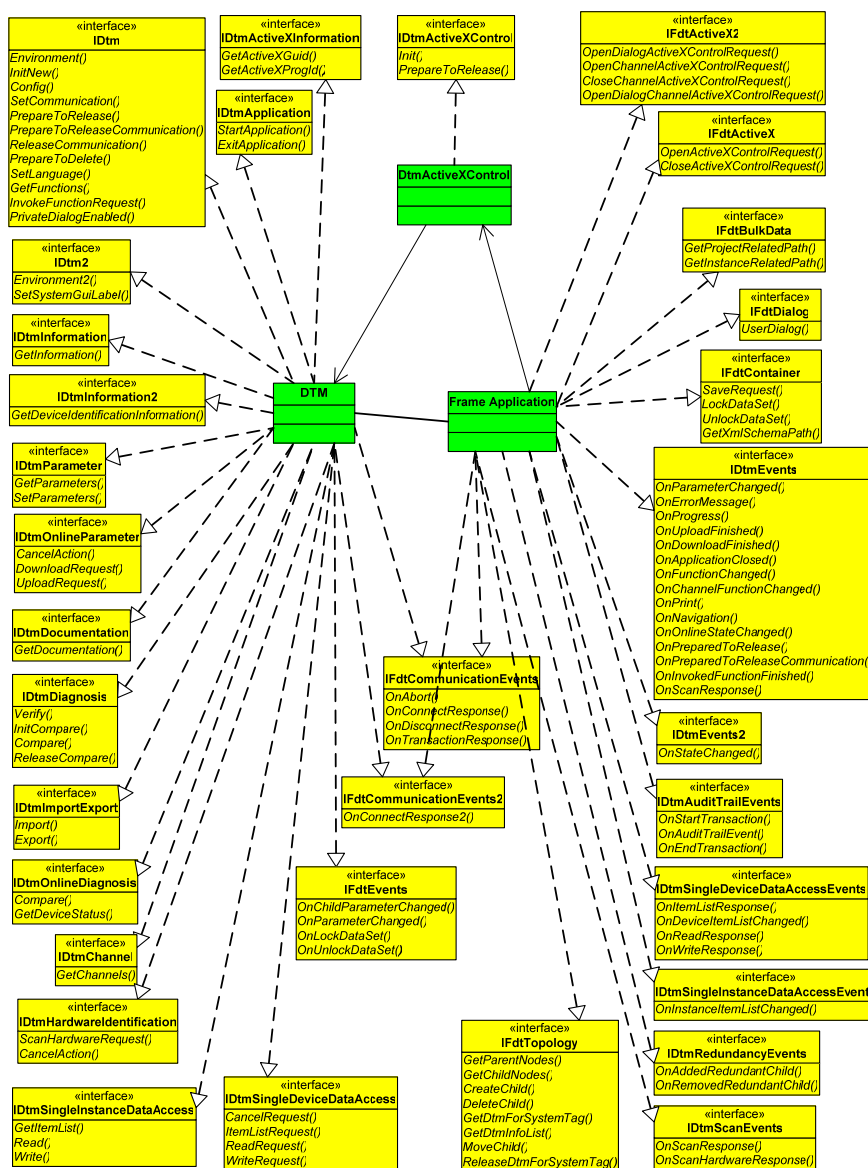


Figure 4-2 – FDT Objects- DTM, DtmActiveXControl and FdtContainer

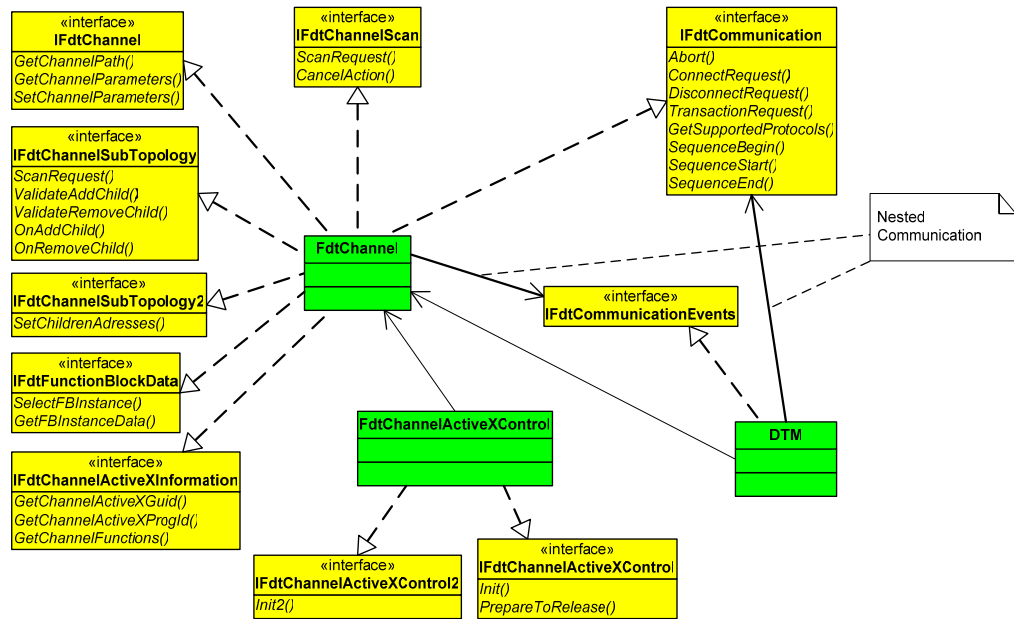


Figure 4-3 – FDT Objects- FdtChannel



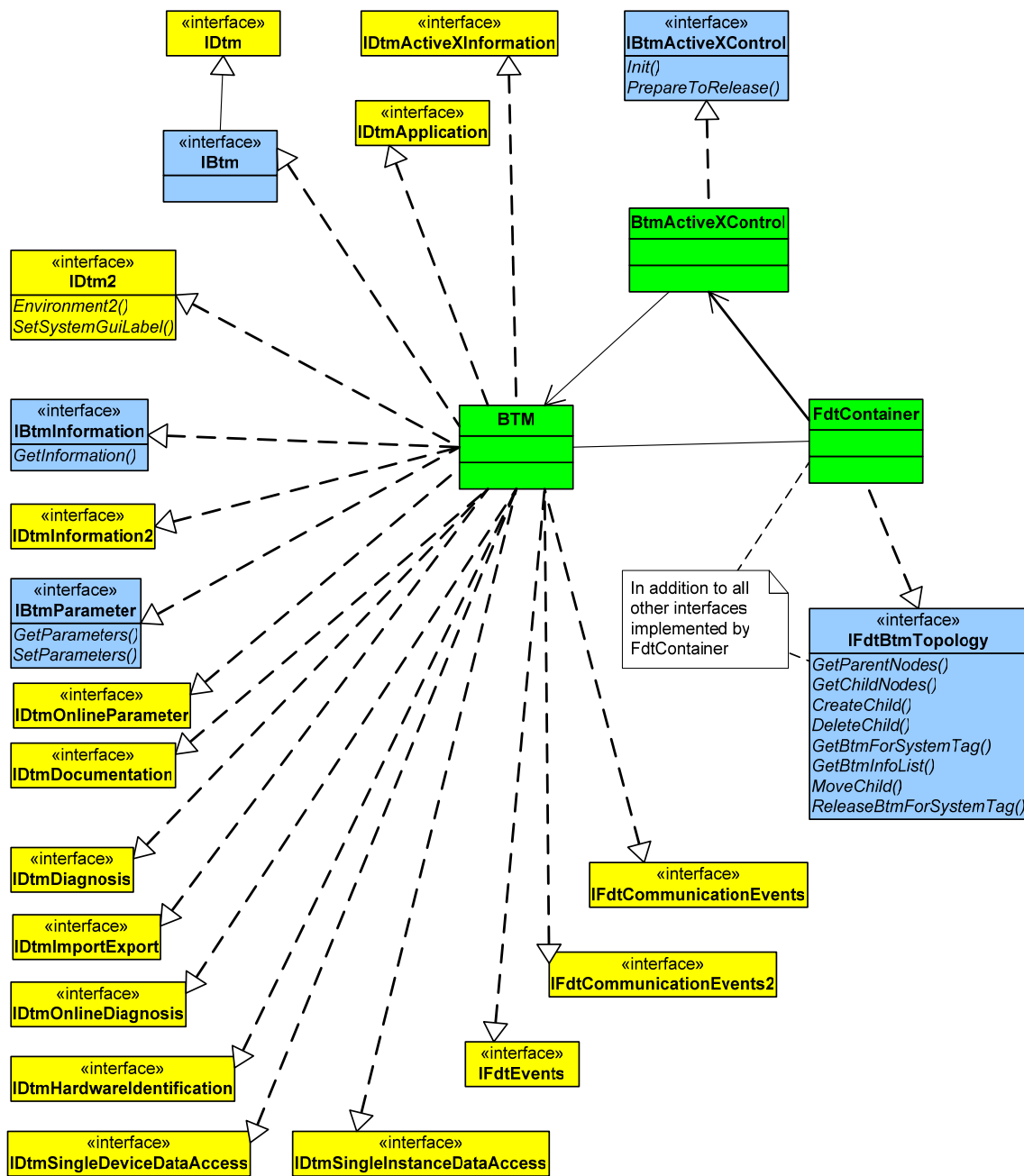


Figure 4-4 – FDT Objects- BTM and BtmActiveXControl

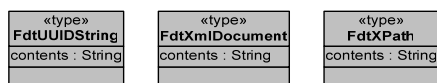
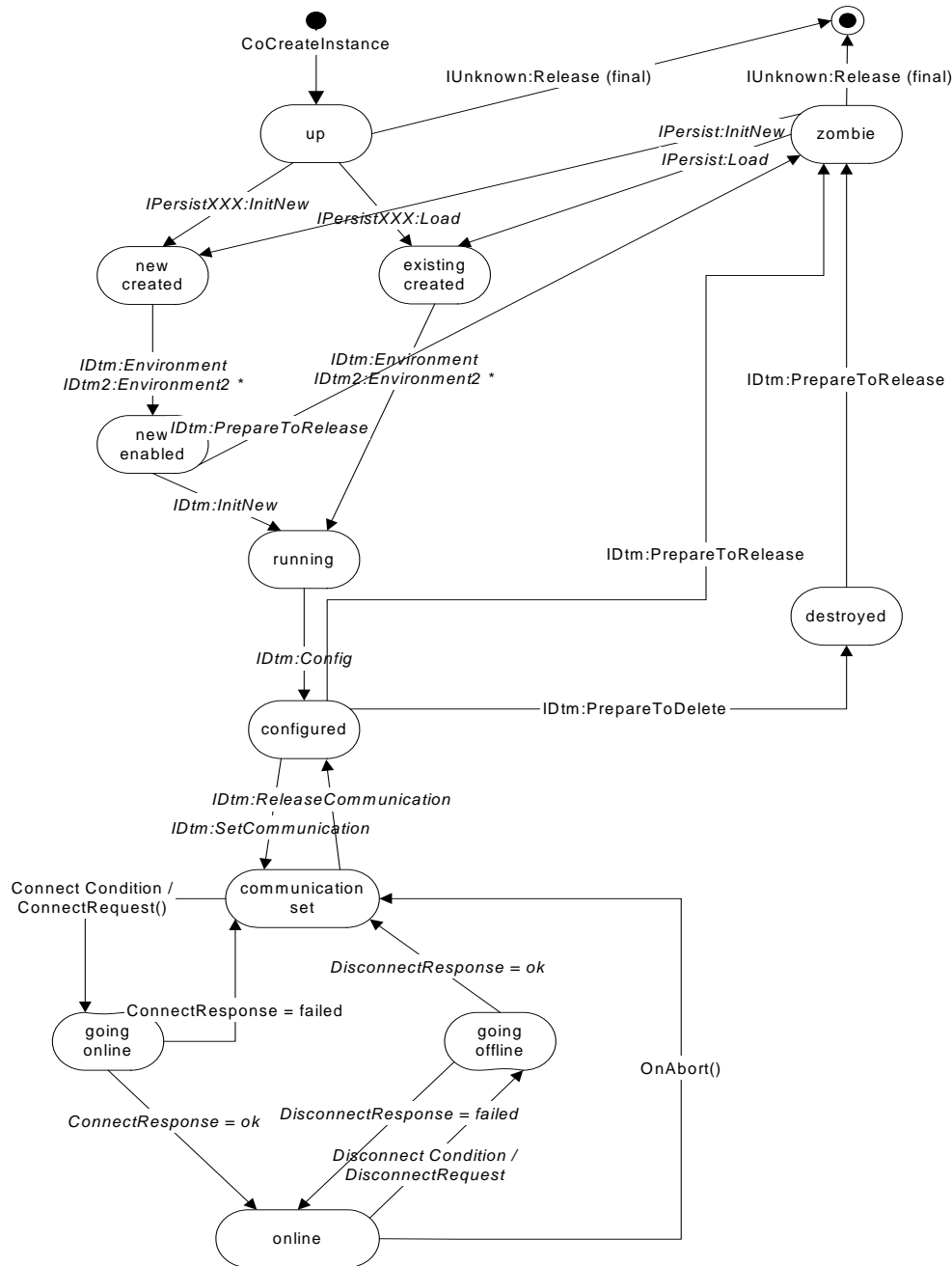


Figure 4-5 – FDT Data types

## 4.2.2 DTM State Machine

The following state machine shows the different states of a DTM.



\* dependent on FDT version of frame-application and DTM version

Figure 4 State machine of a DTM

The table below defines the interfaces of a DTM which can be used by a Frame Application at the shown states.

Interface / Method	up	new created	new enabled	existing created	running	configured	going offline	online	going online	destroyed	communication set	zombie
IPersistXXX												
InitNew	√											(√)
Load	√											(√)
Save						√	√	√	√		√	
IDtm												
Environment()		√		√								
InitNew()			√									
Config()					√							
SetCommunication()						√						
ReleaseCommunication()											√	
PrepareToDelete()						√						
PrepareToRelease()			√			√				√		
SetLanguage()			√		√	√					√	
InvokeFunctionRequest()						√	√	√	√		√	
PrepareToReleaseCommunication()							√	√	√		√	
PrivateDialogEnabled()						√	√	√	√		√	
GetFunctions()					√	√	√	√	√		√	
IDtm2												
Environment2()		√		√								
SetSystemGuiLabel()			√		√	√	√	√	√		√	
IDtmActiveXInformation					√	√	√	√	√		√	
IDtmApplication						√	√	√	√		√	
IDtmChannel					√	√	√	√	√		√	
IDtmDocumentation						√	√	√	√		√	
IDtmDiagnosis						√	√	√	√		√	
IDtmInformation			√		√	√	√	√	√		√	
IDtmInformation2			√		√	√	√	√	√		√	
IDtmImportExport						√						
IDtmOnlineDiagnosis								√	√		√	
IDtmOnlineParameter								√	√		√	
IDtmParameter						√	√	√	√		√	
IDtmHardwareIdentification							√	√	√		√	
IFdtCommunicationEvents							√	√	√		√	
IFdtCommunicationEvents2							√	√	√		√	
IFdtEvents					√	√	√	√	√		√	
IFdtChannel					√	√	√	√	√		√	
IFdtChannelActiveXInformation					√	√	√	√	√		√	
IFdtChannelSubTopology												
OnAddChild()					√	√	√	√	√		√	
OnRemoveChild()					√	√	√	√	√		√	
ScanRequest()								√	√		√	
ValidateAddChild()					√	√	√	√	√		√	

Interface / Method	up	new created	new enabled	existing created	running	configured	going offline	online	going online	destroyed	communication set	zombie
<a href="#">ValidateRemoveChild()</a>					√	√	√	√	√		√	
<a href="#">IFdtChannelSubTopology2</a>												
<a href="#">SetChildrenAddresses()</a>					√	√	√	√	√		√	
<a href="#">IFdtChannelScan</a>												
<a href="#">ScanRequest()</a>								√	√		√	
<a href="#">CancelAction()</a>								√	√		√	
<a href="#">IFdtCommunication</a>												
<a href="#">Abort()</a>							√	√	√		√	
<a href="#">ConnectRequest()</a>							√	√	√		√	
<a href="#">DisconnectRequest()</a>							√	√	√		√	
<a href="#">TransActionRequest()</a>							√	√	√		√	
<a href="#">GetSupportedProtocols()</a>						√	√	√	√		√	
<a href="#">SequenceBegin()</a>							√	√	√		√	
<a href="#">SequenceStart()</a>							√	√	√		√	
<a href="#">SequenceEnd()</a>							√	√	√		√	
<a href="#">IFdtFunctionBlockData</a>						√	√	√	√		√	
<a href="#">IDtmSingleInstanceDataAccess</a>												
<a href="#">GetItemList()</a>						√	√	√	√		√	
<a href="#">Read()</a>						√	√	√	√		√	
<a href="#">Write()</a>						√	√	√	√		√	
<a href="#">IDtmSingleDeviceDataAccess</a>												
<a href="#">CancelRequest()</a>							√	√	√		√	
<a href="#">ItemListRequest()</a>						√	√	√	√		√	
<a href="#">ReadRequest()</a>								√	√		√	
<a href="#">WriteRequest()</a>								√	√		√	

Remark: At the Zombie State not all DTMs must support reload instance via IPersistXXX interfaces, e.g. DTMs written in Visual Basic.

**Remark concerning the transition between states and methods with asynchronous behavior:**

The call of methods, which are defined with an asynchronous behavior (e.g. [PrepareToRelease\(\)](#)) will start the transition. The related end state will be reached, when the according method was called (e.g. [OnPreparedToRelease\(\)](#)).

**Remark concerning the transition between states in case of errors:**

If the method, which leads to the transition between states, fails (e.g. return value is FALSE or a COM error appears), the state is left unchanged.

The table below defines the interfaces of a Frame Application which can be used by a DTM (starting from FDT version 1.2.1) at the shown states. A Frame Application must be aware that after IDtm:Environment an older DTM may call any container interface method.

Interface / Method	up	new created	new enabled	existing created	running	configured	going offline	online	going online	destroyed	communication set	zombie
<b>IFdtContainer</b>												
GetXMLSchemaPath()			√	√	√	√	√	√	√		√	
LockDataSet()					√	√	√	√	√		√	
SaveRequest()					√	√	√	√	√		√	
UnlockDataSet()					√	√	√	√	√		√	
<b>IDtmEvents</b>												
OnApplicationClosed()					√	√	√	√	√		√	
OnDownloadFinished()								√				
OnErrorMessage()			√	√	√	√	√	√	√		√	
OnFunctionChanged()					√	√	√	√	√		√	
OnChannelFunctionChanged()					√	√	√	√	√		√	
OnInvokedFunctionFinished()					√	√	√	√	√		√	
OnNavigation()						√	√	√	√		√	
OnOnlineStateChanged()							√	√	√		√	
OnParameterChanged()					√	√	√	√	√		√	
OnPreparedToRelease()												√
OnPreparedToReleaseCommunication()							√	√	√		√	
OnPrint()					√	√	√	√	√		√	
OnProgress()						√	√	√	√		√	
OnScanResponse()								√				
OnUploadFinished()								√				
<b>IDtmAuditTrailEvents</b>						√	√	√	√		√	
<b>IFdtActiveX</b>					√	√	√	√	√		√	
<b>IFdtActiveX2</b>					√	√	√	√	√		√	
<b>IFdtBulkData</b>			√		√	√	√	√	√		√	
<b>IFdtDialog</b>					√	√	√	√	√		√	
<b>IFdtTopology</b>						√	√	√	√		√	
<b>IDtmSingleDeviceDataAccessEvents</b>												
OnItemListResponse()						√	√	√	√		√	
OnDeviceItemListChanged()						√	√	√	√		√	
OnReadResponse()								√	√		√	
OnWriteResponse()								√	√		√	
<b>IDtmSingleInstanceDataAccessEvents</b>						√	√	√	√		√	

## 4.3 Device Type Manager

### 4.3.1 Interface IDtm

This interface is the main interface of a DTM that supports FDT version 1.2 and older. Via this interface the DTM gets its initialization after the instantiation.

The Frame Application uses the interface to set information, like communication interface or language, the DTM needs during runtime as well as to reset the DTM for release.

At this time the DTM is not connected to an instance data set of a device. At this state the DTM can be asked for its static information like version, vendor, and its capabilities.

If the DTM is initialized it can be asked for its instance independent supported functions.

#### 4.3.1.1 Config

```
HRESULT Config(
    [in] FdtXmlDocument userInfo,
    [out, retval] VARIANT_BOOL* result);
```

##### Description

Is called by the Frame Application for initialization concerning the current user.

Parameters	Description
userInfo	XML document containing the current user rights and the user role specified by the <a href="#">FDTUserInformationSchema</a>

##### Return Value

Return Value	Description
TRUE	DTM accepted the given data
FALSE	The operation failed.

##### Behavior

It informs the DTM during initialization about the role and the rights of the current user.

##### Comments

In general it is expected that a DTM adapts the provided functionality according to the role of the current user. (see 4.3.1.3)

### 4.3.1.2 Environment

```
HRESULT Environment(
    [in] BSTR systemTag,
    [in] IFdtContainer* container
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Is called by the Frame Application to set the [systemTag](#) and the back pointer to the Frame Application.

Parameters	Description
<a href="#">systemTag</a>	Identifier for the device instance; set by the Frame Application
Container	Back pointer to the Frame Application

#### Return Value

Return Value	Description
TRUE	DTM has accepted the data
FALSE	The operation failed.

#### Behavior

Is called by the Frame Application to initialize a DTM for a device instance. Furthermore the Frame Application passes the pointer to its own main interface.

A DTM needs the [systemTag](#) during runtime for navigation or to identify itself at the event interface of the Frame Application. The DTM should not store the [systemTag](#) to prevent side effect if a Frame Application copies, moves, or deletes data sets.

#### Comments

The [systemTag](#) is independent of communication tags (e.g. HART device tag).

### 4.3.1.3 GetFunctions

```
HRESULT GetFunctions(
    [in] FdtXmlDocument operationState,
    [out, retval] FdtXmlDocument* result);
```

#### Description

Returns an XML document containing information about standard (defined by applicationID) or additional functionalities (defined by functionId) and documents supported by a DTM

Parameters	Description
operationState	XML document containing the current operation phase specified by the <a href="#">FDTOperationPhaseSchema</a>

### Return Value

Return Value	Description
Result	XML document containing actual supported functions specified by the <a href="#">DTMFunctionsSchema</a>

### Behavior

This method provides the access to DTM standard functionality, defined by the applicationIDs and specific functionality, which is not within the scope of FDT. This data are available as soon as the DTM is instantiated but the information may change if it is instance specific.

That means that the contents of the document can change or can at least be empty after an [OnFunctionChanged\(\)](#) event. This event is sent by a DTM if the configuration results in a changed extended functionality.

Usually this information is used by the Frame Application to create menus.

Functions with user interface are started via [IDtmApplication::StartApplication\(\)](#), [IDtmActiveXInformation::GetActiveXGuid\(\)](#) or [IDtmActiveXInformation::GetActiveXProgId\(\)](#) and work asynchronous

Function calls without user interface are asynchronous as well and are started via [InvokeFunctionRequest\(\)](#).

The asynchronous behavior is described at the according chapters.

The method additionally provides access to documents provided by a DTM, which can be displayed by the Frame Application or a special viewer program.

It is expected that a DTM adapts the provided list of functions according to the role of the current user (see chapter 4.3.1.1). As long as the DTM does not have valid user information (e.g. in state "running"), it should behave like the user with the least rights ("Observer") is using the DTM.

Functions with associated module Ids (so-called module functions) must not be shown directly in the standard context menu of the DTM node. They may appear in a submenu which is associated to a DTM module or in the context menu of a DTM module node.

In order to ensure that module functions can be called by the user, the Frame Application should provide a submenu "Modules" in the context menu of the DTM node or provide appropriate context menus for DTM modules or offer these functions by other means.

The DTM has to ensure that all accessible module functions (enabled and not hidden) are valid. If the active module collection of a DTM varies due to configuration changes, the DTM must call [OnFunctionChanged\(\)](#) to notify the Frame Application. That means the Frame Application is not responsible for matching the given module Ids to the list of the existing modules returned by [IDtmParameter::GetParameters](#).

### Comments

If a Frame Application does not support the operation phases ('notSupported') the DTM should provide all functions based on the current state (e.g. 'communication set') and the current user role.

If the DTM wants to limit the number of opened ActiveX controls, it should disable the corresponding functions. If an ActiveX is closed, the DTM has to enable the function again.



#### 4.3.1.4 InitNew

```
HRESULT InitNew(
    [in] FdtXmlDocument deviceType,
    [out, retval] VARIANT_BOOL* result);
```

##### Description

Is called by the Frame Application to initialize a newly created instance data set for a specific device type.

Parameters	Description
deviceType	XML document containing the manufacturer specific data like unique identifier for a sub device type specified by <a href="#">DTMInitSchema</a>

##### Return Value

Return Value	Description
TRUE	DTM is initialized
FALSE	The operation failed.

##### Behavior

The Frame Application initializes the DTM for a specific device-type. The supported device types of a DTM are available via [IDtmInformation::GetInformation\(\)](#). This initialization is necessary especially for a DTM that supports more than one device type.

##### Comments

-/-

#### 4.3.1.5 InvokeFunctionRequest

```
HRESULT InvokeFunctionRequest (
    [in] FdtUUIDString invokeld,
    [in] FdtXmlDocument functionCall,
    [out, retval] VARIANT_BOOL* result);
```

##### Description

Starts a function of a DTM.

Parameters	Description
invokeld	Identifier for the started function.
functionCall	XML document containing the DTM specific function id for

	the requested function or user interface specified by the <a href="#">DTMFunctionCallSchema</a>
--	---

**Return Value**

Return Value	Description
TRUE	The function started.
FALSE	The function call failed.

**Behavior**

The function call id associates a DTM with a functional/logical context. Each DTM can provide more than one function. The functions supported by a DTM can be requested via [GetFunctions\(\)](#)

Functions with user interface are started via [IDtmApplication::StartApplication\(\)](#), [IDtmActiveXInformation::GetActiveXGuid\(\)](#) or [IDtmActiveXInformation::GetActiveXProgId\(\)](#) and can work asynchronous. If the called function is finished or terminated by the user the DTM send an [OnApplicationClosed\(\)](#) event to the Frame Application.

Function calls without user interface are asynchronous as well. For this functions the DTM sends an [IDtmEvents::OnInvokedFunctionFinished\(\)](#) event if the function is finished.

**Comments**

To prevent side effects the Frame Application should not send further request to the DTM proceeding the function call.

### 4.3.1.6 PrepareToDelete

HRESULT [PrepareToDelete](#) (  
[out, retval] VARIANT\_BOOL\* result);

**Description**

Returns TRUE if the device instance data set can be deleted at the Frame Applications database. Used to inform the DTM that it has to clean up e.g. its log files or protocols. The data set will be deleted by the Frame Application.

Parameters	Description
------------	-------------

**Return Value**

Return Value	Description
TRUE	Data set can be deleted
FALSE	The operation failed.

**Behavior**

The method is used to inform a DTM that it has to clean up e.g. its log files or protocols. After this function call the data set will be deleted by the Frame Application. The Frame Application is responsible to ensure the pre-conditions for the delete. That means that the Frame Application must ensure, that all DTM instance related to this data set are shut down (either Zombie-State or released)..

If the DTM returns FALSE the Frame Application can inform the user to close the user interfaces or can terminate them by [ExitApplication\(\)](#) or [IDtmActiveXControl::PrepareToRelease\(\)](#). In general a DTM will finish its current communication process during the release of its user interfaces.

#### Comments

-/-

### 4.3.1.7 PrepareToRelease

HRESULT [PrepareToRelease](#)(  
[out, retval] VARIANT\_BOOL\* result);

#### Description

Used to inform the DTM that it has to release its links to other components. The DTM will be released by the Frame Application.

Parameters	Description
------------	-------------

#### Return Value

Return Value	Description
TRUE	The DTM will release its links to other components
FALSE	The operation failed.

#### Behavior

The DTM has to release all links to other components and has to terminate all pending or running functions. It must also close all user interfaces.

The DTM sends a notification via [IDtmEvents::OnPreparedToRelease\(\)](#) to the Frame Application if the DTM can be released.

#### Comments

It is up to a DTM to decide, if transient data should be stored or not. To store the data, [IFdtContainer::SaveRequest\(\)](#) should be called.

Remark: The DTM has to implement a proper behavior concerning chapter '5.13.1.2 Persistence' to give a Frame Application the information about the storing state of DTM related data (refer to '13.1 FDT Data Types' attribute 'storageState').

### 4.3.1.8 PrepareToReleaseCommunication

HRESULT [PrepareToReleaseCommunication](#)(  
[out, retval] VARIANT\_BOOL\* result);

### Description

Used to inform the DTM that it has to release it links to the communication components.

Parameters	Description
------------	-------------

### Return Value

Return Value	Description
TRUE	The DTM will release its references at the communication pointer
FALSE	The operation failed.

### Behavior

The DTM has to release all references to the communication pointer set during [SetCommunication\(\)](#). The method returns FALSE if a communication call is active and cannot be terminated.

The DTM sends a notification via [IDtmEvents::OnPreparedToReleaseCommunication\(\)](#) to the Frame Application if the communication pointer can be released.

### Comments

The method returns FALSE if communication call is active and cannot be terminated.

The method returns TRUE if DTM accepts shutdown of communication. DTM has to fire progress events to inform Frame Application about ongoing progress if [IDtmEvents::OnPrepareToReleaseCommunication\(\)](#) notification may take a longer time, for example if still some communication calls have not returned.

See also 4.9.1.13, OnProgress and 4.3.1.11 SetCommunication

### 4.3.1.9 PrivateDialogEnabled

HRESULT [PrivateDialogEnabled](#)(  
     [in] VARIANT\_BOOL enabled,  
     [out, retval] VARIANT\_BOOL\* result);

#### Description

Sends a notification to a DTM whether it is allowed to open a private dialog window.

Parameters	Description
enabled	TRUE means that it is allowed for a DTM to open a dialog window

#### Return Value

Return Value	Description
TRUE	The function succeeded.
FALSE	The function failed.

#### Behavior

This special state is set by a Frame Application during runtime. This may be necessary if currently a user action must not be disturbed or if it is not allowed, that a dynamically opened window gets the focus or hides other windows.

Enabled set to FALSE mean that the DTM must not open any dialog windows.

If at this state any error occurs the DTM can send a notification to the Frame Application via [IDtmEvents::OnErrorMessage\(\)](#).

If the DTM needs a user action, and therefore has to open a user dialog, it should first ask the Frame Application via the [IFdtDialog](#) interface. At this request the default response is not to open the dialog. So it is up to the Frame Application to allow opening the dialog, delay the request until the current user action is finished, or to refuse the request by sending the default response.

If a DTM uses ActiveX controls as user interfaces, the DTM has to inform its open controls whether they are allowed to open dialog windows.

#### Comments

-/-

### 4.3.1.10 ReleaseCommunication

HRESULT [ReleaseCommunication](#)(  
     [out, retval] VARIANT\_BOOL\* result);

#### Description

Used to inform the DTM that the communication pointer will be released by the Frame Application.

Parameters	Description
------------	-------------

**Return Value**

Return Value	Description
TRUE	The DTM has set at the communication pointer to NULL
FALSE	The operation failed.

**Behavior**

The DTM should set the communication pointer, set during [SetCommunication\(\)](#), to NULL. The method returns FALSE if a communication call is active.

If the DTM returns TRUE it has to assume that the communication pointer is invalid for further function calls.

In general the Frame Application has to ensure that all applications or function calls of a DTM are finished before it releases the communication pointer.

**Comments**

See also chapter 4.9.1.11, [OnPreparedToReleaseCommunication\(\)](#).

### 4.3.1.11 SetCommunication

```
HRESULT SetCommunication(
    [in] IFdtCommunication* communication,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

Set the interface pointer to the communication interface that the DTM has to use for online access.

Parameters	Description
Communication	Interface pointer of a communication channel

**Return Value**

Return Value	Description
TRUE	Pointer accepted
FALSE	Invalid communication pointer

**Behavior**

The pointer to the communication interface of a communication channel is set by the Frame Application for online calls like [DownloadRequest\(\)](#) or [UploadRequest\(\)](#).

The communication channel can check the supported communication protocol via [IFdtChannel::GetChannelParameters\(\)](#) and the attribute gatewayBusCategory. In general the Frame Application is responsible to establish a valid link between a channel and a DTM or between two communication channels. This check can be done to ensure the link or in case of communication problems.

## Comments

To ensure a proper behavior, the Frame Application should implement the following rules:

- For each DTM, which acts as a root concerning the chain of interfaces for nested communication, the method [IDtm::SetCommunication\(\)](#) must be called with a NULL pointer as parameter 'communication'. This leads to the transition from 'configured' to 'communication set' concerning the DTM state machine (refer to chapter 4.2.2, DTM State Machine)
- To set up the chain for nested communication, the Frame Application must start to hand over the interface pointer from the DTM, which acts as a root concerning the chain of interfaces for nested communication. To release the chain, according to 4.3.1.8, [IDtm::PrepareToReleaseCommunication\(\)](#), the Frame Application should act vice versa

### 4.3.1.12 SetLanguage

HRESULT [SetLanguage](#)(  
     [in] long languageId,  
     [out, retval] VARIANT\_BOOL\* result);

## Description

Returns TRUE if the requested language is supported by the DTM.

Parameters	Description
languageId	Unique identifier for user interface localization; defined by Windows as a locale identifier (LCID) containing the language identifier in the lower word and the sorting identifier as well as a reserved value in the upper word. The identifier supplied in an LCID is a standard international numeric abbreviation (e.g. German – Standard: 0x0407, English - United States: 0x0409). (See also WIN32/Platform SDK, locale id or LCID)

## Return Value

Return Value	Description
TRUE	Language supported. All human readable outputs will use the required language
FALSE	Language not supported

## Behavior

The Frame Application sets the language during initialization of the DTM. So all presentation objects of the same instance have the same language. Also the messages on the event interfaces like [OnErrorMessage\(\)](#) and the human readable contents of the XML documents like at the interface [IDtmDocumentation](#) or [IDtmInformation](#) have to use the requested language. If a DTM does not support the requested language it uses the current language in case it is already initialized or sets its default language on the first initialization.

## Comments

The supported languages of a DTM are listed within the [DTMInformationSchema](#). One DTM instance is always initialized with one language. It's up to a DTM whether it can change the current language of an

user interface while the user interface is shown. The output format for numbers, currencies, times, and dates will be based on the regional options of the operation system.

### 4.3.2 Interface IDtm2

This interface is the main interface of a DTM supporting FDT version 1.2.1 or higher versions. Via this interface such a DTM gets its initialization after the instantiation. This interface extends the interface by new methods. This interface is mandatory.

A Frame Application supporting 1.2.1 or higher version has to use IDtm2, if the DTM supports this interface. Instead of calling [IDtm::Environment\(\)](#) such a Frame Application must then use the [IDtm2::Environment2\(\)](#) method.

#### 4.3.2.1 Environment2

```
HRESULT Environment2(
    [in] BSTR systemTag,
    [in] IFdtContainer* container,
    [in] FdtXmlDocument frameInfo,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Is called by a Frame Application supporting 1.2.1 or higher to set the [systemTag](#), the back pointer to the Frame Application and an XML document providing frame version information. Such a Frame Application will not call the [IDtm::Environment\(\)](#) method.

Parameters	Description
<a href="#">systemTag</a>	Identifier for the device instance; set by the Frame Application
Container	Back pointer to the Frame Application
frameInfo	XML document containing frame version information by the <a href="#">DTMEnvironmentSchema</a>

#### Return Value

Return Value	Description
TRUE	DTM has accepted the data
FALSE	The operation failed.

#### Behavior

Is called by the Frame Application supporting 1.2.1 or higher to initialize a DTM supporting 1.2.1 or higher for a device instance. Furthermore the Frame Application passes the pointer to its own main interface and a document providing Frame Application version information.

A DTM needs the [systemTag](#) during runtime for navigation or to identify itself at the event interface of the Frame Application. The DTM should not store the [systemTag](#) to prevent side effect if a Frame Application copies, moves, or deletes data sets.



**Comments**

The [systemTag](#) is independent of communication tags (e.g. HART device tag).

**4.3.2.2 SetSystemGuiLabel**

```
HRESULT SetSystemGuiLabel (
    [in] FdtXmlDocument systemGuiLabel,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

This method is called by the Frame Application to set a unique human readable identifier of the DTM instance in the context of the Frame Application.

Parameters	Description
<a href="#">systemGuiLabel</a>	XML document containing a unique human readable identifier of the DTM instance in the context of the Frame Application. Document specified by <a href="#">DTMSystemGuiLabelSchema</a>

**Return Value**

Return Value	Description
True	DTM has accepted the data
False	The operation failed.

**Behavior**

This method is called by the Frame Application in order to set a system label e.g. for a message box or a user interface which is part of the DTM and can not be embedded within a Frame Application. The Frame Application sets a unique human readable identifier of the DTM instance in the context of the Frame Application which guarantees a unique identification between the device and e.g. a message box of a DTM. The DTM must use this system label for all kinds of windows that will be opened by the DTM themselves. May be the DTM can extend this title with specific information.

The Frame Application has to call this method as early as possible. As long as the method is not called the DTM has to use the tag of the device as human readable string by default.

**Comments**

The human readable identifier must not be stored by the DTM. It is recommended to update the labels of all open windows when [SetSystemGuiLabel\(\)](#) is called.

### 4.3.3 Interface IDtmActiveXInformation

This interface provides the user interface of a DTM as ActiveX controls for embedding within a Frame Application.

#### 4.3.3.1 GetActiveXGuid

```
HRESULT GetActiveXGuid(
    [in] FdtXmlDocument functionCall,
    [out, retval] FdtUUIDString* result);
```

##### Description

Returns the UUID for the ActiveX control according to the function call id.

Parameters	Description
functionCall	XML document containing the function id for the requested user interface specified by the <a href="#">DTMFunctionCallSchema</a>

##### Return Value

Return Value	Description
result	UUID for an ActiveX control

##### Behavior

Returns a UUID that the Frame Application can use to instantiate the control.

If a requested application is not supported the method returns a NULL string.

The kind of user interface that is expected for a DTM is described in detail within the schema provided by [IDtm::GetFunctions\(\)](#)

##### Comments

-/-

#### 4.3.3.2 GetActiveXProgId

```
HRESULT GetActiveXProgId(
    [in] FdtXmlDocument functionCall,
    [out, retval] BSTR* result);
```

##### Description

Returns the ProgId for the ActiveX control according to the function call id.

Parameters	Description
functionCall	XML document containing the function id for the requested user interface specified by the <a href="#">DTMFunctionCallSchema</a>

**Return Value**

Return Value	Description
result	ProgId for an ActiveX control

**Behavior**

Returns the ProgId for the ActiveX control according to the function call id. Frame Applications implemented with scripting languages can use this ProgId to instantiate the control.

If a requested application is not supported the method returns NULL pointer

The kind of user interface that is expected for a DTM is described in detail within the schema provided by [IDtm::GetFunctions\(\)](#).

**Comments**

-/-

### 4.3.4 Interface IDtmApplication

This interface provides the function to start a user interface of a DTM. These user interfaces are part of the DTM itself and cannot be embedded within a Frame Application.

#### 4.3.4.1 ExitApplication

```
HRESULT ExitApplication(
    [in] FdtUUIDString invokeId,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

Notification to a DTM to close an user interfaces identified by the invoke id.

Parameters	Description
invokeId	Identifier for the started application. Same value as provided in the corresponding call of <a href="#">IDtmApplication::StartApplication()</a>

**Return Value**

Return Value	Description
TRUE	The specified application will be closed
FALSE	The operation failed.

**Behavior**

This method works asynchronous. That means that the DTM just checks whether it can close the user interfaces or not. In case it can, it first returns TRUE and then starts its shut down procedure for the user interface. During this shut down it has to unlock its instance data set and release the online connection to its device if necessary. Finally, it has to notify the Frame Application via [IDtmEvents::OnApplicationClosed\(\)](#). This notification will cause the related releases on Frame Application's side. The DTM itself is not terminated.

In case of errors, the DTM should supply further details via [IDtmEvents::OnErrorMessage\(\)](#).

The invoke id is used by a Frame Application for the association at the callback interface if the application is terminated. (see [IDtmEvents::OnApplicationClosed\(\)](#)).

**Comments**

This method has to work asynchronous, because a synchronous call may block the Frame Application interfaces.

**4.3.4.2 StartApplication**

```
HRESULT StartApplication(
    [in] FdtUUIDString invokeld,
    [in] FdtXmlDocument functionCall,
    [in] BSTR windowTitle,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

Opens a user interface of a DTM for a specific function call.

Parameters	Description
invokeld	Identifier for the started application.
functionCall	XML document containing the function id for the requested function or user interface specified by the <a href="#">DTMFunctionCallSchema</a>
windowTitle	Window title required by the Frame Application

**Return Value**

Return Value	Description
TRUE	The requested application is started
FALSE	The operation failed.

**Behavior**

The function call id associates a DTM with a functional/logical context. Each DTM can provide more than one function. Which functions are supported by a DTM can be requested via the schema provided by [IDtm::GetFunctions\(\)](#).

In general, it is up to the Frame Application to determine the passed function call id and the DTM decides the kind of presentation.

[IDtmApplication::StartApplication\(\)](#) brings always a user interface to the foreground, or at least an error message. Already started applications, identified by the invoke id, will be popped to the foreground. The request of an already started application with a new invoke id will be rejected by the DTM.

The invoke id is used by a Frame Application for the association at the callback interface if the application is terminated within the user interface of the DTM. (see [IDtmEvents::OnApplicationClosed\(\)](#)). Furthermore it allows the Frame Application to handle a list of open user interfaces.

**Comments**

-/-

### 4.3.5 Interface IDtmChannel

This interface is used for accessing channel objects. On one hand the supplied channel objects carry the information which are necessary to create the association between I/O channels of a device and the functions of the Frame Application. On the other hand, in case of communication channels, these channel objects are used to build the communication chain for Nested Communication.

#### 4.3.5.1 GetChannels

HRESULT [GetChannels](#)(  
[out, retval] IFdtChannelCollection\*\* result);

**Description**

Returns the channel objects of a DTM.

Parameters	Description
------------	-------------

**Return Value**

Return Value	Description
result	Collection of IFdtChannel of the requested channel objects

**Behavior**

This method returns the channels of a DTM. The DTM itself can represent a device or a module of a device.

For simple devices a channel object provides only the information for channel assignment.

In case the channel provides gateway functionality, the channel object additionally supplies the communication interface for nested communication.

**Comments**

-/-

## 4.3.6 Interface IDtmDocumentation

This interface provides the DTM specific documentation for a device instance as XML document.

### 4.3.6.1 GetDocumentation

```
HRESULT GetDocumentation(
    [in] FdtXmlDocument functionCall,
    [out, retval] FdtXmlDocument* result);
```

**Description**

Returns the device specific documentation according to the function call as XML document.

Parameters	Description
functionCall	XML document containing the function id for the requested document specified by the <a href="#">DTMFunctionCallSchema</a>

**Return Value**

Return Value	Description
result	XML document containing the requested documentation specified by the <a href="#">DTMDocumentationSchema</a>

**Behavior**

This method returns an XML-Document which can be used directly for documentation purposes. The format of this document is defined by the passed function call id, which is available via [IDtm::GetFunctions\(\)](#) Only functions with the attribute 'printable' = TRUE will be supported. The Frame Application can use a XSL style sheet to transform the returned XML document to HTML. Nesting DTM specific style sheets can be used to transform the XML document into a DTM specific

HTML. Within these nested style sheets also hyperlinks to additional documents or into the World Wide Web can be placed.

## Comments

For an example style sheet please have a look to DTMDocumentationStyle.xsl. Refer to chapter 13.1

## 4.3.7 Interface IDtmDiagnosis

This interface provides the base diagnosis functions required by a Frame Application for DTMs with configuration parameters.

### 4.3.7.1 Compare

HRESULT [Compare](#)(  
[out, retval] VARIANT\_BOOL\* result);

## Description

Returns TRUE if the complete data sets are equal.

Parameters	Description
------------	-------------

## Return Value

Return Value	Description
TRUE	The data sets are equal
FALSE	The data sets are not equal or compare failed

## Behavior

Compares the data set of the external DTM with its own and returns TRUE if the data sets are equal.

This function fails if it is called outside of an [InitCompare\(\)](#) – [ReleaseCompare\(\)](#) sequence.

In case of errors the DTM should inform the Frame Application via the callback interface [IDTMEvt::OnErrorMessage\(\)](#).

## Comments

The structure and the parameter values for configuration, parameterization and identification are relevant for the comparison. Runtime dependent parameters (e.g. operation hours) of the data set are not relevant for comparison.

### 4.3.7.2 InitCompare

HRESULT [InitCompare](#)(  
     [in] BSTR [systemTag](#),  
     [out, retval] VARIANT\_BOOL\* result);

#### Description

Initializes a DTM for comparison of two device instances

Parameters	Description
<a href="#">systemTag</a>	<a href="#">systemTag</a> of a second DTM of the same type

#### Return Value

Return Value	Description
TRUE	Initialization successful
FALSE	Initialization failed (e. g. a compare is already in progress or the DTM is not of the same type)

#### Behavior

Initializes the compare of the data set owned by the DTM itself with a data set of a second device. Such a comparison is only possible within an [InitCompare](#)() – [ReleaseCompare](#)() sequence.

The DTM can access the second device data set by requesting the according DTM instance via [IFdtTopology::GetDtmForSystemTag](#)() with the received [systemTag](#).

To perform a comparison in the background the [Compare](#)() method can be called. Starting a compare user interface may perform a user interactive comparison.

It is only possible to compare data sets handled by DTMs of the same type.

#### Comments

Every comparison sequence started with [InitCompare](#)() must be closed using [ReleaseCompare](#)().

### 4.3.7.3 ReleaseCompare

HRESULT [ReleaseCompare](#)(  
     [out, retval] VARIANT\_BOOL\* result);

#### Description

Returns TRUE if an existing compare sequence initialized by [InitCompare](#)() has been closed successfully.

Parameters	Description
------------	-------------



**Return Value**

Return Value	Description
TRUE	Compare sequence closed and external DTM reference released.
FALSE	A comparison is in progress (e. g. an user interface is currently open)

**Behavior**

If this function is called, the DTM has to release its reference to the external DTM by calling [IFdtTopology::ReleaseDtmForSystemTag\(\)](#)..

This method only succeeds, if the comparison is finished and the references to the external DTM are released. Especially in case of open user interfaces these references must be solved first.

**Comments**

If the [ReleaseCompare\(\)](#) function is not handled in a correct manner on both sides, the Frame Application and the DTM, the DTM referenced as the external DTM cannot be released during the lifetime of the current DTM.

#### 4.3.7.4 Verify

HRESULT [Verify](#)(  
[out, retval] VARIANT\_BOOL\* result);

**Description**

Returns TRUE if the complete data set is valid.

Parameters	Description
------------	-------------

**Return Value**

Return Value	Description
TRUE	The complete data set is valid
FALSE	The data set or a part of the data set is invalid

**Behavior**

Validates the complete data set by internal business rules of the DTM.

**Comments**

A Frame Application calls this method typically to ensure a consistent dataset, for example after persistent load or before going online.

#### 4.3.8 Interface IDtmImportExport

To build an export image of a DTM a Frame Application uses one IStream object for each device instance. This IStream object is used as argument to IDtmImportExport::Load() or

IDtmlmortExport::Save(). If a DTM does not offer an IDtmlImportExport interface the Frame Application must use one of the IPersistXXX interfaces to retrieve and restore the instance data.

### 4.3.8.1 Export

```
HRESULT Export(
    [in] IStream* stream
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Saves data of the private data storage to the specified stream.

Parameters	Description
stream	Stream containing all DTM specific data of an instance

#### Return Value

Return Value	Description
TRUE	The operation succeeded.
FALSE	The operation failed.

#### Behavior

The import/export interface is a mandatory interface for DTMs which do not store the complete instance specific data via IPersistXXX mechanism. It is up to a DTM to specify the contents of the appropriate stream object provided by the Frame Application via the IDtmlImportExport interface. If this interface is not provided by a DTM the Frame Application uses one of the IPersistXXX interfaces for export/import.

#### Comments

-/-

### 4.3.8.2 Import

```
HRESULT Import(
    [in] IStream* stream,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Loads data of the private data storage from the specified stream.

Parameters	Description
stream	Stream containing all DTM specific data of an instance

**Return Value**

Return Value	Description
TRUE	The operation succeeded.
FALSE	The operation failed.

**Behavior**

The import/export interface is a mandatory interface for DTMs which do not store the complete instance specific data via IPersistXXX mechanism. It is up to a DTM to specify the contents of the appropriate stream object provided by the Frame Application via the [IDtmImportExport](#) interface. If this interface is not provided by a DTM the Frame Application uses one of the IPersistXXX interfaces for export/import.

**Comments**

-/-

### 4.3.9 Interface IDtmInformation

This interface is the second main interface of a DTM according to FDT version 1.2 and older. Via this interface the DTM can be asked for its static information like version, vendor, and its capabilities to allow integration into the libraries of a Frame Application.

#### 4.3.9.1 GetInformation

HRESULT [GetInformation](#)(  
[out, retval] [FdtXmlDocument](#)\* result);

**Description**

Returns a static XML-document containing information like vendor, icon, GSD, ...

Parameters	Description
------------	-------------

**Return Value**

Return Value	Description
Result	XML document containing static DTM information specified by the <a href="#">DTMInformationSchema</a>

**Behavior**

This method provides a fast access to DTM specific information. This data are available as soon as the DTM is instantiated. The DTM do not need any instance specific data to provide this information.

Usually this information is used by the Frame Application to create libraries, to select a DTM, or for simple validations.

**Comments**

-/-

**4.3.10 Interface IDtmInformation2**

This interface extends the interface IDtmInformation by new methods. This interface is mandatory.

**4.3.10.1 GetDeviceIdentificationInformation()**

```
HRESULT GetDeviceIdentificationInformation (
    [in] FdtXmlDocument typeRequest,
    [in] FdtUUIDString protocolId,
    [out, retval] FdtXmlDocument* result);
```

**Description**

Requests device or block identification information for specified type and protocol.

Parameters	Description
typeRequest	Defines the DTMDDeviceType or BTMBlockType for which the identification is requested. (XML according <a href="#">TypeRequestSchema</a> )
protocolId	Defines UUID of protocol for which the device identification is requested.

**Return Value**

Return Value	Description
result	<p>Protocol specific device identification information for a DTMDDeviceType specified by a field bus specific schema. (<a href="#">FDTxxxDeviceTypeIdentSchema</a> where xxx is the protocol name)</p> <p>If method was called at a Communication-DTM, then XML document according <a href="#">DTMDDeviceTypeIdentSchema</a> is returned and must not be transformed.</p>

**Behavior**

This method returns device identification information for a requested device type. The response contains a protocol specific document, which can be validated by Frame Application against the protocol specific schemas.

**Comments**

DTM must check frames protocol specific FDT schema sub-path for an already existing protocol specific schema. If the protocol schemas are missing Device-DTM has to inform the user about missing schema which is provided by a Communication-DTM of the required protocol.

See usage of information returned by [IDtmInformation2::GetDeviceIdentificationInformation\(\)](#) in chapter 4.10.8

## 4.3.11 Interface IDtmOnlineDiagnosis

This interface provides an optional online diagnosis functions used by a Frame Application to validate complete bus systems within a batch process.

### 4.3.11.1 Compare

HRESULT [Compare](#)(  
[out, retval] [FdtXmlDocument](#)\* result);

#### Description

Returns an XML document containing the result of the compare.

Parameters	Description
------------	-------------

#### Return Value

Return Value	Description
Result	XML document containing the result of the compare specified by the <a href="#">DTMOnlineCompareSchema</a>

#### Behavior

Compares its data set received from the database with the parameter uploaded from the according device.

If the data stored in database and the data uploaded from the device could be compared the result shows whether the data are equal or not. Otherwise the document contains the communication error.

This method is used for batch processing and works without user interface.

If the DTM has no comparable online data, it must return 'noComparableData' as value of the attribute 'statusFlag'.

#### Comments

Comparison should only include data significant for the configuration, parameterization and identification of the device. Data related to runtime (e.g. operation hours) should not be included.

### 4.3.11.2 GetDeviceStatus

HRESULT [GetDeviceStatus](#)(  
[out, retval] [FdtXmlDocument](#)\* result);

#### Description

Returns an XML document which describes the status of the device.

Parameters	Description
------------	-------------

#### Return Value

Return Value	Description
Result	XML document containing the status of the device specified by the <a href="#">DTMDeviceStatusSchema</a>

#### Behavior

The DTM loads the current status from the device. Depending on the fieldbus protocol, the DTM should additionally upload its actual diagnosis information. Depending on this information the DTM provides a human readable status and returns the information within an XML document. The function must work without a user interface to allow the check of complete networks.

#### Comments

A BTM must return the status of the related block.

## 4.3.12 Interface IDtmOnlineParameter

This interface allows a Frame Application the online access to a device. This interface is mandatory for all devices which must be loaded during commissioning.

### 4.3.12.1 CancelAction

HRESULT [CancelAction](#)(  
[in] [FdtUUIDString](#) invokeld,  
[out, retval] VARIANT\_BOOL\* result);

#### Description

Cancels an active parameter-upload or download.

Parameters	Description
invokeld	Identifier of the action to be canceled

**Return Value**

Return Value	Description
TRUE	Cancel action accepted
FALSE	Cancel action cannot be performed

**Behavior**

The method cancels an active parameter-upload or download. If the DTM has accepted the CancelAction request it returns TRUE. The DTM may not be able to cancel the action immediately after accepting the request, but it should do so as soon as possible. The DTM must not fire the [IDtmEvents::OnDownloadFinished\(\)](#) or [IDtmEvents::OnUploadFinished\(\)](#) events.

If the DTM cannot cancel the selected action, it must return FALSE and must fire one of the “finished” events when the action is finished.

**Comments**

-/-

**4.3.12.2 DownloadRequest**

```
HRESULT DownloadRequest(
    [in] FdtUUIDString invokeId,
    [in] FdtXPath parameterPath,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

Sends the request to write online data to the device.

Parameters	Description
invokeId	Identifier of the request
parameterPath	FdtXPath within the XML document

**Return Value**

Return Value	Description
TRUE	Request accepted
FALSE	Request cannot be performed

**Behavior**

Asynchronous function call that sends an XML document with the device specific parameters according to the specified schema of [IDtmParameter::GetParameters\(\)](#) to the connected device. The response whether the download was successful will be provided by [IDtmEvents::OnDownloadFinished\(\)](#).

In case of errors the DTM should inform the Frame Application via the callback interface [IDtmEvents::OnErrorMessage\(\)](#).

**Downloading of all parameters of the device, will be done if the path contains the root tag. In this case the DTM sends all parameters for the commissioning of the device.**

#### Comments

-/-

### 4.3.12.3 UploadRequest

```
HRESULT UploadRequest(
    [in] FdtUUIDString invokeld,
    [in] FdtXPath parameterPath,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Sends the request to read online data from a device.

Parameters	Description
invokeld	Identifier of the request
parameterPath	FdtXPath within the XML document

#### Return Value

Return Value	Description
TRUE	Request accepted
FALSE	Request cannot be performed

#### Behavior

Asynchronous function call that requires a DTM to upload parameters according to the path which points to an element of the XML document of [IDtmParameter::GetParameters\(\)](#) from the connected device. The response whether the upload was successful will be provided by [IDtmEvents::OnUploadFinished\(\)](#).

In case of errors the DTM should inform the Frame Application via the callback interface [IDtmEvents::OnErrorMessage\(\)](#).

**Uploading of all parameters of the device, will be done if the path contains the root tag. In this case the DTM loads all parameters from the device which were send during commissioning.**

#### Comments

-/-



## 4.3.13 Interface IDtmParameter

This interface allows a Frame Application the access to device parameters. The DTM provides its actual in-memory representation of its instance data set. It is up to a DTM and depends on the device- and fieldbus-type which parameters are available.

### 4.3.13.1 GetParameters

```
HRESULT GetParameters(
    [in] FdtXPath parameterPath,
    [out, retval] FdtXmlDocument* result);
```

#### Description

Returns an XML document with the device specific parameters.

Parameters	Description
parameterPath	FdtXPath within the XML document

#### Return Value

Return Value	Description
result	XML document with the device specific parameters specified by the <a href="#">DTMParameterSchema</a>

#### Behavior

Returns an XML document with the device specific parameters. The document can be empty for devices without public data.

The method provides the transient data of a DTM. That means, if the DTM is active or has for example an open user interface the provided parameter can differ from the actual stored instance data.

The state of the received data is classified within the [DTMParameterSchema](#).

The DTM must always return within the XML document the current DtmDeviceType. That means in case of an update of the DTM the changed DtmDeviceType must be returned instead of the DtmDeviceType given during [IDtm::InitNew\(\)](#).

#### Comments

The integration of devices depends on the amount of data which is available via this method. Thus a DTM should provide all data which are necessary to support a seamless integration.

See also chapter: [State machine of instance data](#)

### 4.3.13.2 SetParameters

```
HRESULT SetParameters(
    [in] FdtXPath parameterPath,
    [in] FdtXmlDocument xmlDocument,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Sets changes of device specific parameters.

Parameters	Description
parameterPath	FdtXPath within the XML document
xmlDocument	XML document specified by the <a href="#">DTMParameterSchema</a>

#### Return Value

Return Value	Description
TRUE	The DTM has accepted the complete document
FALSE	The document contains invalid data and was not accepted

#### Behavior

Only values of the writable elements can be changed by calling SetParameters. The DTM must verify complete document according to the business rules before accepting the requested changes.

The method returns TRUE if the DTM has accepted the changes for the complete document.

The method returns FALSE if the DTM has rejected any of the value changes. The Transient data will remain unchanged. The DTM informs the Frame Application about the error via the callback interface [IDtmEvents::OnErrorMessage\(\)](#).

The method works on the transient data of a DTM. That means that the new data are not stored implicitly.

The DTM can request transient data to become persistent by calling [IFdtContainer::SaveRequest\(\)](#).

#### Comments

See also chapter: [State machine of instance data](#)

### 4.3.14 Interface IFdtCommunicationEvents

This interface IFdtCommunicationEvents is the callback-interface for the associated communication component.

### 4.3.14.1 OnAbort

HRESULT [OnAbort](#)(  
     [in] [FdtUUIDString communicationReference](#));

#### Description

Notification that the connection identified by the [communicationReference](#) has been aborted.

Parameters	Description
<a href="#">communicationReference</a>	Unique identifier for the connection

#### Return Value

Return Value	Description
--------------	-------------

#### Behavior

The method sends the abort notification to a connected communication component or to a connected DTM, that a connection is terminated. All pending requests on this connection are also terminated. The termination of the connection will not be confirmed.

A termination of a connection can be result of an invoked function or can occur “spontaneous” (e.g. if the absense of a device is noted automatically by the underlying communication infrastructure).

#### Comments

The difference between [IFdtCommunicationEvents::OnAbort\(\)](#) and all ofther events of this interface is, that OnAbort provides information regarding the state of a connection. All other events provide information regarding an invoked functionality.

### 4.3.14.2 OnConnectResponse

HRESULT [OnConnectResponse](#)(  
     [in] [FdtUUIDString invokeld](#),  
     [in] [FdtXmlDocument](#) response);

#### Description

Provides the response of [ConnectRequest\(\)](#) identified by the invoke id.

Parameters	Description
<a href="#">invokeld</a>	Unique identifier for the request
<a href="#">response</a>	Fieldbus-protocol-specific information about the established connection specified by a fieldbus specific schema e.g. like <a href="#">FDTHARTCommunicationSchema</a> or <a href="#">FDTProfibusCommunicationSchema</a>

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

Via this method the sender of the connect-request receives from the next communication component the information whether the connection is established.

The method provides an XML document with communication parameters specified by a fieldbus specific schema.

**Comments**

-/-

### 4.3.14.3 OnDisconnectResponse

```
HRESULT OnDisconnectResponse(
    [in] FdtUUIDString invokeld,
    [in] FdtXmlDocument response);
```

**Description**

Provides the response of [DisconnectRequest\(\)](#) identified by the invoke id.

Parameters	Description
invokeld	Unique identifier for the request
response	Fieldbus-protocol-specific information about the released connection specified by a fieldbus specific schema e.g. like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

Via this method the sender of the disconnect-request receives from the next communication component the information whether the connection is released.

The method provides an XML document with communication parameters specified by a fieldbus specific schema.

[OnDisconnectResponse\(\)](#) causes the release all pending response data and at least the release of the callback pointer passed during [ConnectRequest\(\)](#)

**Comments**

-/-

#### 4.3.14.4 OnTransactionResponse

```
HRESULT OnTransactionResponse(
    [in] FdtUUIDString invokeld,
    [in] FdtXmlDocument response);
```

##### Description

Provides the response of [TransactionRequest\(\)](#) identified by the invoke id.

Parameters	Description
invokeld	Unique identifier for the request
response	Received data, status- and error-codes specified by a fieldbus-specific schema e.g. like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

##### Return Value

Return Value	Description
--------------	-------------

##### Behavior

Via this method the sender of the data-exchange-request receives from the next communication component the transferred communication data.

The method provides an XML document with communication parameters specified by a fieldbus specific schema and results in the release of the response data within the response data queue communication component.

##### Comments

-/-

#### 4.3.15 Interface IFdtCommunicationEvents2

This interface IFdtCommunicationEvents2 is the callback-interface for a DTM supporting FDT version 1.2.1 or higher version for the associated parent communication component. This interface extends the interface IFdtCommunicationEvents by new methods. This interface is mandatory.

A parent component supporting 1.2.1 or higher version has to call IFdtCommunicationEvents2, if the DTM supports this interface. Instead of calling [IFdtCommunicationEvents::OnConnectResponse\(\)](#) such a parent component must then use the [IFdtCommunicationEvents2::OnConnectResponse2\(\)](#) method.

### 4.3.15.1 OnConnectResponse2

```
HRESULT OnConnectResponse2(
    [in] FdtUUIDString invokeld,
    [in] FdtXmlDocument parentinformation,
    [in] FdtXmlDocument response);
```

#### Description

Provides the response of [ConnectRequest\(\)](#) identified by the invoke id.

Parameters	Description
invokeld	Unique identifier for the request
parentinformation	version information of the parent component according to <a href="#">FDTConnectResponseSchema</a> .
response	Fieldbus-protocol-specific information about the established connection specified by a fieldbus specific schema e.g. like <a href="#">FDTHARTCommunicationSchema</a> or <a href="#">FDTProfibusCommunicationSchema</a>

#### Return Value

Return Value	Description
--------------	-------------

#### Behavior

Via this method the sender of the connect-request receives from the next communication component whether the connection is established.

The method provides

- with parameter “parentinformation” the FDT version of the parent component. DTM may only use XML communication documents compatible to the version of the parent component.
- with parameter “response” an XML document with communication parameters specified by a fieldbus specific schema.

#### Comments

-/-

### 4.3.16 Interface IFdtEvents

This interface is the callback-interface for the Frame Application.

### 4.3.16.1 OnChildParameterChanged

HRESULT [OnChildParameterChanged](#)(  
[in] BSTR [systemTag](#));

#### Description

In case of a DTM topology, it can be necessary to inform the parent DTM about parameter changes.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

#### Return Value

Return Value	Description
--------------	-------------

#### Behavior

If a DTM has changed any data it has to call [IDtmEvents::OnParameterChanged\(\)](#) with an XML document containing the instance specific changes. The Frame Application will send this document via [IFdtEvents::OnParameterChanged\(\)](#) to all DTMs which reference the same device instance. Concerning the according parent DTMs (primary parent, secondary parents, see [IFdtTopology::GetParentNodes\(\)](#)), the Frame Application must implement the following behavior:

- The Frame Application must send a notification to the according parent DTM via [IFdtEvents::OnChildParameterChanged\(\)](#)
- Within a multi user environment, this notification will only be send to one parent DTM instance
- This notification must be send to the parent DTM which has the lock concerning the related instance data set
- If currently no parent DTM is started, the Frame Application must start the parent DTM

#### Comments

The parent DTM gets only a notification, because the XML document, exchanged via [OnParameterChanged\(\)](#), is DTM specific and cannot be interpreted by a parent DTM. A parent DTM, which receives such a notification, can update its child specific data by calling [GetParameters\(\)](#) at the child DTM. The Frame Application has to ensure, that the parent DTM instance which will be selected to perform [IFdtEvents::OnChildParameterChanged\(\)](#), is in any case capable to lock its data set.

### 4.3.16.2 OnLockDataSet

HRESULT [OnLockDataSet](#)(  
[in] BSTR [systemTag](#),  
[in] BSTR [userName](#));

#### Description

In case of a multi-user environment, it is necessary to inform the DTM about its current access mode especially if another DTM gets the read/write access for its data set .

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
userName	Human readable name of the user who has locked the data set

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

If a DTM has locked a data set via [IFdtContainer::LockDataSet\(\)](#) the Frame Application has to send [OnLockDataSet\(\)](#) to all DTM instances which have a reference to the same data set.

Receiving this notification a DTM should disable its input fields in case of an open user interface and keep in mind that it is not allowed to perform any function which needs any data storage.

**Comments**

The userName can be the login name of a user or an identifier within a user management of a Frame Application. At least the userName should provide the information where to find the other user within a multi-user environment.

### 4.3.16.3 OnParameterChanged

```
HRESULT OnParameterChanged(
    [in] BSTR systemTag,
    [in] FdtXmlDocument parameter);
```

**Description**

In case of a multi-user environment, it can be necessary to inform the DTM about parameter changes.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
Parameter	XML document containing the changed parameters

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

If a DTM has stored any changed data it has to call [IDtmEvents::OnParameterChanged\(\)](#) with an XML document containing DTM specific information. The Frame Application will send this document via [IFdtEvents::OnParameterChanged\(\)](#) to all DTMs that reference the same device instance.

**Comments**



#### 4.3.16.4 OnUnlockDataSet

```
HRESULT OnUnlockDataSet(
    [in] BSTR systemTag,
    [in] BSTR userName,
    [out, retval] VARIANT_BOOL* result);
```

##### Description

In case of a multi-user environment, it is necessary to inform a DTM about its current access mode especially if another DTM gets the read/write access for its data set .

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
username	Human readable name of the user who has locked the data set

##### Return Value

Return Value	Description
TRUE	The DTM has actual data
FALSE	The DTM has old data and must be closed

##### Behavior

If a DTM has unlocked a data set via [IFdtContainer::UnlockDataSet\(\)](#) the Frame Application has to send [OnUnlockDataSet\(\)](#) to all DTM instances which have a reference to the same data set.

Receiving this notification a DTM returns TRUE if it has current data and can enable its input fields in case of an open user interface. To support this feature the DTM has to implement a synchronization mechanism for its DTM instances via [OnParameterChanged\(\)](#). If the DTM does not support such synchronization it has to return FALSE. That means that the DTM has old data and will not get write access for a data set. In this case of an open user interface the Frame Application will inform the user that he has to close and to restart the DTM.

##### Comments

-/-

#### 4.3.17 Interface IDtmHardwareIdentification

This interface is used by Frame Application to detect if specific communication hardware is available or to request information from a field device.

The interface is for example implemented by Communication-DTMs to detect if corresponding hardware is responsive or by Gateway-/Device-DTMs to request manufacturer specific identification information from a field device.

### 4.3.17.1 ScanHardwareRequest

```
HRESULT ScanHardwareRequest(
    [in] FdtUUIDString invokeID,
    [in] FdtXmlDocument request,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Requests scan for availability of hardware and corresponding identification information.

Parameters	Description
invokeID	Unique identifier for the request.
request	Always set to "<FDT/>" (parameter is reserved for future use)

#### Return Value

Return Value	Description
TRUE	DTM has accepted the call.
FALSE	The operation failed.

#### Behavior

Frame Application executes this function to check if corresponding hardware is responsive and to request identification information.

DTM should connect to the device and request required information. DTM must call [IDtmScanEvents::OnScanHardwareResponse\(\)](#) when operation has finished.

#### Comments

DTM must also call [IDtmScanEvents::OnScanHardwareResponse\(\)](#) and pass XML according a field bus specific schema ([FDTxxxScanIdentSchema](#)) if requests fails (i.e. because of communication failures).

### 4.3.17.2 CancelAction

```
HRESULT CancelAction (
    [in] FdtUUIDString invokeID,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Cancels active hardware check request.

Parameters	Description
<a href="#">invokeID</a>	Unique identification of scan hardware request.

**Return Value**

Return Value	Description
TRUE	Cancel action accepted
FALSE	Cancel action not accepted.

**Behavior**

The Frame Application calls this method to cancel started scan hardware operation. If the DTM can canceled the operation it will return TRUE and will not fire the [IDtmScanEvents::OnScanHardwareResponse\(\)](#) event.

If the DTM can't cancel the operation then it returns FALSE and will fire the [IDtmScanEvents::OnScanHardwareResponse\(\)](#) event when operation has finished.

**Comments**

- / -

### 4.3.18 Interface IDtmSingleDeviceDataAccess

This interface allows a Frame Application the online access to specific parameters of a device.

This interface is implemented in an asynchronous way. The data in the device can be accessed by multiple threads of the Frame Application. For example: the Frame Application can perform a [IDtmOnlineParameter::DownloadRequest\(\)](#) in parallel to a [WriteRequest\(\)](#) via this interface.

The DTM must be prepared for multiple asynchronous requests in parallel. The requests must be processed in the order received.

**IDtmSingleDeviceDataAccess methods must not modify the instance data set , but must taken the attribute 'modifiedInDevice' (see [FDTDataTypesSchema](#)) into account.**

#### 4.3.18.1 CancelRequest

```
HRESULT CancelRequest (
    [in] FdtUUID invokeID,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

Cancels active read, write or item list request identified by its invoke ID.

Parameters	Description
invokedID	Unique identification of a read or write request.

**Return Value**

Return Value	Description
TRUE	Request to cancel pending request is accepted.
FALSE	Request to cancel pending request is not accepted.

**Behavior**

The Frame Application calls this method to cancel an active request. If the DTM has accepted the `CancelRequest()`, it returns TRUE and must not fire the response event for the canceled operation.

The DTM may not be able to cancel the action immediately after accepting the request, but it should do it as soon as possible.

**Comments**

-/-

**4.3.18.2 ItemListRequest**

HRESULT [ItemListRequest](#)(  
     [in] [FdtUUIDString](#) invokeld )

**Description**

`ItemListRequest` performs an asynchronously request of an XML document containing a list of the available device specific parameters and process values.

Parameters	Description
------------	-------------

**Return Value**

Return Value	Description
invokeld	Unique identifier for the request

**Behavior**

Via this method the Frame Application may request a list of items that can be accessed via the DTM. The source for this data is the device itself. The DTM must always accept the request. If the request cannot be processed, the reason for failure must be provided asynchronously as part of the response. The response (either failure or the result) must be provided at [IDtmSingleDeviceDataAccessEvents::OnItemListResponse\(\)](#)

**Comments**

Dependent on the user roles, the contents of the item list may change.

### 4.3.18.3 ReadRequest

HRESULT [ReadRequest](#)(  
     [in] [FdtUUIDString](#) invokeld,  
     [in] [FdtXmlDocument](#) itemSelectionList);

#### Description

ReadRequest performs asynchronous exchange of a data structure with the related device via the DTM.

Parameters	Description
Invokeld	Unique identifier for the request
itemSelectionList	List of required items described by a DtmItemSelectionList specified by the <a href="#">DTMItemListSchema</a>

#### Return Value

Return Value	Description
--------------	-------------

#### Behavior

Via this method a Frame Application may request data from a device. Error information will be handed over to the Frame Application via the related response XML-Document. If a request can not be accepted by the DTM it is possible to send the response within the call.

Execution of the [ReadRequest\(\)](#) method must not change the data of the instance data set.

The DTM must always accept the request. If the request cannot be processed, the reason for failure must be provided asynchronously as part of the response. The response (either failure or the result) must be provided at [IDtmSingleDeviceDataAccessEvents::OnReadResponse\(\)](#).

#### Comments

In order to inform the Frame Application regarding ongoing activities it is recommended to fire the [IDtmEvents::OnProgress\(\)](#) event while a response is pending.

The DTM should be able to handle more than one request at a time. The order of execution is like the order of the requests. For each request there should be a corresponding response. If a request can not be executed, an appropriate response must be provided.

### 4.3.18.4 WriteRequest

HRESULT [WriteRequest](#)(  
     [in] [FdtUUIDString](#) invokeld,  
     [in] [FdtXmlDocument](#) itemList);

#### Description

WriteDeviceRequest performs asynchronous exchange of a data structure with a DTM.

Parameters	Description
invokeld	Unique identifier for the request
itemList	List of required items described by a DtmlItemList specified by the <a href="#">DTMItemListSchema</a>

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

Via this method the Frame Application requests a DTM to write the specified data to its device according to the device specific rules. Error information will be handed over to the Frame Application via the related response XML-Document. If a request can not be accepted by the DTM it is possible to send the response within the call.

Execution of the [WriteRequest\(\)](#) method must not change the data of the instance data set.

The DTM has to check, whether it could manipulate the flag 'modifiedInDevice' (refer chapter 'FDT Data Types) in the instance data set by requesting a lock. If the lock request fails the DTM has also to refuse the [WriteRequest\(\)](#) - an appropriate response must be provided.

The DTM must always accept the request. If the request cannot be processed, the reason for failure must be provided asynchronously as part of the response. The response (either failure or the result) must be provided at [IDtmSingleDeviceDataAccessEvents::OnWriteResponse\(\)](#).

**Comments**

In order to inform the Frame Application regarding an ongoing activities It is recommended to fire the [IDtmEvents::OnProgress\(\)](#) event while a response is pending.

The DTM should be able to handle more than one request at a time. The order of execution is like the order of appearance of the requests. For each request there should be a corresponding response.

### 4.3.19 Interface IDtmSingleInstanceDataAccess

This interface allows a Frame Application the offline access to specific parameters of a device.

### 4.3.19.1 GetItemList

HRESULT [GetItemList](#)(  
[out, retval] [FdtXmlDocument](#)\* result);

#### Description

GetItemList returns an XML document containing a list of the available device specific parameters. Within a DTM this list may contain items related to configuration parameters as well as asset management related data.

Parameters	Description
------------	-------------

#### Return Value

Return Value	Description
Result	XML document containing a DtmItemInfoList with the actual available parameters specified by the <a href="#">DTMItemListSchema</a>

#### Behavior

This method provides a list of items that can be accessed via the DTM. The source for this data is the DTM instance data set.

Items provided within these list may also be available as channel objects (provided by [IDtmChannel::GetChannels\(\)](#)) or modeled as an exported variable (DtmVariable provided by [IDtmParameter::GetParameters\(\)](#) or IBtmParameter::GetParameters()). The relation of these items can be identified via the attribute 'semanticId'

#### Comments

The contents of the provided XML document may depend on the current configuration of the device. If the contents is changed, a DTM has to inform the Frame Application by sending [IDtmSingleInstanceDataAccess::OnItemChanged\(\)](#).

Dependent on the user roles, the item list items may vary.

### 4.3.19.2 Read

HRESULT [Read](#)(  
[in] [FdtXmlDocument](#) itemSelectionList,  
[out, retval] [FdtXmlDocument](#)\* result);

#### Description

Read performs synchronous exchange of a data structure with the related instance data set.

Parameters	Description
itemSelectionList	List of required items described by a DtmItemSelectionList specified by the <a href="#">DTMItemListSchema</a>

**Return Value**

Return Value	Description
Result	Requested data as DtmlItemList specified by the <a href="#">DTMItemListSchema</a>

**Behavior**

Via this method a Frame Application may request data from an instance data set.

**Comments****4.3.19.3 Write**

```
HRESULT Write(
    [in] FdtXmlDocument itemList,
    [out, retval] FdtXmlDocument* result);
```

**Description**

Write performs synchronous exchange of a data structure with a DTM.

Parameters	Description
itemList	List of required items described by a DtmlItemList specified by the <a href="#">DTMItemListSchema</a>

**Return Value**

Return Value	Description
Result	Data as DtmlItemList that contains the device data of the successfully written data specified by the <a href="#">DTMItemListSchema</a> (may differ to the written value due to e. g. rounding procedures within the device)

**Behavior**

Via this method the Frame Application requests a DTM to write the specified data to its instance data set. The DTM has to check, whether it could manipulate the instance data set by requesting a lock. If the lock request fails the DTM has also to refuse the request.

Furthermore the DTM has to apply the business rules in order to keep the instance data set consistent.

**Comments**



## 4.4 DTM ActiveXControl

### 4.4.1 Interface IDtmActiveXControl

This interface is an extension of a standard ActiveX control and allows connecting a DTM object with the ActiveX control.

#### 4.4.1.1 Init

```
HRESULT Init(
    [in] FdtUUIDString invokeId,
    [in] FdtXmlDocument,
    [in] IDtm* dtm,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Set the callback pointer of an ActiveX control to the according DTM.

Parameters	Description
invokeId	Identifier for the started application.
functionCall	XML document containing the function id for the requested function or user interface specified by the <a href="#">DTMFunctionCallSchema</a>
Dtm	Pointer to the DTM business object

#### Return Value

Return Value	Description
TRUE	The control is initialized
FALSE	The operation failed.

#### Behavior

Set the callback pointer of an ActiveX control to the according DTM. The function id can be used to toggle a user interface during runtime. It is up to the control whether its supports this functionality.

If the initialization returns FALSE, the Frame Application has to release the control.

The invoke id is used by a Frame Application for the association at the callback interface if the control is terminated within the user interface of the DTM (see [IDtmEvents::OnApplicationClosed\(\)](#)). Furthermore it allows the Frame Application to handle a list of open user interfaces.

#### Comments

-/-

#### 4.4.1.2 PrepareToRelease

HRESULT [PrepareToRelease](#)(  
[out, retval] VARIANT\_BOOL\* result);

##### Description

Used to inform the DTM control that it has to release its links to other components. The Frame Application will release the control after the DTM has send [IDtmEvents::OnApplicationClosed](#) ().

Parameters	Description
------------	-------------

##### Return Value

Return Value	Description
TRUE	The request was accepted
FALSE	The operation rejected.

##### Behavior

If the request is accepted, the ActiveX will release the callback pointer to the DTM set during [Init](#) (). Furthermore the control has to release all links to other components and has to terminate all pending or running functions. The DTM has to inform the Frame Application via [IDtmEvents::OnApplicationClosed](#) () that the user interface could be released.

If FALSE is returned, the DTM will not call [OnApplicationClosed](#) () and will preserve its current state

##### Comments

-/-

## 4.5 FDT Channel

### 4.5.1 Interface IFdtChannel

This is the main interface of a channel that provides all information which is necessary for the channel assignment.

#### 4.5.1.1 GetChannelParameters

```
HRESULT GetChannelParameters(
    [in] FdtXPath parameterPath,
    [in] FdtUUIDString protocolId,
    [out, retval] FdtXmlDocument* result);
```

#### Description

Returns an XML document with fieldbus dependent channel specific parameters.

Parameters	Description
parameterPath	FdtXPath within the XML document
protocolId	UUID of a fieldbusprotocol. This UUID may be <ul style="list-style-type: none"> <li>one of the UUIDs returned by <code>GetSupportedProtocols()</code> and specified by <code>DTMProtocolsSchema</code></li> <li>a UUID defined within the appropriate <code>&lt;ChannelReference&gt;</code> element of the <code>DTMParameter</code> document returned by the DTM</li> </ul>

#### Return Value

Return Value	Description
Result	XML document with the channel specific parameters specified by a schema e.g. like <code>FDTHARTChannelParameterSchema</code> or <code>FDTProfibusChannelParameterSchema</code>

#### Behavior

Returns an XML document with the channel specific parameters specified by a fieldbus specific schema. The fieldbus is selected by the parameter `protocolId`. The returned parameters are especially used for channel assignment. The document can be empty for devices without fieldbus master.

Channels that do not have any process related data (e.g. a pure Communication-Channel) should return a document based on [FDTBasicChannelParameterSchema](#).

It is recommended, to return instead of an empty document a document based on the [FDTBasicChannelParameterSchema](#).

If the requested protocol UUID is not supported by this channel it is recommended to return a document based on the [FDTBasicChannelParameterSchema](#). This at least gives some basic info for the caller.

## Comments

-/-

### 4.5.1.2 GetChannelPath

```
HRESULT GetChannelPath(
    [out, retval] FdtXPath* result);
```

## Description

Returns an identifier for a channel.

Parameters	Description
------------	-------------

## Return Value

Return Value	Description
Result	Path that identifies the channel within the device.

## Behavior

Returns the path that identifies the channel within the device. The string must not be empty. It always starts with the [systemTag](#) of the device instance followed by the channel id. The DTM has to guarantee that the path is unique for a device instance. The channel path is the base information to handle the system structure at [IFdtTopology](#) and [IFdtChannelSubTopology](#).

[<systemTag>/<id>](#)

Furthermore the channelPath contains the information where to find the channel within the parameter document available via [IDtmParameter::GetParameters\(\)](#) and so at least in case of a monolithic DTM the information whether the channel belongs to a device or module.

In case of communication channels there are some special rules for building the channelPath.

How to generate the channelPath of a communication channel, which also acts as a process channel for data that it receives from a process channel of a child device, depends on the functionality of the channel.

If the channel is passive, that means that it receives its data from a child device and provides this data without any changes within its own channel-parameter document, the channelPath must be built out of system tag and channel id of the own device instance and the system tag of the child device and the id of the marshalled channel.

[<systemTag>/<id>//<systemTag>/<id>](#)

If the channel is active, that means that it receives its data from a child device for processing and provides the result within its own channel-parameter document, the channelPath must be built out of the system tag and channel id of the own device instance.

<systemTag>/<id>

This allows navigation through the internal channel assignment of a device with gateway functionality.

In general if the channelPath of an channel has changed the according DTM has to send [OnParameterChanged\(\)](#) with respect to the possible behavior of other FDT objects.

### Comments

An example for such a channel is a Profibus remote I/O that reads the primary variable (provided as HART channel) of an underlying HART device and provides this value within its own cyclic I/O data (Profibus DP Channel).

## 4.5.1.3 SetChannelParameters

```
HRESULT SetChannelParameters(
    [in] FdtXPath parameterPath,
    [in] FdtUUIDString protocolId,
    [in] FdtXmlDocument xmlDocument,
    [out, retval] VARIANT_BOOL* result);
```

### Description

Sets changes of channel specific parameters.

Parameters	Description
parameterPath	FdtXPath within the XML document
protocolId	UUID of a fieldbusprotocol. This UUID must be one of the UUIDs returned by <a href="#">GetSupportedProtocols()</a> and specified by <a href="#">DTMProtocolsSchema</a>
xmlDocument	XML document specified by a schema e.g. like FDTHARTChannelParameterSchema or FDTProfibusChannelParameterSchema

### Return Value

Return Value	Description
TRUE	The channel has accepted the complete document with all changes
FALSE	The document contains invalid changes

### Behavior

The method passes an XML document with the channel specific parameters according to a fieldbus specific schema. The fieldbus is defined by the parameter protocolId.

Returns TRUE if the channel has accepted the complete document with all changes. FALSE means that the channel has rejected all transferred changes. In this case the channel informs the Frame Application about the error in detail via the callback interface [IDtmEvents::OnErrorMessage\(\)](#).

The method works on the transient data of a DTM. That means that the new data are not stored implicitly. Transient data will become persistent e.g. by calling [IFdtContainer::SaveRequest\(\)](#).

## Comments

See also chapter: State machine of instance data

## 4.5.2 Interface IFdtChannelActiveXInformation

Usually an FdtChannel does not have a user-interface, but in case of communication channels it may be required to have one. Depending on the fieldbus protocol and the capability of the communication channel it might be necessary to implement a user interface to configure the communication itself.

For example, if a hardware driver is included, parameters like selected COM port or interrupt addresses must be set by the user. These data are encapsulated within the communication channel and can only be configured by a gateway specific user-interface

### 4.5.2.1 GetChannelActiveXGuid

```
HRESULT GetChannelActiveXGuid(
    [in] FdtXmlDocument,
    [out, retval] FdtUUIDString* result);
```

## Description

Returns the UUID for the ActiveX control according to the function call.

Parameters	Description
functionCall	XML document containing the function id for the requested user interface specified by the <a href="#">DTMFunctionCallSchema</a>

## Return Value

Return Value	Description
result	UUID for an ActiveX control

## Behavior

Returns a UUID that the Frame Application can use to instantiate the control.

If a requested function is not supported the method returns NULL pointer

For a communication channel, it would be the user interface to set communication specific parameters.

## Comments

-/-

### 4.5.2.2 GetChannelActiveXProgId

HRESULT [GetChannelActiveXProgId](#)(  
     [in] [FdtXmlDocument](#) functionCall,  
     [out, retval] BSTR\* result);

#### Description

Returns the ProgId for the ActiveX control according to the function call.

Parameters	Description
functionCall	XML document containing the function id for the requested user interface specified by the <a href="#">DTMFunctionCallSchema</a>

#### Return Value

Return Value	Description
result	ProgId for an ActiveX control

#### Behavior

Returns the ProgId for the ActiveX control according to the function id. Frame Applications implemented with scripting languages can use this ProgId to instantiate the control.

If a requested application is not supported the method returns [NULL](#) pointer

For a communication channel, it would be the user interface to set communication specific parameters.

#### Comments

-/-

### 4.5.2.3 GetChannelFunctions

HRESULT [GetChannelFunctions](#) (  
     [in] [FdtXmlDocument](#) operationState,  
     [out, retval] [FdtXmlDocument](#)\* result);

#### Description

Returns an XML document containing information about standard (defined by applicationID) or additional functionalities (defined by functionId) of a DTM channel object

Parameters	Description
operationState	XML document containing the current operation phase specified by the <a href="#">FDTOperationPhaseSchema</a>

**Return Value**

Return Value	Description
result	XML document containing actual supported functions specified by the <a href="#">DTMChannelFunctionsSchema</a>

**Behavior**

This method provides the access to DTM channel object functionality, defined by the applicationIDs and specific functionality which is not within the scope of FDT. This data are available as soon as the DTM channel object is instantiated but the information may change if it is instance specific.

That means that the contents of the document can change or can at least be empty after an [OnChannelFunctionChanged\(\)](#) event. This event is sent by a DTM if the configuration results in a changed extended functionality.

Usually this information is used by the Frame Application to create menus. Only functions with user interface are supported. These user interfaces are accessible via [GetChannelActiveXGuid\(\)](#) or [GetChannelActiveXProgId\(\)](#) and work asynchronous

The asynchronous behavior is described at the according chapters.

**Comments**

-/-

### 4.5.3 Interface IFdtCommunication

This interface is the communication entry point of a channel with communication functionality. The connection of this interface with a following component builds the chain for nested communication. The communication pointer to the following communication component can be requested at the DTM which owns the communication channel.

A channel is able to support a number of different fieldbus protocols. Protocol specific XML documents are exchanged between communication channel and connected client via the IFdtCommunication and IFdtCommunicationEvents interfaces. The type of protocol to be used must be specified with a connect request.

Dividing a communication function call to a request and a response function causes a non-blocking behavior. The DTM sends one or several requests to the next communication component. For the next communication component it is not allowed to send the response to the corresponding response method within the request method.

#### 4.5.3.1 Abort

HRESULT [Abort](#)(  
     [in] [FdtXmlDocument](#) fieldbusFrame);

**Description**

Aborts a communication link to a device without any response.



Parameters	Description
fieldbusFrame	Fieldbus-protocol-specific information how to abort. The structure is specified by a fieldbus specific schema e.g. like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

The method sends the abort to the next communication component or to the connected device, terminates all pending requests and returns without waiting for a result. The termination of the connection will not be confirmed.

**Comments**

-/-

### 4.5.3.2 ConnectRequest

```
HRESULT ConnectRequest(
    [in] IFdtCommunicationEvents* callBack,
    [in] FdtUUIDString invokeld,
    [in] FdtUUIDString protocolld,
    [in] FdtXmlDocument fieldbusFrame,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

Establishes asynchronously a new communication link to a device specified by the fieldbus frame. [ConnectRequest\(\)](#) establishes a routing to a device as a peer-to-peer connection.

Parameters	Description
callBack	Callback interface for the notification if the response is available
invokeld	Unique identifier for the request
protocolld	UUID of a fieldbusprotocol to be used. Identifies type of fieldbus specific schema
fieldbusFrame	Fieldbus-protocol-specific information how to connect. The structure is specified by a fieldbus specific schema e.g. like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

**Return Value**

Return Value	Description
TRUE	Request sent
FALSE	Request refused

**Behavior**

The method passes an XML document with communication parameters specified by a fieldbus specific schema. The fieldbus protocol to be used is identified by the parameter protocolId.

Based on this information the method sends the request to the next communication component or to the connected device and returns without waiting for the established connection.

The response will be provided by [IFdtCommunicationEvents2::OnConnectResponse\(\)](#).

The fieldbus frame contains additional fieldbus-protocol-specific information for the fieldbus master how to establish the connection. For example, information like repeat counts or preamble counts in case of HART sent by a DTM is a hint for the HART master. It is up to the environment to decide whether this information fits.

Furthermore the fieldbus frame contains fieldbus-protocol-specific information how to address the device connected to a specific fieldbus.

The [systemTag](#) provided in the connect request is the [systemTag](#) of the communication client. It can be used to retrieve the IDtm interface of communication client by calling [IFdtTopology::GetDtmForSystemTag\(\)](#).

If the systemTag is empty (""), the communication client is not a DTM (may be the Frame Application or some other component).

**Comments**

See also chapter FDT 'Use Cases and Scenarios'.

**4.5.3.3 DisconnectRequest**

```
HRESULT DisconnectRequest(
    [in] FdtUUIDString invokeld,
    [in] FdtXmlDocument fieldbusFrame,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

Releases a communication link to a device by an asynchronous function call. For more than one connection to the same device, the link is identified by the communication reference which is part of the fieldbus frame.

Parameters	Description
invokeld	Unique identifier for the request
fieldbusFrame	Fieldbus-protocol-specific information how to release the connection specified by a fieldbus specific schema e.g. like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

**Return Value**

Return Value	Description
TRUE	Request sent
FALSE	Request refused

**Behavior**

The method passes an XML document with communication parameters specified by a fieldbus specific schema.

Based on this information the method sends the request to the next communication component or to the connected device, terminates all pending requests and returns without waiting for the result.

The response will be provided by [OnDisconnectResponse\(\)](#).

**Comments**

See also chapter FDT 'Use Cases and Scenarios'.

### 4.5.3.4 GetSupportedProtocols

HRESULT [GetSupportedProtocols](#)(  
[out, retval] [FdtXmlDocument](#) \* result);

**Description**

Gets a document describing the supported protocols of the communication interface.

**Return Value**

Return Value	Description
result	XML document specified by <a href="#">DTMProtocolsSchema</a> describing the protocols supported by the communication interface.

**Behavior**

Via this method the DTM that wants to establish a connection asks the next communication component for the supported protocols.

The method returns an XML document with fieldbus protocol UUIDs specified by [DTMProtocolsSchema](#). Only protocols supported by the configured sub-device can be returned.

If a channel supports more than one protocol during runtime it has to support all protocols in parallel.

[GetSupportedProtocols\(\)](#) has to return static information if a child is connected to the channel because a change may cause an invalid topology. Which protocols are supported can be determined during topology planning (see [ValidateAddChild\(\)](#), [OnAddChild\(\)](#)).

So if the communication channel can be configured to support different protocols, this can only be done if there are no connected childes.

**Comments**

A DTM, which wants to use more than one protocol, has to ask the channel for its supported protocols before it starts the communication.

### 4.5.3.5 SequenceBegin

HRESULT [SequenceBegin](#)(  
     [in] [FdtXmlDocument](#) fieldbusFrame,  
     [out, retval] VARIANT\_BOOL\* result);

#### Description

The communication component has to observe that the transaction communication calls of the block started with [SequenceBegin](#)() and closed by [SequenceEnd](#)() are finished during the period of time defined by the sequence time.

The block supports asynchronous read/write and data exchange requests.

Parameters	Description
fieldbusFrame	Fieldbus-protocol-specific information describing the sequence. The structure is specified by a fieldbus specific schema e.g. like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

#### Return Value

Return Value	Description
TRUE	Loading of sequences supported
FALSE	Function not supported

#### Behavior

After a successful sequence start the last communication component or at least the hardware itself collects all sent transaction request. This can be a sequence containing several [TransactionRequest](#)() calls on one connection. The collection of pending requests is closed by [IFdtCommunication::SequenceEnd](#)().

The fieldbusFrame parameter of this method has to contain the element 'sequence' (see schemas). This element contains the following attributes:

Attribute	Description
sequenceTime	Period of time in [ms] for the whole sequence
delayTime	Minimum Delay time in [ms] between two consecutive communication calls
communicationReference	Identifier for the communication link.

In case of a sequence time > 0 the communication component has to check, if the execution time of the complete sequence is less or equal the sequence time.

In case of a delay time > 0 the last communication component, which has collected the communication calls, has to wait the defined time after each command before it sends the next.

The communication component decides according to the associated hardware whether it can support

this function.

The method returns FALSE if the last of the following communication components, possibly caused by the associated hardware, does not supported this functionality.

The actual communication starts with the call to IFdtCommunication::SequenceStart().

For each TransactionRequest there must be a TransactionResponse. As soon as the Communication-DTM detects that the SequenceTime has expired it will send TransactionResponses for any pending Requests with a CommunicationError "sequenceTimeExpired".

### Comments

The default implementation is to hand down the function call through all communication components. Only the last communication component can decide whether it supports the functionality or not.

## 4.5.3.6 SequenceEnd

```
HRESULT SequenceEnd(
    [in] FdtXmlDocument fieldbusFrame,
    [out, retval] VARIANT_BOOL* result);
```

### Description

Closes the communication block started with [SequenceBegin\(\)](#)

Parameters	Description
fieldbusFrame	Fieldbus-protocol-specific information closing a sequence of transaction calls. The structure is specified by a fieldbus specific schema e.g. like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

### Return Value

Return Value	Description
TRUE	Sequence is closed.
FALSE	No open sequence

### Behavior

Closes the communication block started with [SequenceBegin\(\)](#) The actual communication to the device starts on [SequenceStart\(\)](#).

### Comments

The default implementation is to hand down the function call through all communication components. Only the last communication component can decide whether it supports the functionality or not.

### 4.5.3.7 SequenceStart

```
HRESULT SequenceStart(
    [in] FdtXmlDocument fieldbusFrame,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Starts the execution of a communication sequence at the controller. The communication sequence breaks on error. The communication results are accessible by the according response function calls.

Parameters	Description
fieldbusFrame	Fieldbus-protocol-specific information starting a sequence. The structure is specified by a fieldbus specific schema e.g. like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

#### Return Value

Return Value	Description
TRUE	Communication sequence started
FALSE	Communication sequence could not be started

#### Behavior

Starts the communication by executing the pending requests without any interruptions. The method returns without waiting for a result so that the calling application will not be blocked.

The communication sequence breaks on error.

The communication data or errors are accessible by the according response events [OnTransactionResponse\(\)](#).

#### Comments

The default implementation is to hand down the function call through all communication components. Only the last communication component can decide whether it supports the functionality or not.

### 4.5.3.8 TransactionRequest

```
HRESULT TransactionRequest(
    [in] FdtUUIDString invokeld,
    [in] FdtXmlDocument fieldbusFrame,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

[TransactionRequest](#) performs asynchronously exchange of a data structure with a device specified by the fieldbus frame.

Parameters	Description
invokeld	Unique identifier for the request
fieldbusFrame	Fieldbus-protocol-specific information to be transferred, specified by a fieldbus specific schema e.g. like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

**Return Value**

Return Value	Description
TRUE	Request sent
FALSE	Request refused

**Behavior**

The method passes an XML document with communication parameters specified by a fieldbus specific.

Based on this information the method sends the request for data exchange to the next communication component or to the connected device and returns without waiting for the result. In case of more than one pending request the association of request and response is done by the passed invoke id. The client is responsible to pass a unique invoke id for the specified communication link.

The response will be provided by [OnTransactionResponse\(\)](#).

If a [TransactionRequest\(\)](#) is called as part of a sequence definition and the given SequenceTime is expired, the return Value has to be false.

**Comments**

It depends on the fieldbus protocol which internal communication methods are implemented at the communication channel. For example HART supports data exchange methods while Profibus offers read/write services.

Developers of Communication channels should not expect that there will be only one pending request for a certain device at one time. For instance several clients (e.g. frame application and Device-DTM's) are trying to retrieve information from same device.

Developers of Device-DTM's should consider that the used communication infrastructure creates delays in the communication. So the Device-DTM's should limit the number of communication requests. Also the Device-DTM must be able to handle a refused request, since there may be a variety of reasons to refuse a transaction request.

Even if the underlying fieldbus protocol allows sending only one request at a time to one or more devices, Communication channels must be able to manage a number of requests.

It is expected that the requests are processed by the Communication channel in the order received if not specified otherwise by the protocol.

#### 4.5.4 Interface IFdtChannelSubTopology

This interface must be supported by communication channels. It allows validating the sub-topology beneath a channel. A Frame Application always is responsible for the sub-topology of a channel, it has to call this interface so that the channel or at least the according DTM can validate the configured connections.

#### 4.5.4.1 OnAddChild

HRESULT [OnAddChild](#)(  
[in] BSTR [childSystemTag](#));

##### Description

The channel is informed that a new device was added to the sub topology.

Parameters	Description
<a href="#">childSystemTag</a>	SystemTag of the newly added child instance

##### Return Value

Return Value	Description
--------------	-------------

##### Behavior

The channel is informed that a new device, specified by its [systemTag](#), has been added to the sub topology.

If the channel needs more information about the child DTM it can get the DTM via [IFdtTopology::GetDtmForSystemTag\(\)](#) passing the received [systemTag](#).

##### Comments

This method will only be used in order to inform a parent DTM that the topology was changed. In case of reloading e.g. project related data, this method must not be called.

#### 4.5.4.2 OnRemoveChild

HRESULT [OnRemoveChild](#)(  
[in] BSTR [childSystemTag](#));

##### Description

The channel is informed that a device was removed from the sub topology.

Parameters	Description
<a href="#">childSystemTag</a>	SystemTag of the child DTM which was removed

##### Return Value

Return Value	Description
--------------	-------------

##### Behavior

The channel is informed that a device, specified by its [systemTag](#), was removed from the sub topology.



If the DTM needs more information about the child DTM it can get the DTM via [IFdtTopology::GetDtmForSystemTag\(\)](#) passing the received [systemTag](#).

#### Comments

This method will only be used in order to inform a parent DTM that the topology was changed. In case of destruction e.g. closing a project, this method must not be called.

A DTM should release all references to the removed child before returning from [OnRemoveChild\(\)](#)

### 4.5.4.3 ScanRequest

```
HRESULT ScanRequest(
    [in] FdtUUIDString invokeld,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Requests the asynchronously scan of the sub topology.

Parameters	Description
<a href="#">invokeld</a>	Unique identifier for the request

#### Return Value

Return Value	Description
TRUE	Request of sub-topology scan accepted.
FALSE	The operation failed.

#### Behavior

Requests the scan of the sub topology. If the scan is finished, the result will be provided via [IDtmEvents::OnScanResponse\(\)](#)

#### Comments

This method is depreciated and should only be supported if running in an FDT 1.2 environment.

### 4.5.4.4 ValidateAddChild

```
HRESULT ValidateAddChild(
    [in] BSTR childSystemTag,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Validates if a given device, specified by its [systemTag](#), can be added to the sub-topology

Parameters	Description
<a href="#">childSystemTag</a>	SystemTag of to the child DTM

**Return Value**

Return Value	Description
TRUE	Topology valid
FALSE	Topology invalid

**Behavior**

The channel validates the connection. If the connection is valid it will return TRUE.

If the DTM needs more information about the child it can get the DTM via [IFdtTopology::GetDtmForSystemTag\(\)](#) passing the received [systemTag](#).

**Comments**

Device-DTMs with more than one required protocol should include busCategory information in the <BusInformation> element. .

- If a 1.2.1. DTM is connected to a 1.2 communication channel,
  - An 1.2.1 Frame Application has to consider:
    - the first BusInformation element in the DtmParameter document of the DTM is regarded as primary protocol
    - the Frame Application needs to check if the primary protocol of the DTM is supported by the communication channel
  - An 1.2 Frame Application:
    - The DTM can support only one protocol, because the old schemas supports only one BusInformation element

See also chapter 4.5.4.1

### 4.5.4.5 ValidateRemoveChild

```
HRESULT ValidateRemoveChild(
    [in] BSTR childSystemTag,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

Validates if a given device, specified by its [systemTag](#), can be removed from the sub-topology

Parameters	Description
<a href="#">childSystemTag</a>	SystemTag of the child DTM

**Return Value**

Return Value	Description
TRUE	Topology valid
FALSE	Topology invalid

**Behavior**

The channel has to validate if the device can be removed from the topology.

If the DTM needs more information about the child it can get the DTM via [IFdtTopology::GetDtmForSystemTag\(\)](#) passing the received [systemTag](#).

**Comments**

The validation must include a check for active connection and return FALSE if a connection is active via this channel.

## 4.5.5 Interface IFdtChannelSubTopology2

This mandatory interface is implemented by a communication channel and extends the interface IFdtChannelSubTopology by new methods

This interface supports network topology management functions for address setting. The interface is used to ask a DTM communication channel to set the fieldbus address of a single DTM or DTMs of a subtopology.

### 4.5.5.1 SetChildrenAddresses

```
HRESULT SetChildrenAddresses (
    [in] FdtXmlDocument dtmDeviceList,
    [out, retval] FdtXmlDocument* result);
```

**Description**

Requests bus address setting for specified device list.

Parameters	Description
dtmDeviceList	XML according <a href="#">DTMDeviceListSchema</a> defining device instances where to set the address.

**Return Value**

Return Value	Description
Result	XML document containing result of address setting. <a href="#">DTMDeviceListSchema</a>

**Behavior**

Requests setting of bus address for one device or a list of devices via IDtmParameter interface of the

corresponding child DTMs. The request may specify that the called communication channel should open a user interface to request device address settings from user. To get a qualified response, error information is included in the returned document.

## Comments

As part of executing this method the method [IFdtActiveX2:OpenDialogActiveXControlRequest\(\)](#) method may be used to request address selection from the user.

Setting the bus address on a Device-DTM via the IDtmParameter interface provides the Device-DTM with information. The Device-DTM must not use this information for execution of communication transactions, that set the address in the field device.

## 4.5.6 Interface IFdtChannelScan

This interface defines methods, which replace scan related methods of existing IFdtChannelSubTopology interface. If a communication channel supports FDT1.2.1 and runs in an environment that supports FDT1.2.1, the channel object can rely on that the methods of this interface are used instead of the replaced methods of the IFdtChannelSubTopology interface.

### 4.5.6.1 ScanRequest

```
HRESULT ScanRequest(
    [in] FdtUUIDString invokeld,
    [in] FdtXmlDocument request,
    [out, retval] VARIANT_BOOL* result);
```

## Description

Requests the asynchronously scan of the sub topology.

Parameters	Description
invokeld	Unique identifier for the request
Request	Information about the address range(s) to scan. <a href="#">FDTScanRequestSchema</a>

## Return Value

Return Value	Description
TRUE	Request of sub-topology scan accepted.
FALSE	The operation failed. <a href="#">IDtmEvents:OnErrorMessage()</a> assumed.

## Behavior

This method requests the scan of the sub topology. More than one scan response (provisional, final or error) can be returned via [IDtmScanEvents::OnScanResponse\(\)](#)

**Comments**

The ability to provide provisional scan responses is intended especially for “slow” fieldbus protocols. The operator will see how the life list is growing. It is possible to stop the scan, if the operator received enough information.

**4.5.6.2 CancelAction**

```
HRESULT CancelAction (
    [in] FdtUUIDString invokeID,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

Cancels an active asynchronous scan request identified by its invoke ID.

Parameters	Description
<a href="#">invokeID</a>	Unique identification of scan request.

**Return Value**

Return Value	Description
TRUE	Cancel of scan request accepted.
FALSE	Cancel of scan request not accepted.

**Behavior**

The Frame Application calls this method to cancel active scan request operation. If the channel has accepted the cancel action request it will return TRUE.

The channel may not be able to cancel the action immediately after accepting the request, but it should do it as soon as possible. In this case the channel will not fire the [IDtmScanEvents::OnScanResponse\(\)](#) event.

If the channel can't cancel the operation FALSE is returned. [IDtmScanEvents::OnScanResponse\(\)](#) event is fired when the operation has finished.

**Comments**

CancelAction does not automatically invalidate the provisional scan results, but stops retrieving additional information. The communication channel should stop the scan process.

**4.5.7 Interface IFdtFunctionBlockData**

This interface must be supported by DTMs for failsafe devices

The root communication component of an FDT system defines whether the system can provide failsafe access or not.

In a system that is able to provide failsafe access, it is important that all communication components support failsafe access. Communication channels have to support the propagation of the failsafe function calls. That is why the interface is mandatory for all channels with communication functionality. If the

represented gateway device does not provide failsafe functionality itself, it is sufficient to pass the function call to the underlying communication.

Gateway-DTMs and Device-DTMs have to be aware that the underlying communication might not support the interface.

The interface is part of the communication channel of a bus-master-DTM to allow an extendable topology. The DTM that manages the failsafe function blocks (FB) would be part of the Frame Application and would be DCS specific.

Communication channels have to do the propagation of the failsafe function calls. That is why this interface is mandatory for all channels with gateway functionality.

The interface to the bus master comprises two calls. The first one causes the Frame Application to open up a browser displaying the failsafe host and the available device function blocks. The user may select one that will be associated (assigned) with the DTM.

The second call provides the individual device parameter block packed within an XML frame. The XML frame provides information about the bus master and the device FB.

#### 4.5.7.1 GetFBInstanceData

```
HRESULT GetFBInstanceData(
    [in] BSTR* systemTag,
    [out, retval] FdtXmlDocument* result);
```

##### Description

This method is used by DTMs of failsafe devices to verify the consistency of device parameters.

The user can compare device parameters which are uploaded directly from the device with device parameters which are read indirectly from the device via the Proxy FB, located in the bus master.

Returns a static XML-document containing the parameters of the failsafe device

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

##### Return Codes

Return Code	Description
result	XML document containing the parameters of the failsafe device specified by the <a href="#">FDTFailSafeDataSchema</a>

##### Behavior

The channel object routes upward in the topology to the bus master driver that contains the corresponding Proxy FB of the DTM, reads the individual device parameter set directly out of the bus master and passes them to the DTM.

If there is still no assignment of the DTM to a Proxy FB, at first it executes the browsing function as it is described for [SelectFBInstance](#) ()

If a Communication-DTM is not able support failsafe functionality, it will return a valid xmlDocument that contains no "FDTFailSafeData" element.

**Comments**

-/-

**4.5.7.2 SelectFBInstance**

```
HRESULT SelectFBInstance(
    [in] BSTR* systemTag,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

Before a DTM of a failsafe device can verify the consistency of the device parameters, the user has to assign the Proxy FB in the host (bus master) to the DTM which contains the parameters of the failsafe device.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

**Return Codes**

Return Code	Description
TRUE	Function block associated.
FALSE	No function block associated

**Behavior**

Opens a browser which offers all available Proxy FBs that contain individual device parameter of devices. The user has to select the Proxy FB of the corresponding failsafe device for the DTM. The bus-master-DTM stores the assignment of the Proxy FB to the calling DTM-Instance.

The mechanism of assigning a DTM to a failsafe device is defined in more detail in section “4.5.3 F Parameter Assignment Paths” of the document “PROFIBUS Profile for Safety Technology, Version 1.20, 23-Oct-2002” .

If a Communication-DTM is not able to support failsafe functionality, it will return a “false” success

**Comments**

-/-

## 4.6 FDT Channel ActiveXControl

### 4.6.1 Interface IFdtChannelActiveXControl

This interface is an extension of a standard ActiveX control and allows connecting a channel object with the ActiveX control.

#### 4.6.1.1 Init

```
HRESULT Init(
    [in] FdtUUIDString invokeld,
    [in] IFdtChannel* channel,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Sets the callback pointer of an ActiveX control to the according FdtChannel.

Parameters	Description
invokeld	Identifier for the started application.
channel	Pointer to the channel business object

#### Return Value

Return Value	Description
TRUE	Channel initialized
FALSE	The operation failed.

#### Behavior

Sets the callback pointer of an ActiveX control to the according FdtChannel.

If the initialization returns FALSE, the Frame Application has to release the control.

The invoke id is used by a Frame Application for the association at the callback interface if the control is terminated within the user interface of the DTM. (see [IDtmEvents::OnApplicationClosed\(\)](#)).

Furthermore it allows the Frame Application to handle a list of open user interfaces.

#### Comments

-/-



### 4.6.1.2 PrepareToRelease

HRESULT [PrepareToRelease](#)(  
[out, retval] VARIANT\_BOOL\* result);

#### Description

Used to inform the channel control that it has to release its links to other components. The control will be released by the Frame Application after the DTM has send [IDtmEvents::OnApplicationClosed](#)().

Parameters	Description
------------	-------------

#### Return Value

Return Value	Description
TRUE	The request was accepted
FALSE	The operation failed.

#### Behavior

Releases the callback pointer of an ActiveX control to the according channel set during [Init](#)(). Furthermore the control has to release all links to other components and has to terminate all pending or running functions. The DTM has to inform the Frame Application via [IDtmEvents::OnApplicationClosed](#)() that the user interface could be released.

#### Comments

-/-

## 4.6.2 Interface IFdtChannelActiveXControl2

This interface extends the interface IFdtChannelActiveXControl by new methods. This interface is mandatory.

### 4.6.2.1 Init2

HRESULT [Init2](#)(  
[in] [FdtUUIDString](#) invokeld,  
[in] [FdtXmlDocument](#) functionCall,  
[in] IFdtChannel\* channel,  
[out, retval] VARIANT\_BOOL\* result);

#### Description

Sets the callback pointer of an ActiveX control to the according FdtChannel.

Parameters	Description
invokeld	Identifier for the started application.
functionCall	XML document containing the function id for the requested function or user interface specified by the <a href="#">DTMFunctionCallSchema</a>
Channel	Pointer to the channel business object

**Return Value**

Return Value	Description
TRUE	Channel initialized
FALSE	The operation failed.

**Behavior**

Sets the callback pointer of an ActiveX control to the according FdtChannel. The functionCall document informs the instance of the ActiveXControl about the context, it is started.

If the initialization returns FALSE, the Frame Application has to release the control.

The invoke id is used by a Frame Application for the association at the callback interface if the control is terminated within the user interface of the DTM. (see [IDtmEvents::OnApplicationClosed\(\)](#)). Furthermore it allows the Frame Application to handle a list of open user interfaces.

**Comments**

This function replaces the former [IFdtChannelActiveXControl::Init\(\)](#) function that did not provide the current functionCall document to the channel ActiveX control.

A Frame Application according to FDT version 1.2.1 must use this method instead of the former method.

## 4.7 Block Type Manager

The BTM follows the rules specified for the DTM. A set of new interfaces is defined to replace the corresponding DTM interfaces and to be used with the BTM. The new interfaces follow the specification for the corresponding interfaces for the DTM, but are applicable to the BTM and are using the schemas specified for the BTM. These interfaces are:

- IBtm
- IBtmInformation
- IBtmParameter

### 4.7.1 Interface IBtm

This interface is the main interface of a BTM. IBtm methods have the same behavior as specified with the interface IDtm. The only difference is that the methods are applied on a block type object (not on a device). The same methods are used for a DTM and for a BTM and the corresponding XML schemas definitions reflect the differences between the Block and the Device Type Manager objects.

For the IBtm interface, the only difference is in the schemas used in [IDtm::InitNew\(\)](#) method.

#### 4.7.1.1 Config

For description of this method refer to the method [IDtm::Config\(\)](#)..

#### 4.7.1.2 Environment

For description of this method refer to the method [IDtm::Environment\(\)](#).

#### 4.7.1.3 GetFunctions

For description of this method refer to the method [IDtm::GetFunctions\(\)](#).

#### 4.7.1.4 InitNew

The deviceType parameter changes as follows:

Parameters	Description
deviceType	XML document containing the manufacturer specific data like unique identifier for a block type specified by <a href="#">BtmInitSchema</a>

For description of the method refer to [IDtm::InitNew\(\)](#).

#### 4.7.1.5 InvokeFunctionRequest

For description of this method refer to the method [IDtm::InvokeFunctionRequest\(\)](#).

#### 4.7.1.6 PrepareToDelete

For description of this method refer to the method [IDtm::PrepareToDelete\(\)](#).

#### 4.7.1.7 PrepareToRelease

For description of this method refer to the method [IDtm::PrepareToRelease\(\)](#).

#### 4.7.1.8 PrepareToReleaseCommunication

For description of this method refer to the method [IDtm::PrepareToReleaseCommunication\(\)](#).

#### 4.7.1.9 PrivateDialogEnabled

For description of this method refer to the method [IDtm::PrivateDialogEnabled\(\)](#).

#### 4.7.1.10 ReleaseCommunication

For description of this method refer to method [IDtm::ReleaseCommunication\(\)](#).

#### 4.7.1.11 SetCommunication

For description of this method refer to method [IDtm::SetCommunication\(\)](#).

#### 4.7.1.12 SetLanguage

For description of this method, refer to method [IDtm::SetLanguage\(\)](#).

### 4.7.2 Interface IBtmInformation

IBtmInformation methods have the same behavior as specified with the interface IDtmInformation. The only difference is that the GetInformation method is applied on a block type object (not on a device). The new XML schema definition reflects the differences between the Block and the Device Type Manager objects.

#### 4.7.2.1 GetInformation

The result parameter changes as follows:

Parameters	Description
result	XML document containing static BTM information specified by the <a href="#">BtmInformationSchema</a>

For description of the method refer to [IDtmInformation::GetInformation\(\)](#).

### 4.7.3 Interface IBtmParameter

This interface allows a Frame Application the access to BTM parameters. The IBtmParameter methods have the same behavior as specified with the interface IDtmParameter. The only difference is that the methods [IDtmParameter::GetParameters\(\)](#) and [IDtmParameter::SetParameters\(\)](#) are applied to a block type object (not to a device). The new XML schema definition reflects the differences between the Block and the Device Type Manager objects.

#### 4.7.3.1 GetParameters

The return parameter changes as follows:

Parameters	Description
result	XML document with the block type device specific parameters specified by the <a href="#">BtmParameterSchema</a>

For a description of the method refer to [IDtmParameter::GetParameters\(\)](#).

### 4.7.3.2 SetParameters

The input parameter xmlDocument changes as follows:

Parameters	Description
xmlDocument	XML document specified by the <a href="#">BtmParamterSchema</a> . This document details block type specific parameters.

For a description of the method refer to [IDtmParameter::SetParameters\(\)](#).

## 4.8 BTM ActiveXControl

### 4.8.1 Interface IBtmActiveXControl

This interface is an extension of a standard ActiveX control and allows connecting a BTM object with the ActiveX control.

#### 4.8.1.1 Init

```
HRESULT Init(
    [in] FdtUUIDString invokeld,
    [in] FdtXmlDocument functionCall,
    [in] IBtm* btm,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Set the callback pointer of an ActiveX control to the according BTM.

Parameters	Description
invokeld	This is a unique identifier for the started application.
functionCall	XML document containing the function id for the requested function or user interface specified by the <a href="#">DTMFunctionCallSchema</a>
Btm	Pointer to the BTM business object

#### Return Value

Return Value	Description
TRUE	The control is initialized
FALSE	The operation failed.

#### Behavior

Set the callback pointer of an ActiveX control to the according BTM.

For detailed description of the method refer to [IDtmActiveXControl::Init\(\)](#) but note that the ActiveX application is associated to the Block Type Manager.

## Comments

-/-

### 4.8.1.2 PrepareToRelease

For a description of this method, refer to the method [IDtmActiveXControl::PrepareToRelease\(\)](#) but note that the ActiveX control is associated to the Block Type Manager.

## 4.9 Frame Application

### 4.9.1 Interface IDtmEvents

This interface is the callback-interface for the DTMs.

#### 4.9.1.1 OnApplicationClosed

```
HRESULT OnApplicationClosed(
    [in] FdtUUIDString invokeld);
```

##### Description

Notification by a DTM, that its user interface identified by the invoke id is closed.

Parameters	Description
invokeld	Identifier of the closed application

##### Return Value

Return Value	Description
--------------	-------------

##### Behavior

Notification by a DTM, that the user interface of a DTM opened by [IDtmApplication::StartApplication\(\)](#) or embedded as ActiveX Control is closed. There is no difference whether the user interface was closed by a user action or via [IDtmApplication::ExitApplication\(\)](#), [IDtmActiveXControl::PrepareToRelease\(\)](#) or [IFdtChannelActiveXControl::PrepareToRelease\(\)](#).

##### Comments

The invokeld was set at the startup of an application or during the initialization of the ActiveX control

#### 4.9.1.2 OnDownloadFinished

```
HRESULT OnDownloadFinished(
    [in] FdtUUIDString invokeld,
    [in] VARIANT_BOOL success);
```

##### Description

Notification by a DTM, that the asynchronous download function call identified by the invoke id is finished.

Parameters	Description
invokeld	Identifier of the download request
success	Notification by a DTM, whether the asynchronously download function call <a href="#">IDtmOnlineParameter::DownloadRequest()</a> is successfully finished.

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

Notification by a DTM, whether the asynchronously download function call [IDtmOnlineParameter::DownloadRequest\(\)](#) is successfully finished.

**Comments**

-/-

### 4.9.1.3 OnErrorMessage

HRESULT [OnErrorMessage](#)(  
     [in] BSTR [systemTag](#),  
     [in] BSTR errorMessage);

**Description**

Notification by a DTM about errors during a function call.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
errorMessage	Human readable error message

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

The method is necessary if the DTM works without a user interface. If a DTM works without user interface it is not allowed to display any error message within own dialog windows.

It's up to the Frame Application to handle the information. In case of errors or warnings, the human readable string can be displayed in a dialog box of the Frame Application.

**Comments**

In order to give user helpful hints concerning errors, it is recommended to use this method in cases, where a function call failed and [IFdtDialog::UserDialog\(\)](#) not be used.



#### 4.9.1.4 OnFunctionChanged

HRESULT [OnFunctionChanged](#)(  
     [in] BSTR [systemTag](#));

##### Description

Notification of a DTM that the information about its current available additional functionality has changed

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

##### Return Value

Return Value	Description
--------------	-------------

##### Behavior

The method is used if the additional functionality depends on the configuration of a device. Via this method the DTM can inform the Frame Application to update its menus or function calls which reference on these extended functionality. The Frame Application gets the actual available functionality via [IDtm::GetFunctions\(\)](#).

##### Comments

-/-

#### 4.9.1.5 OnChannelFunctionChanged

HRESULT [OnChannelFunctionChanged](#)(  
     [in] BSTR [systemTag](#),  
     [in] FdtXPath channelPath);

##### Description

Notification of a DTM that the information about channel related functionality has changed.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
channelPath	Identifier of the channel

##### Return Value

Return Value	Description
--------------	-------------

##### Behavior

Via this method the DTM can inform the Frame Application to update its channel related menus which

reference on these functionality. The Frame Application gets the actual available functionality via [IFdtChannelActiveXInformation::GetChannelFunctions\(\)](#).

#### Comments

-/-

### 4.9.1.6 OnInvokedFunctionFinished

HRESULT [OnInvokedFunctionFinished](#)(  
     [in] [FdtUUIDString](#) [invokeld](#),  
     [in] VARIANT\_BOOL success);

#### Description

Notification by a DTM, that the asynchronously invoked function call identified by the invoke id is finished.

Parameters	Description
<a href="#">invokeld</a>	Identifier of the closed application
<a href="#">success</a>	TRUE if the operation is successfully finished

#### Return Value

Return Value	Description
--------------	-------------

#### Behavior

Notification by a DTM, whether the asynchronously invoked function call [IDtm::InvokeFunctionRequest\(\)](#) is successfully finished.

#### Comments

-/-

### 4.9.1.7 OnNavigation

HRESULT [OnNavigation](#)(  
     [in] BSTR [systemTag](#));

#### Description

A DTM sends an notification to cause the navigation to a Frame Application-specific application.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

If a DTM sends a navigation request, the Frame Application decides which application will be opened. For example, if a DTM is started for diagnostic and shows the current device status, the user may want to know, where the device is connected within the system topology. Therefore the DTM provides a menu item or button for changing the application. If the user selects such a 'navigation' element the DTM send the OnNavigate() event and the Frame Application can open the system topology tree.

In general, the Frame Application has started the DTM and checks the application context to decide which Frame Application specific application will be started to guarantee a unique navigation behavior for the user.

The Frame Application can identify the calling DTM by the [systemTag](#).

**Comments**

-/-

### 4.9.1.8 OnOnlineStateChanged

HRESULT [OnOnlineStateChanged](#)(  
     [in] BSTR [systemTag](#),  
     [in] VARIANT\_BOOL onlineState);

**Description**

A DTM sends an notification about its online state.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
onlineState	TRUE means that the DTM is currently online FALSE means that the DTM is offline

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

If a DTM has successfully established a connection to its device it sends this notification (onlineState=TRUE) to the Frame Application so that the Frame Application can visualize the online state of the DTM. After the DTM has released the connection it sends again the notification (onlineState=FALSE) so that the Frame Application can update its view.

**Comments**

The DTM is allowed to send this notification if state has not changed to inform Frame Application that it is still in the same state. This may for example happen if no connection can be established (onlineState=FALSE).

The Frame Application should ignore these additional notifications.

#### 4.9.1.9 OnParameterChanged

HRESULT [OnParameterChanged](#)(  
     [in] BSTR [systemTag](#),  
     [in] [FdtXmlDocument](#) parameter);

##### Description

In case of a multi-user environment, it can be necessary to inform the Frame Application about parameter changes.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
parameter	XML document containing the changed parameters

##### Return Value

Return Value	Description
--------------	-------------

##### Behavior

If a DTM has stored any changed data it has to call [IDtmEvents::OnParameterChanged\(\)](#) with an XML document containing the instance specific changes. The Frame Application has now to inform all DTMs which reference the same device instance. The Frame Application will send this XML document via [IFdtEvents::OnParameterChanged\(\)](#) to all those DTMs.

Furthermore the Frame Application will send a notification to the according parent DTM via [IFdtEvents::OnChildParameterChanged\(\)](#).

##### Comments

This notification could also be used by a Frame Application to trigger an update e.g. to visualize the topology information.

The parent DTM gets only a notification, because the XML document, exchanged via [OnParameterChanged\(\)](#), is DTM specific and cannot be interpreted by a parent DTM [IDtmParameter::GetParameters\(\)](#). A parent DTM, which receives such a notification, can update its child specific data by calling [GetParameters\(\)](#) at the child DTM.

#### 4.9.1.10 OnPreparedToRelease

HRESULT [OnPreparedToRelease](#)(  
     [in] BSTR [systemTag](#));

##### Description

A DTM sends an notification that it can be released.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

The DTM has released all references to other components. After the Frame Application has received this notification it can release the DTM.

**Comments**

-/-

### 4.9.1.11 OnPreparedToReleaseCommunication

HRESULT [OnPreparedToReleaseCommunication](#)(  
     [in] BSTR [systemTag](#));

**Description**

A DTM sends an notification that its communication pointer can be released.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

The DTM has released all references of the communication pointer set during [IDtm::SetCommunication\(\)](#). After the Frame Application has received this notification it can call [IDtm::ReleaseCommunication\(\)](#) and release the communication pointer itself.

**Comments**

-/-

### 4.9.1.12 OnPrint

HRESULT [OnPrint](#)(  
     [in] BSTR [systemTag](#),  
     [in] [FdtXmlDocument functionCall](#));

**Description**

A DTM sends an notification that it wants to print a DTM specific document.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
functionCall	XML document containing the DTM specific function id for the requested document specified by the <a href="#">DTMFunctionCallSchema</a>

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

The method is used if a DTM wants to print its specific documentation. Therefore it sends via [OnPrint\(\)](#) a request to the Frame Application with a function id which identifies the DTM-specific document. Now the Frame Application can receive this document via [IDtmDocumentation::GetDocumentation\(\)](#) and send it to the environment-specific printer.

**Comments**

-/-

### 4.9.1.13 OnProgress

```
HRESULT OnProgress(
    [in] BSTR systemTag,
    [in] BSTR title,
    [in] short percent,
    [in] VARIANT_BOOL show);
```

**Description**

A DTM sends a notification about the progress on handling of a function call.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
title	Description of the running process
percent	State of progress 0..100%
show	Set to TRUE, if the progress should be displayed, otherwise the progress would not be shown and an open progress bar must be closed within the Frame Application

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

This method should be used by DTMs during functions, which may take a longer time, to inform the Frame Application and at least the user about ongoing activities. If a DTM cannot determine the real

progress, it can be useful to change e.g. the title.

It's up to the Frame Application how to handle the information.

#### Comments

-/-

### 4.9.1.14 OnScanResponse

```
HRESULT OnScanResponse(
    [in] FdtUUIDString invokeld,
    [in] FdtXmlDocument response);
```

#### Description

Returns a list of fieldbus related information to identify the connected devices.

Parameters	Description
invokeld	Unique identifier for the request
Response	XML document containing the result of the topology scan specified by the <a href="#">DTMTopologyScanSchema</a> .

#### Return Value

Return Value	Description
--------------	-------------

#### Behavior

Returns an XML document which contains a list of fieldbus related information to identify the connected devices. If no devices could be found the list will be empty.

#### Comments

Within FDT version 1.2.1 this method is obsolete. Only DTMs based on FDT version 1.2 are allowed to call this method.

### 4.9.1.15 OnUploadFinished

```
HRESULT OnUploadFinished(
    [in] FdtUUIDString invokeld,
    [in] VARIANT_BOOL success);
```

#### Description

Notification by a DTM, that the asynchronous upload function call identified by the invoke id is finished.

Parameters	Description
invokeld	Identifier of the upload request
success	Notification by a DTM, whether the asynchronously upload function call <a href="#">IDtmOnlineParameter::UploadRequest()</a> is successfully finished.

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

Notification by a DTM, whether the asynchronously upload function call [IDtmOnlineParameter::UploadRequest\(\)](#) is successfully finished.

**Comments**

-/-

## 4.9.2 Interface IDtmEvents2

This interface is the callback-interface for DTMs supporting FDT version 1.2.1 or higher version. This interface extends the interface IDtmEvents by a new method. This interface is mandatory.

A DTM supporting 1.2.1 or higher version must call IDtmEvents2, if the Frame Application supports this interface. Instead of calling [IDtmEvents::OnOnlineStateChanged\(\)](#) such a DTM then must use the [IDtmEvents2::OnStateChanged\(\)](#) method.

### 4.9.2.1 OnStateChanged

```
HRESULT OnStateChanged(
    [in] BSTR systemTag,
    [in] FdtXmlDocument xmlDoc);
```

**Description**

A DTM sends a notification about change in its state.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
xmlDoc	XML document containing the function id for the requested function or user interface specified by the <a href="#">DTMStateSchema</a>



## Return Value

Return Value	Description
--------------	-------------

## Behavior

A DTM has performed a state transitions according to the DTM state machine between one of the following states:

communication-set  
going-online  
going-offline  
online

If the transition was triggered by an error condition e.g., ConnectResponse failed or OnAbort, the fdt:CommunicationError information must be provided within the XML document.

## Comments

- / -

## 4.9.3 Interface IDtmScanEvents

This interface defines callback methods, called by DTMs when returning scan information.

### 4.9.3.1 OnScanResponse

HRESULT [OnScanResponse](#) (  
    [in] [FdtUUIDString](#) *invokeld*,  
    [in] [FdtXmlDocument](#) response);

## Description

Returns a list of field bus related information to identify the connected devices.

Parameters	Description
<a href="#">invokedID</a>	Unique identification of scan request.

**Return Value**

Return Value	Description
response	XML document containing the result of the topology scan as specified by a field bus specific schema ( <a href="#">FDTxxxScanIdentSchema</a> ) . The fieldbus specific document must be transformed using protocol specific xsl to get a protocol independent document, which must validate against <a href="#">DTMScanIdentSchema</a>

**Behavior**

The communication channel calls this method to return provisional results, final scan request results or error information.

**Comments**

Information regarding the type of response (provisional, final or error) is part of the response document.

**4.9.3.2 OnScanHardwareResponse**

HRESULT [OnScanHardwareResponse](#) (  
     [in] [FdtUUIDString](#) invokeID,  
     [in] [FdtXmlDocument](#) response);

**Description**

Notification by a DTM, that asynchronous scan hardware request operation has finished.

Parameters	Description
invokeID	Identifier of the request.
Response	XML document containing the result of the scan hardware request specified by a field bus specific schema ( <a href="#">FDTxxxScanIdentSchema</a> ) if request was called at a Gateway- or Device-DTM. The fieldbus specific document containing additional manufacturer specific extensions and must be transformed using protocol specific xsl to get a protocol independent document according <a href="#">DTMScanIdentSchema</a> .  If request was called at a Communication-DTM, then XML document according <a href="#">DTMScanIdentSchema</a> is returned, which must not be transformed (ID entries which can not be filled are left empty)

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

Notification by a DTM, that asynchronous scan hardware request operation has finished. DTM passes

XML containing information about found hardware.

If the operation was called in context of a Communication-DTM, then the XML contains information of all responsive hardware entities (interface cards, modems...), which can be handled by this DTM device type.

If the operation was called in context of a Gateway-/Device-DTM, then the XML only contains information of single device where the DTM was started for.

The XML contains error information if the scan hardware request failed.

#### Comments

-/-

### 4.9.4 Interface IDtmAuditTrailEvents

This interface must be used by all DTMs to send their audit trail information to the Frame Application. It is up to the Frame Application to implement the audit-trail-application itself which records the device specific information and supplies the user interface.

#### 4.9.4.1 OnAuditTrailEvent

```
HRESULT OnAuditTrailEvent(
    [in] BSTR systemTag,
    [in] FdtXmlDocument logEntry);
```

#### Description

Notification by a DTM about changed data to be recorded by the Frame Application's audit trail tool.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
logEntry	XML document containing the changes specified by the <a href="#">DTMAuditTrailSchema</a>

#### Return Value

Return Value	Description
--------------	-------------

#### Behavior

A DTM must call this function at the Frame Application to add an entry to the audit trail record.

#### Comments

The content of the log-entry depends on the DTM. The implementation of the audit trail tool is Frame Application-specific.

#### 4.9.4.2 OnEndTransaction

```
HRESULT OnEndTransaction(
    [in] BSTR systemTag,
    [out, retval] VARIANT_BOOL* result);
```

##### Description

A DTM sends an notification to close the audit trail sequence.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

##### Return Value

Return Value	Description
TRUE	Audit trail session closed
FALSE	The operation failed.

##### Behavior

A DTM calls this function at the Frame Application if it wants to close the audit trail record opened by [IDtmAuditTrailEvents::OnStartTransaction\(\)](#).

##### Comments

When the record has been closed, the Frame Application may use it for its own specific audit trail functions like adding comments, time stamps etc.

#### 4.9.4.3 OnStartTransaction

```
HRESULT OnStartTransaction(
    [in] BSTR systemTag,
    [out, retval] VARIANT_BOOL* result);
```

##### Description

Notification by a DTM that the following changes should be recorded by the Frame Application's audit trail tool.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

**Return Value**

Return Value	Description
TRUE	The audit trail application allows to open a new session
FALSE	The operation failed.

**Behavior**

A DTM calls this function at the Frame Application to request audit trail for the following actions like configuration or simulation. All calls of [OnAuditTrailEvent\(\)](#) will be recorded until the record is closed by [IDtmAuditTrailEvents::OnEndTransaction\(\)](#).

**Comments**

-/-

## 4.9.5 Interface IFdtActiveX

This interface must be provided by a Frame Application that supports a GUI.

### 4.9.5.1 CloseActiveXControlRequest

```
HRESULT CloseActiveXControlRequest(
    [in] FdtUUIDString invokeld,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

A DTM sends a request to close one of its ActiveX controls.

Parameters	Description
invokeld	Identifier for the started ActiveX control.

**Return Value**

Return Value	Description
TRUE	The ActiveX control will be released by the Frame Application
FALSE	The operation failed.

**Behavior**

This method is used by a DTM if it wants to close an ActiveX user interface which was instantiated and embedded by the Frame Application. The Frame Application will release the link between the user interface and the DTM via [IDtmActiveXControl::PrepareToRelease\(\)](#).

**Comments**

-/-

### 4.9.5.2 OpenActiveXControlRequest

```
HRESULT OpenActiveXControlRequest(
    [in] BSTR systemTag,
    [in] FdtXmlDocument functionCall,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Request of a DTM to start a DTM functionality defined by the function call id

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
functionCall	XML document containing the DTM specific function id for the requested user interface specified by the <a href="#">DTMFunctionCallSchema</a>

#### Return Value

Return Value	Description
TRUE	The requested ActiveX control will be instantiated by the Frame Application
FALSE	The operation failed.

#### Behavior

This method is used by a DTM if it wants to open an ActiveX user interface which must be instantiated and embedded by the Frame Application. The Frame Application will establish the link between the new user interface and the DTM via [IDtmActiveXControl::Init\(\)](#).

#### Comments

The DTM has also take into account the additional information (application id and operation phase) passed via the XML document.

In general it is expected that the ActiveX behaves like a modeless dialog.

### 4.9.6 Interface IFdtActiveX2

This interface extends the interface IFdtActiveX by new methods. This interface is mandatory.

### 4.9.6.1 OpenDialogActiveXControlRequest

```
HRESULT OpenDialogActiveXControlRequest(
    [in] BSTR systemTag,
    [in] FdtXmlDocument functionCall,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Request to open ActiveX control for a certain DTM in a modal dialog of Frame Application.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
functionCall	XML document containing the DTM specific function id for the requested user interface specified by the <a href="#">DTMFunctionCallSchema</a>

#### Return Value

Return Value	Description
TRUE	The requested ActiveX control was instantiated by the Frame Application
FALSE	The operation failed (i.e. because Frame Application is running without user interface).

#### Behavior

This method is used to open an ActiveX user interface, which must be instantiated and embedded in a modal dialog by the Frame Application. The Frame Application will establish the link between the new user interface and the DTM via [IDtmActiveXControl::Init\(\)](#).

This method behaves always modal. Frame Application at least has to ensure that all ActiveX controls of calling DTM are disabled; no further user input is possible.

The opened ActiveX user interface must call [IFdtActiveX::CloseActiveXControlRequest\(\)](#) if it wants to be closed later. [OpenDialogActiveXControlRequest\(\)](#) works synchronously so that the DTM is block until ActiveX user interface and corresponding dialog are closed.

#### Comments

A DTM always should use this method for more complex dialogs rather than calling [IFdtDialog::UserDialog\(\)](#) with [FDTUserMessageSchema](#) XML containing variables to edit, because this is not supported by almost all Frame Applications. Call of [IFdtDialog::UserDialog\(\)](#) should be preferred if FDTUserMessage XML only contains text lines.

Caller of [OpenDialogActiveXControlRequest\(\)](#) have to be aware of some synchronization circumstances which arise if modal dialogs are opened. FDT 1.2 Addendum specification discusses this topic for [IFdtDialog::UserDialog\(\)](#) and [IDtmEvents::OnErrorMessage\(\)](#) calls (refer *FDT Best Practices – A closer look at message loops*).

### 4.9.6.2 OpenFileDialogChannelActiveXControlRequest

```
HRESULT OpenFileDialogChannelActiveXControlRequest(
    [in] BSTR channelPath,
    [in] FdtXmlDocument functionCall,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Request to open ActiveX control for a certain channel object (identified by the channelPath) in a modal dialog of Frame Application.

Parameters	Description
channelPath	Identifier of the channel instance (as returned by <a href="#">IFdtChannel::GetChannelPath()</a> )
functionCall	XML document containing the <b>ActiveX</b> function id for the requested user interface specified by the <a href="#">DTMFunctionCallSchema</a>

#### Return Value

Return Value	Description
TRUE	The requested ActiveX control was instantiated by the Frame Application
FALSE	The operation failed (i.e. because Frame Application is running without user interface).

#### Behavior

This method is used to open an ActiveX user interface, which must be instantiated and embedded in a modal dialog by the Frame Application. The Frame Application will establish the link between the new user interface and the channel via [IFdtChannelActiveXControl2::Init2\(\)](#).

This method behaves always modal. Frame Application at least has to ensure that all ActiveX controls of related DTM are disabled; no further user input is possible.

The opened ActiveX user interface must call [IFdtActiveX2::CloseChannelActiveXControlRequest\(\)](#) if it wants to be closed later. [OpenDialogChannelActiveXControlRequest\(\)](#) works synchronously so that the call is block until ActiveX user interface and corresponding dialog are closed.

#### Comments

This function should be used for more complex dialogs rather than calling [IFdtDialog::UserDialog\(\)](#) with [FDTUserMessageSchema](#) XML containing variables to edit, because this is not supported by almost all Frame Applications. Call of [IFdtDialog::UserDialog\(\)](#) should be preferred if [FDTUserMessage](#) XML only contains text lines.

Caller of [OpenDialogChannelActiveXControlRequest\(\)](#) have to be aware of some synchronization circumstances which arise if modal dialogs are opened. FDT 1.2 Addendum specification discusses this topic for [IFdtDialog::UserDialog\(\)](#) and [IDtmEvents::OnErrorMessage\(\)](#) calls (refer *FDT V 1.2 – Mai 2003 - Addendum*, chapter 2.30.4 – A closer look at message loops, paragraph DTM's point of view).



#### 4.9.6.3 CloseChannelActiveXControlRequest

```
HRESULT CloseChannelActiveXControlRequest(
    [in] FdtUUIDString invokeld,
    [out, retval] VARIANT_BOOL* result);
```

##### Description

A DTM sends a request to close one of its ActiveX controls.

Parameters	Description
invokeld	Identifier for the started ActiveX control.

##### Return Value

Return Value	Description
TRUE	The ActiveX control will be released by the Frame Application
FALSE	The operation failed.

##### Behavior

This method is used by a DTM if it wants to close an ActiveX user interface which was instantiated and embedded by the Frame Application. The Frame Application will release the link between the user interface and the channel object. This is done via [IFdtChannelActiveXControl::PrepareToRelease\(\)](#).

##### Comments

-/-

#### 4.9.6.4 OpenChannelActiveXControlRequest

```
HRESULT OpenChannelActiveXControlRequest(
    [in] BSTR channelPath,
    [in] FdtXmlDocument functionCall,
    [out, retval] VARIANT_BOOL* result);
```

##### Description

Request to start an ActiveX functionality defined by the function call id for a certain channel object (identified by the channelPath).

Parameters	Description
channelPath	Identifier of the channel instance (as returned by <a href="#">IFdtChannel::GetChannelPath()</a> )
functionCall	XML document containing the ActiveX function id for the requested user interface specified by the <a href="#">DTMFunctionCallSchema</a>

**Return Value**

Return Value	Description
TRUE	The requested ActiveX control will be instantiated by the Frame Application
FALSE	The operation failed.

**Behavior**

This method is used if the caller wants to open an ActiveX user interface which must be instantiated and embedded by the Frame Application. The Frame Application will establish the link between the new user interface and the related object via [IFdtChannelActiveXControl::Init\(\)](#).

**Comments**

The caller of this method has also to take into account the additional information (application id and operation phase) passed via the XML document.

The Frame creates the Channel-ActiveX and assigns it to the channel object (referenced by channelPath) by calling [IFdtChannelActiveXControl::Init\(\)](#).

## 4.9.7 Interface IFdtBulkData

The Bulk Data Interface offers DTMs the possibility, to store a big amount of additional data like protocols of measured values or historical data of configuration changes. The DTMs are able to do that in a private way.

If access to these data is not possible, there must be no impact to the instance specific configuration data set of a DTM. The instance data set of a DTM must be always consistent to the configuration data stored via the standard IPersistXXX interface. It is in the responsibility of the Frame Application to create a backup strategy for this type of data.

The interface handles bulks of data at a device level.

### 4.9.7.1 GetInstanceRelatedPath

```
HRESULT GetInstanceRelatedPath(
    [in] BSTR systemTag,
    [out, retval] BSTR* result);
```

**Description**

Returns the instance related path for bulk data.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

**Return Value**

Parameters	Description
Result	Instance related path (file system-directory) for bulk data including a trailing backslash

**Behavior**

The Frame Application offers a DTM a way to request an instance specific path to an area which could be used to store data in a DTM specific way. It is up to a DTM to decide in which way the data will be stored. It could be a binary format by using IStorage or PropertyBag or even an ASCII-File. The Frame Application is responsible to provide an unique path for each instance. It must be an absolute path to allow a DTM the direct access.

**Comments**

A DTM must work without any side effects if a path is not available.  
 A DTM must clean up the area specified by the instance related path if [IDtm::PrepareToDelete\(\)](#) is called.  
 There is no FDT specific locking mechanism, so the DTM is responsible for consistence data.

**4.9.7.2 GetProjectRelatedPath**

```
HRESULT GetProjectRelatedPath(
    [in] BSTR systemTag,
    [out, retval] BSTR* result);
```

**Description**

Returns the project related path for bulk data. Returns a unique file system path (directory) for any combination of project and DTM type (e.g. it returns different paths for the same DTM type within two projects).

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

**Return Value**

Parameters	Description
result	Project related path (directory) for bulk data including a trailing backslash

## Behavior

The Frame Application offers a DTM a way to request a project specific path to an area which could be used to store data in a DTM specific way. It is up to a DTM to decide in which way the data will be stored. It could be a binary format by using IStorage or PropertyBag or even an ASCII-File.

The Frame Application is responsible to provide an unique path for each DTM type within a project. It must be an absolute path to allow a DTM the direct access.

## Comments

A DTM must work without any side effects if a path is not available.

If the DTM holds any references between project and instance related data it must clean up these data if [IDtm::PrepareToDelete\(\)](#) is called.

There is no FDT specific locking mechanism, so the DTM is responsible for consistence data.

## 4.9.8 Interface IFdtContainer

This is the main interface of the Frame Application. It supports the functions for the instance data management like locking within a multi user system.

### 4.9.8.1 GetXmlSchemaPath

HRESULT [GetXmlSchemaPath](#) (  
[out, retval] BSTR\* result);

## Description

Returns a path where the default schemas are stored

Parameters	Description
------------	-------------

## Return Value

Parameters	Description
Result	Path to the default schemas including a trailing backslash

## Behavior

This function returns the file system path to the FDT default XML schemas

## Comments

-/-

### 4.9.8.2 LockDataSet

```
HRESULT LockDataSet(
    [in] BSTR systemTag,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

A DTM sends an notification to the Frame Application that it wants to have exclusive write access for the currently loaded data set.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

#### Return Value

Return Value	Description
TRUE	Data set is locked for write access
FALSE	Data set could not be locked. DTM has read access only

#### Behavior

Via this method a DTM notifies the database that it wants to modify or delete the specified instance data set. It is up to the Frame Application to validate this request within a multi-user multi-session system. If the request fails, the DTM must not change any data and should set all input fields to 'non edit' in case of an open user interface.

It is in the responsibility of the Frame Application to reject write-requests if a DTM does not take care about its read only status.

A DTM must not lock it's instance data for the complete lifetime. Instead the DTM should try to lock data only if instance data is going to be modified and should unlock the data after the instance data is saved and no further modifications are expected.

#### Comments

Within a single user system the method returns always TRUE.

Examples for lock conditions are:

- a user interface is active, that allows changing the instance data
- the DTM received the request to change data via its COM-interfaces (IDtmParameter, IDtmInstanceData)

### 4.9.8.3 SaveRequest

```
HRESULT SaveRequest(
    [in] BSTR systemTag,
    [out, retval] VARIANT_BOOL* result);
```

#### Description

Informs the Frame Application that it should store the changed data.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

**Return Value**

Return Value	Description
TRUE	Data set will be saved
FALSE	The operation failed.

**Behavior**

Via this method a DTM notifies the Frame Application that it wants to save its data. It is up to the Frame Application to store the data.

The Frame Application get the data it has to store via the standard storage interfaces.

Transient data remains in transient state until the Frame Application successfully completes IPersistXXX:Save().

**Comments**

This method is the only method to inform the Frame Application that it should store the changed data. Even if the IPersistXXX::IsDirty property is available, it will not be used by a Frame Application. The Frame Application could also initiate the persistence interface of a DTM by itself.

Concerning multi user access the Frame Application must reject the save request if the DTM has no write access rights.

**4.9.8.4 UnlockDataSet**

```
HRESULT UnlockDataSet(
    [in] BSTR systemTag,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

Notification to the Frame Application that the DTM wants to unlock a data set and needs only read access for the currently loaded data set.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

**Return Value**

Return Value	Description
TRUE	Data set is unlocked
FALSE	Data set is still locked

**Behavior**

Via this method a DTM notifies the Frame Application that it has finished the modification of the instance data set. It is up to the Frame Application to manage the notification of all dependent components, for example via [IFdtEvents::OnParameterChanged\(\)](#) within a multi-user multi-session system.

If the request fails, the DTM should notify the Frame Application via [IDtmEvents:OnErrorMessage\(\)](#) to cause a system administrator to clean up the database.

A DTM must not lock it's instance data for its complete lifetime. Instead the DTM should try to lock data only if instance data is going to be modified and should unlock the data after the instance data is saved and no further modifications are expected.

**Comments**

Within a single user system the method returns always TRUE.

## 4.9.9 Interface IFdtDialog

This interface provides a functionality which allows a DTM to display messages like error, warning, and information.

### 4.9.9.1 UserDialog

```
HRESULT UserDialog(
    [in] BSTR systemTag,
    [in] FdtXmlDocument userMessage,
    [out, retval] FdtXmlDocument* result);
```

**Description**

Call the Container to display a message

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
userMessage	XML document according to the message specified by the <a href="#">FDTUserMessageSchema</a>

**Return Value**

Return Value	Description
result	XML document according to the user action specified by the

	<a href="#">FDTUserMessageSchema</a>
--	--------------------------------------

**Behavior**

A DTM should always use this method for standard user dialogs like error or information messages. Especially if a DTM is not allowed to open a user dialog (see [IDtm::PrivateDialogEnabled\(\)](#)) this method is called to instruct the Frame Application to open it. The method will return the selection of the user action or the specified default answer of the dialog.

It is up to the Frame Application to open a dialog or to send the default answer.

The Frame Application should answer within a proper time-space, because the method works synchronously so that the DTM is blocked until it receives the answer.

In case of a distributed system the Frame Application must ensure displaying the user dialog and the user interface of the DTM at the same workplace.

**Comments**

-/-

## 4.9.10 Interface IFdtTopology

This interface provides the access to the complete system topology. A Frame Application has always to configure the sub-topology of a channel via the interface [IFdtChannelSubTopology](#), so that the channel or at least the according DTM can validate the connections.

### 4.9.10.1 CreateChild

```
HRESULT CreateChild(
    [in] FdtXmlDocument deviceType,
    [in] FdtXPath channelPath,
    [out, retval] BSTR* result);
```

**Description**

A DTM sends a request to the Frame Application to create a new instance data set of the specified device type.

Parameters	Description
deviceType	XML document containing the information specified by <a href="#">DTMInitSchema</a>
channelPath	Specifies the channel path of the parent DTM to which the newly created instance data set should be placed



**Return Value**

Return Value	Description
result	System tag of the DTM. If the operation failed, a NULL pointer will be returned

**Behavior**

Returns the system tag of the DTM which is newly created. The DTM is instantiated by the Frame Application. If the operation failed, a NULL Pointer will be returned. The Frame Application has to implement the behavior described in chapter 6.13.1 Instantiation of a New DTM. It is also in the responsibility of the Frame Application to insert the created DTM into the topology.

**Comments**

-/-

## 4.9.10.2 DeleteChild

```
HRESULT DeleteChild(
    [in] BSTR systemTag,
    [in] FdtXPath channelPath,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

Remove the DTM specified by [systemTag](#) from the topology identified by channelPath. If this was the last reference within the topology, remove the instance data set

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device to remove
channelPath	Path of channel of parent

**Return Value**

Return Value	Description
TRUE	Operation succeeded
FALSE	Operation Failed

**Behavior**

Remove the DTM specified by [systemTag](#) from the topology identified by channelPath. Therefore the Frame Application has to call [ValidateRemoveChild\(\)](#). If this was the last reference within the topology, the Frame Application has to delete the instance data set. The Frame Application has to call [IDtm::PrepareToDelete\(\)](#) with respect of the behavior. The operation will also fail, if a sub topology exists.

**Comments**

-/-

### 4.9.10.3 GetChildNodes

```
HRESULT GetChildNodes(
    [in] BSTR systemTag,
    [in] FdtXPath channelPath,
    [out, retval] FdtXmlDocument* result);
```

#### Description

Returns an XML document containing the [systemTags](#) of all child DTMs of the DTM identified by its system tag and the channel path.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance
channelPath	Identifier of the channel

#### Return Value

Return Value	Description
result	XML document containing information according to the definition of <a href="#">DTMSystemTagListSchema</a>

#### Behavior

Returns an XML document containing the systemTags of all child DTMs of the DTM identified by its system tag and the channel path.

The topology information is globally accessible for all DTMs.

#### Comments

Only a Frame Application that supports Nested Communication has to implement this method.

### 4.9.10.4 GetDtmForSystemTag

```
HRESULT GetDtmForSystemTag(
    [in] BSTR systemTag,
    [out, retval] IDtm* result);
```

#### Description

Return the associated DTM according the given system tag

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device

**Return Value**

Return Value	Description
result	Pointer to a DTM

**Behavior**

Return the associated DTM according the given system tag. Additional calls within a Frame Application instance must return the identical interface pointer.

The Frame Application has to implement a kind of reference counting which is required to handle multiple references to the same DTM instance. The caller has to call [ReleaseDtmForSystemTag\(\)](#) to release the reference.

**Comments**

-/-

**4.9.10.5 GetDtmInfoList**

HRESULT [GetDtmInfoList](#)(  
[out, retval] [FdtXmlDocument](#) \* result);

**Description**

Returns an XML-document containing a list of DTM related information. This information is provided via the DtmInfo-structure defined within the [DTMInformationSchema](#).

Parameters	Description
------------	-------------

**Return Value**

Return Value	Description
Result	List of DtmInfo according the <a href="#">DTMInfoListSchema</a>

**Behavior**

Returns an XML-document containing a list of DTM related information. This information could be used to create an XML document of type DTMInitSchema according the usage of CreateDtmInstance()

**Comments**

It is up to the Frame Application to decide which DTM information will be available via the list. E.g. the list could contain only information concerning HART devices even if PROFIBUS devices are installed.

#### 4.9.10.6 GetParentNodes

```
HRESULT GetParentNodes(
    [in] BSTR systemTag,
    [out, retval] FdtXmlDocument* result);
```

##### Description

Returns an XML-document containing a list of system tags of all parent DTMs of the DTM identified by its system tag.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

##### Return Value

Return Value	Description
result	XML document containing a list of system tags <a href="#">DTMSystemTagListSchema</a>

##### Behavior

Returns a list of system tags of parent DTMs. The topology information is globally accessible for all DTMs.

##### Comments

Only a Frame Application that supports Nested Communication has to implement this interface.

#### 4.9.10.7 MoveChild

```
HRESULT MoveChild(
    [in] BSTR systemTag,
    [in] FdtXPath destinationChannelPath,
    [out, retval] VARIANT_BOOL* result);
```

##### Description

Move a DTM defined by [systemTag](#) from the current position within the topology to the position defined by destinationChannelPath. The complete sub topology will be moved.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device to move
destinationChannelPath	Path of channel of destination parent

**Return Value**

Return Value	Description
TRUE	Data Set moved
FALSE	Operation failed

**Behavior**

Moves the instance data set related to the device which is identified by the [systemTag](#).  
The Frame Application has to call [OnRemoveChild\(\)](#) and [OnAddChild\(\)](#).

**Comments**

-/-

### 4.9.10.8 ReleaseDtmForSystemTag

```
HRESULT ReleaseDtmForSystemTag(
    [in] BSTR systemTag,
    [out, retval] VARIANT_BOOL* result);
```

**Description**

Release the associated DTM according the given system tag

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device to release

**Return Value**

Return Value	Description
TRUE	The operation succeeded
FALSE	The operation failed.

**Behavior**

Release the reference to the associated DTM according the given system tag. This method is used only in combination with [GetDtmForSystemTag\(\)](#).

**Comments**

-/-

### 4.9.11 Interface IDtmRedundancyEvents

This Frame Application interface provides the access for parent components handling redundant slaves. A Frame Application not implementing this interface is not able to display redundancy information within it's topology information. (for redundancy refer chapter 4.10.7)

#### 4.9.11.1 OnAddedRedundantChild

```
HRESULT OnAddedRedundantChild(
    [in] BSTR systemTag,
    [in] FdtXPath channelPath,
    [out, retval] VARIANT_BOOL* result);
```

##### Description

A parent component sends this event to the Frame Application if a Device-DTM handling a redundant device is added to the topology. The Frame Application is then able to display the instance at an additional redundant communication channel.

Parameters	Description
systemTag	Identifier of the Device-DTM instance representing a redundant slave
channelPath	Specifies the redundant channel path of the parent DTM to which the DTM is connected

##### Return Value

Return Value	Description
TRUE	Operation succeeded
FALSE	Operation Failed

##### Behavior

The parent component adds the Device-DTM specified by [systemTag](#), handling a redundant slave, to the communication channel identified by channelPath. The Frame Application must not call [ValidateAddChild\(\)](#) or [OnAddChild\(\)](#) on this channel.

##### Comments

-/-

#### 4.9.11.2 OnRemovedRedundantChild

```
HRESULT OnRemovedRedundantChild(
    [in] BSTR systemTag,
    [in] FdtXPath channelPath,
    [out, retval] VARIANT_BOOL* result);
```

##### Description

A parent component sends this event to the Frame Application if a Device-DTM handling a redundant device is removed from the topology. The Frame Application is able to hide the instance at the additional redundant communication channel.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance representing a redundant slave
channelPath	Specifies the redundant channel path of the parent DTM to which the DTM is connected

**Return Value**

Return Value	Description
TRUE	Operation succeeded
FALSE	Operation Failed

**Behavior**

The parent component removes the Device-DTM specified by [systemTag](#), handling a redundant slave, from the communication channel identified by channelPath. The Frame Application must not call [ValidateRemoveChild\(\)](#) or [OnRemoveChild\(\)](#) on this channel.

**Comments**

-/-

## 4.9.12 Interface IDtmSingleDeviceDataAccessEvents

This interface is the callback interface for single device data access implemented by the Frame Application.

### 4.9.12.1 OnItemListResponse

```
HRESULT OnItemListResponse(
    [in] FdtUUIDString invokeId,
    [in] FdtXmlDocument response);
```

**Description**

Provides the response to [ItemListRequest\(\)](#) identified by the invoke id. ItemListResponse provides an XML document containing a list of the available device specific parameters and process values. Within a DTM this list may contain items related to configuration parameters, process values as well as asset management related data like stroke counter. In DTM state 'configured' the returned item list is based on the current instance data set, which could be different from the configuration of the device. In this case Read- and WriteRequests may fail.

Parameters	Description
invokeId	Unique identifier for the request
response	XML document containing a DtmItemInfoList with the actual

	available parameters specified by the <a href="#">DTMItemListSchema</a>
--	---

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

The method provides a list of items that can be read or written from/to the DTM via [ReadRequest\(\)](#) or written to the DTM via [WriteRequest\(\)](#). The source for this data is the device itself.

Items provided within these list may also be available as channel objects (provided by [IDtmChannel::GetChannels\(\)](#)) or modeled as an exported variable (DtmVariable provided by [IDtmParameter::GetParameters\(\)](#) or [IBtmParameter::GetParameters\(\)](#)). The related items can be identified via the attribute 'semanticId' (refer to chapter FDT Data Types)

**Comments**

The contents of the provided XML document may depend on the current configuration of the device. If the contents is changed, a DTM has to inform the Frame Application by sending [IDtmSingleDeviceDataAccess::OnItemListChanged\(\)](#)

**4.9.12.2 OnDeviceItemListChanged**

HRESULT [OnDeviceItemListChanged](#)(  
[in] BSTR [systemTag](#));

**Description**

The DTM informs the Frame Application that the content of the item list has been changed.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

Via this method a DTM informs the Frame Application that the content of the item list has changed (the available items in general, not the value). This may happen if the content of the list depends on the configuration of the device.

[OnItemListChanged](#) should not be fired in case of pending responses.



**Comments**

-/-

**4.9.12.3 OnReadResponse**

```
HRESULT OnReadResponse(  
    [in] FdtUUIDString invokeld,  
    [in] FdtXmlDocument response);
```

**Description**

Provides the response to [ReadRequest\(\)](#) identified by the invoke id.

Parameters	Description
invokeld	Unique identifier for the request
Response	Received data as DtmlItemList specified by the <a href="#">DTMLItemListSchema</a>

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

Via this method a Frame Application that sent the read-request, receives the requested data from the DTM.

**Comments**

-/-

**4.9.12.4 OnWriteResponse**

```
HRESULT OnWriteResponse(  
    [in] FdtUUIDString invokeld,  
    [in] FdtXmlDocument response);
```

**Description**

Provides the response to [WriteRequest\(\)](#) identified by the invoke id.

Parameters	Description
invokeld	Unique identifier for the request
Response	Received data as DtmlItemList that contains the device data of the successfully written data specified by the <a href="#">DTMItemListSchema</a> (may differ to the written value due to e. g. rounding procedures within the device)

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

Via this method a Frame Application may receive information from the DTM about the successfully written data.

**Comments**

-/-

### 4.9.13 Interface IDtmSingleInstanceDataAccessEvents

This interface is the callback interface for single instance data access implemented by the Frame Application.

#### 4.9.13.1 OnInstanceltemListChanged

HRESULT [OnInstanceltemListChanged](#)(  
[in] BSTR [systemTag](#));

**Description**

The DTM informs the Frame Application that the content of the item list has been changed.

Parameters	Description
<a href="#">systemTag</a>	Identifier of the device instance

**Return Value**

Return Value	Description
--------------	-------------

**Behavior**

Via this method a DTM informs the Frame Application that the content of the item list has changed (the available items in general, not the value). This may happen if the content of the list depends on the configuration of the device.

OnItemListChanged should not be fired in case of pending responses.

**Comments**

-/-

#### 4.9.14 Interface IFdtBtmTopology

This interface provides the access to the Block topology. IFdtBtmTopology methods have the same behavior as specified with the interface IFdtTopology. The only difference is that the methods are used to apply to a block type object (not to a device). The new XML schema definition reflects the differences between the Block and the Device Type Manager objects.

If a DTM has communication channels to support both - DTMs and BTMs, the IFdtBtmTopology interface will provide information only about BTMs. The information related to the DTM topology will be provided by the IFdtTopology interface.

##### 4.9.14.1 CreateChild

The input parameter deviceType changes as follows:

Parameters	Description
deviceType	XML document containing the information specified by the <a href="#">BTMInitSchema</a>

For description of the method refer to [IFdtTopology::CreateChild\(\)](#).

##### 4.9.14.2 DeleteChild

For description of the method refer to [IFdtTopology::DeleteChild\(\)](#).

##### 4.9.14.3 GetChildNodes

For description of the method refer to [IFdtTopology::GetChildNodes\(\)](#).

##### 4.9.14.4 GetBtmForSystemTag

For description of the method refer to [IFdtTopology::GetDtmForSystemTag\(\)](#).

#### 4.9.14.5 GetBtmInfoList

For description of the method refer to [IFdtTopology::GetDtmInfoList\(\)](#).

#### 4.9.14.6 GetParentNodes

For description of the method refer to [IFdtTopology::GetParentNodes\(\)](#).

#### 4.9.14.7 MoveChild

For description of the method refer to [IFdtTopology::MoveChild\(\)](#).

#### 4.9.14.8 ReleaseBtmForSystemTag

For description of the method refer to [IFdtTopology::ReleaseDtmForSystemTag\(\)](#).

## 4.10 General Concepts

This section provides general information about the FDT Interfaces, and some background information about how the designers of FDT expect these interfaces to be implemented and used.

### 4.10.1 Task Related FDT Interfaces

All FDT interfaces are task related. Each object must implement a mandatory set of interfaces expected by all other objects. By implementing optional FDT interfaces an object is able to support additional functionality, e.g., a DTM may provide documentation in XML format or special diagnostics, a frame-application may provide audit trail functionality.

Each object is able to determine the availability of such optional interfaces of other objects during runtime. All defined FDT interfaces are fixed and will never be changed. Additional future extensions will be based on additional optional interfaces. A DTM or frame-application is then able to add a higher FDT version support by implementing or using such additional FDT interfaces.

Device Type Manager	Availability	User Interface	ActiveX Control User Interface	Device with Online Data	Gateway DTM	Communication-DTM for PC/Fieldus adapter
<b>IPersistXXX</b>	Mandatory					
IDtm	Mandatory					
IDtm2	Mandatory					
IDtmActiveXInformation	Optional		Mandatory			
IDtmApplication	Optional	Mandatory				
IDtmChannel	Optional				Mandatory	Mandatory
IDtmDocumentation	Mandatory					
IDtmDiagnosis	Mandatory					
IDtmImportExport	Optional					
IDtmInformation	Mandatory					
IDtmInformation2	Mandatory					
IDtmOnlineDiagnosis	Mandatory					
IDtmOnlineParameter	Optional			Mandatory		
IDtmParameter	Mandatory					
IFdtCommunicationEvents	Optional			Mandatory	Mandatory	
IFdtCommunicationEvents2	Optional			Mandatory	Mandatory	
IDtmHardwareIdentification	Optional					
IDtmSingleDeviceDataAccess	Optional			Mandatory		
IDtmSingleInstanceDataAccess	Mandatory					
IFdtEvents	Mandatory					

DTM ActiveX Control	Availability
IDtmActiveXControl	Mandatory

FDT Channel	Availability	Channel with User Interface	Channel of Gateway DTM	Channel of Communication-DTM for PC/Fieldus adapter
IFdtChannel	Mandatory			
IFdtChannelActiveXInformation	Optional	Mandatory		
IfdtChannelSubTopology	Optional		Mandatory	Mandatory
IfdtChannelSubTopology2	Optional		Mandatory	Mandatory
IfdtCommunication	Optional		Mandatory	Mandatory
IfdtFunctionBlockData	Optional		Mandatory	Mandatory
IfdtChannelScan	Optional		Mandatory	Mandatory

<b>FDT Channel ActiveXControl</b>	<b>Availability</b>
IfdtChannelActiveXControl	Mandatory
IfdtChannelActiveXControl2	Mandatory

<b>BTM</b>	<b>Availability</b>
IBtm	Mandatory
IDtmActiveXControlInformation	Mandatory
IDtmChannel	Optional
IDtmDocumentation	Mandatory
IDtmDiagnosis	Mandatory
IDtmImportExport	Optional
IBtmInformation	Mandatory
IDtmInformation2	Mandatory
IDtmOnlineDiagnosis	Mandatory
IDtmOnlineParameter	Mandatory
IBtmParameter	Mandatory
IFdtCommunicationEvents	Mandatory
IFdtCommunicationEvents2	Mandatory
IDtmHardwareIdentification	Mandatory
IDtmSingleDeviceDataAccess	Mandatory
IDtmSingleInstanceDataAccess	Mandatory
IFdtEvents	Mandatory

<b>BTM ActiveXControl</b>	<b>Availability</b>
IBtmActiveXControl	Mandatory

<b>Frame Application</b>	<b>Availability</b>	<b>With User Interface</b>
IDtmEvents	Mandatory	
IDtmEvents2	Mandatory	
IDtmAuditTrailEvents	Mandatory	
IFdtActiveX	Optional	Mandatory
IFdtActiveX2	Optional	Mandatory
IFdtBulkData	Optional	
IFdtContainer	Mandatory	
IFdtDialog	Mandatory	
IFdtTopology	Mandatory	
IFdtBtmTopology	Mandatory	
IDtmScanEvents	Optional	
IDtmRedundancyEvents	Optional	
IDtmSingleDeviceDataAccessEvents	Mandatory	
IDtmSingleInstanceDataAccessEvents	Mandatory	

**Furthermore, the prefixes FDT, DTM and BTM are reserved for identifiers and names defined in the FDT specification. This prevents conflicts of further releases with private extensions of interfaces or definitions.**

In general, all FDT interfaces are designed with fieldbus- and manufacturer-neutral methods. Extensions for a new fieldbus are done via new XML schemas. Functional extensions for new tasks will be provided by new interfaces.

#### 4.10.2 Return Values of Interface Methods

Interface methods indicate success or failure of a method call by well defined return values (marked as [out, retval]). COM errors (HRESULT) must not be used to return FDT function related errors, except it is stated in the specification. If no return value is defined (e.g. for all Event methods) it is assumed that the method always succeeds.

### 4.10.3 Dual Interfaces

All interfaces defined within the FDT Specification are implemented as dual interfaces. This decision was made to support C++, Visual Basic, Java and other COM compliant development languages. The functionality of an object is implemented in separate task oriented interfaces, so that only the default interface is accessible via the dispatch interface. This prevents marshaling of the extra interfaces to dispatch-only clients, but the extra interfaces can be made available for a dispatch only-client via a wrapper that holds the other interfaces as properties or merges all methods to a single interface.

Due to the better performance, the developer should use the custom interface. However, in general the dispatch interface can be accepted, because the marshalling time of most of the FDT methods can be neglected compared with the runtime of each method.

### 4.10.4 Unicode

All string parameters to the FDT interfaces are BSTRs and are therefore UNICODE strings.

Microsoft MIDL Version 3.0 or later is required to correctly compile the IDL code and generate proxy/stub software. Microsoft Windows NT 4.0 (or later), or Windows 95 with DCOM support is required to properly handle the marshaling of FDT parameters.

Note that in order to implement FDT software that will run on both Microsoft Windows NT and Microsoft Windows 95, it is necessary for these components to test the platform at runtime. In the case of Microsoft Windows 95, usually the conversion of any strings to be passed to Win32 from UNICODE to ANSI needs to be done. Visual Basic makes this conversion implicitly.

The only limitation within this specification is that a NUL character (i.e.0) is only allowed as the last character of any BSTR method parameter to prevent conversion errors (UNICODE<->ANSI, uppercase<->lowercase, etc.) within the database.

### 4.10.5 Asynchronous vs. Synchronous Behavior

In general each function call is synchronous. Within FDT there are two special cases of asynchronous behavior.

- After starting the user interface of a DTM, the DTM works asynchronous to the Frame Application. Asynchronous in this case means that the user works with the DTM and the Frame Application is the server for communication and data access. This state ends by a notification to the Frame Application, when the DTM closes the user interface.
- While a DTM has opened its user interface, the DTM uses the asynchronous behavior at the communication interface. The time a communication function call needs to return depends on the system topology and the bus protocol and would block an application for seconds. Dividing a communication function call to a request and a response function causes a non-blocking behavior without the pain of multi-threading implementation. The DTM sends its request or several requests without being disturbed by incoming responses. When a response is available, the DTM gets a notification and can receive the response from the communication component. Due to this mechanism, on one hand a DTM should not implement a timeout control and on the other hand the communication has to provide a response for each request. Only a response can contain the timeout information.

## 4.10.6 ProglDs

The usage of proglDs is limited to 39 characters. This decision was made to support C++, Visual Basic, Java and other COM compliant development languages.

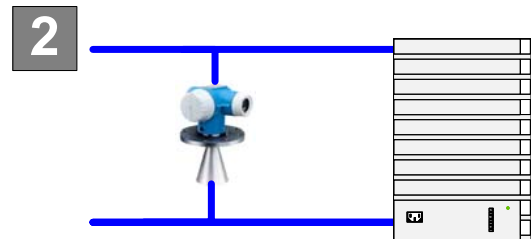
## 4.10.7 Slave Redundancy

### 4.10.7.1 Redundancy Scenarios

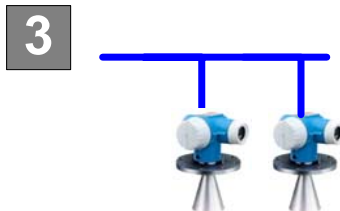
one fieldbus, one device, two addresses



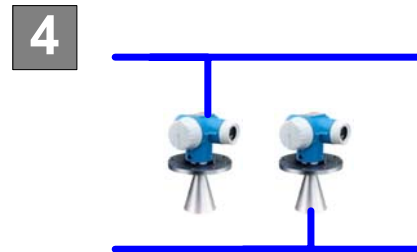
two fieldbuses, one device, two addresses, one fieldbus master



one fieldbus, two devices, each with specific address



two fieldbuses, each with a device



Scope of slave redundancy within FDT is one DTM for configuration of one device connected to either

- Scenario 1: one fieldbus using two different addresses
- Scenario 2: connected to two different fieldbuses controlled by one bus master.

Separate redundant devices (scenario 3 and 4) can not be handled within FDT, these scenarios would require additional Frame Application functionality e.g. with regard to data synchronization.

In Scenario 1 and 2 redundant field buses must be managed by one redundancy aware parent component (e.g. a COMM-DTM) specific for the appropriate redundant field bus technology. This parent component should typically also be able to handle DTMs for redundant and not-redundant field devices concurrently.

### 4.10.7.2 Redundancy Support in Frame Application

The DTMs able to handle slave redundancy can be used by any FDT Frame Application without need for specific redundancy support or knowledge. A FDT Frame Application may optionally implement the IDtmRedundancyEvents interface. Such a Frame Application is then able to display redundant slaves in it's topology view, e.g. a field device connected to two different lines or by using specific device icons within a topology treeview.



Within a Frame Application not implementing the `IDtmRedundancyEvents` interface a DTM representing a redundant slave can not be distinguished within the topology view. But the DTM and the parent component themselves typically display additional information, e.g. additional fieldbus addresses. Nevertheless within such a Frame Application these DTMs provide full functionality with regard to fieldbus redundancy.

As a redundant slave is represented by one DTM instance no additional functionality with regard to dataset synchronization or multiuser is required.

#### 4.10.7.3 Parent Component for Redundant Fieldbus

Fieldbus communication for redundant slaves is handled by a fieldbus and redundancy technology specific parent component, e.g. a specific COMM-DTM. All fieldbuses used to connect redundant slaves must be handled by one instance of such a parent component. In Scenario 2 for example, each fieldbus line is represented by an appropriate communication channel of the parent component. Therefore a Frame Application is able to use such a parent component without knowledge about the physical redundant fieldbus structure.

During a `ValidateAddChild()` and a `OnAddChild()` the parent component is able to detect if a Device-DTM to be added is able to handle redundancy by examining the parameter document of the instantiated DTM. In this case a specific address selection dialog within the parent component can be used. This address selection is fieldbus and redundancy specific. It's up to the parent component if redundancy is handled automatically or only after user interaction.

If the Frame Application has implemented the `IDtmRedundancyEvents` Interface the parent component has to inform the Frame Application about the redundant DTM by calling the `OnAddedRedundantChild()` method.

The parent component must be able to handle all possible communication paths to the redundant device. Within a `SetParameter()` call complete redundant address information can be provided to the DTM instance handling this redundant device. This DTM is then able to use this information to display all available communication paths. During a `ConnectRequest()` the DTM provides this complete address information to the parent component. The parent component is then able to select the currently active communication path. The communication path can be changed within the parent component without need for interaction with the Device-DTM.

A parent component may also be able to handle a DTM representing a not redundant field device. In this case this DTM is handled without using the redundancy specific information within FDT xml documents.

#### 4.10.7.4 Redundancy Support in Device-DTM

A DTM able to handle a redundant device provides additional information within it's parameter document.

After an `OnAddChild()` complete redundant address information can be given by the parent component to the Device-DTM. The Device-DTM is then able to detect if it's appropriate slave is used as redundant slave. This complete address information must be used by the Device-DTM within a `IFdtCommunication::ConnectRequest()`, but can also be used for diagnosis and status information. Typically now additional redundancy handling is required within the Device-DTM itself.

A DTM handling a redundant device must check all addresses provided during a `SetParameter()` call. If the Device-DTM is not able to handle this addresses, e.g., because only a vendor-specific offset can be used, an error message should be created and a `FALSE` should be returned to the calling parent component. In this case the parent component is able to detect Device-DTMs which can not be used at the redundant fieldbus. A Device-DTM must save all redundancy information in it's instance dataset.

A DTM representing a redundant slave can be used as child of a non redundant aware parent component. In this case the Device-DTM must not use the additional FDT redundancy functionality, dialogs or redundant specific contents of FDT xml documents.

#### 4.10.7.5 Scan and Redundant Slaves

Fieldbus scan provides for each address of a redundant slave one entry, a redundant slave can not be detected.

In scenario 1 mapping of redundant addresses to one instance is only possible for a DTM itself, in scenario 2 mapping can only be done based on project knowledge. Typically a scan is not executed on a redundant system.

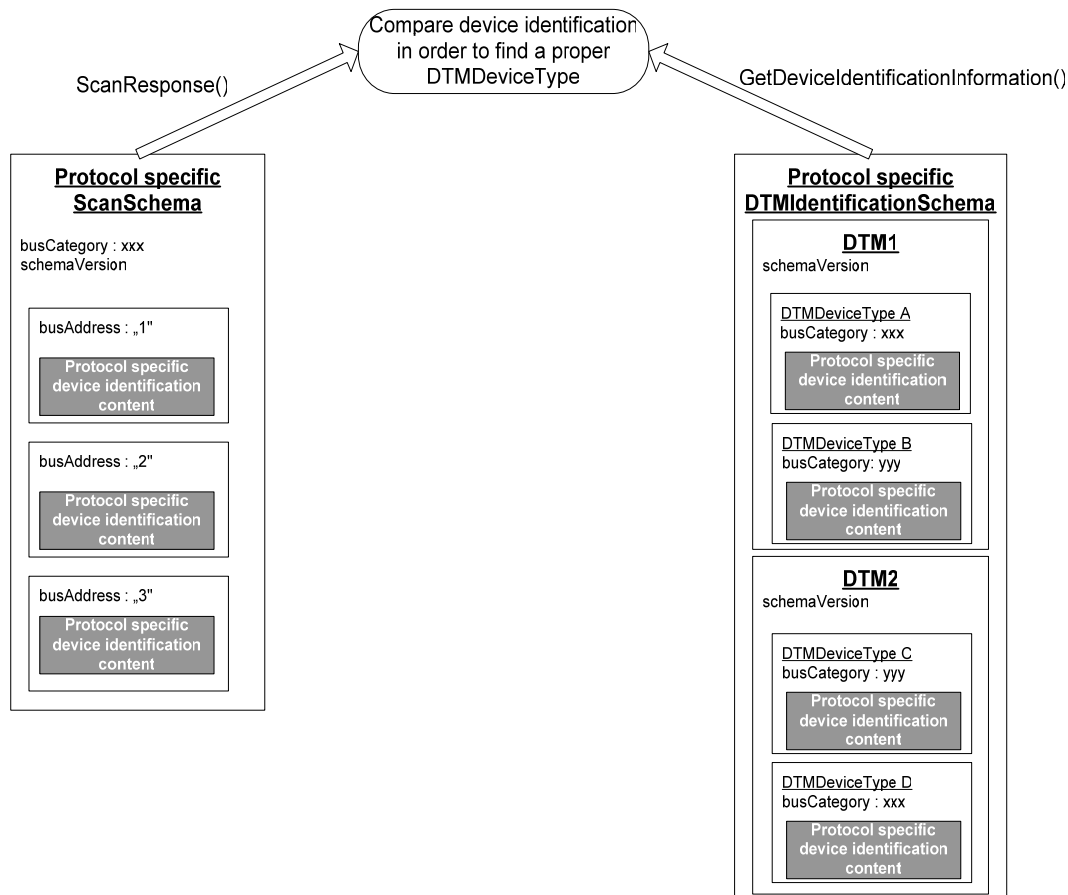
#### 4.10.7.6 Topology Import-/Export

A Frame Application not aware of DTMs handling redundant slaves is not able to provide redundancy information within a FDT topology export file.

A Frame Application aware of DTMs handling redundant slaves can only add a DTMNode element at ChannelNodes if the appropriate DTM instance has not been added to topology by a [IDtmRedundancy::OnAddedRedundantChild\(\)](#) event call. Instead the appropriate DTM element of the topology document should contain a BusInformation element containing the redundant address information.

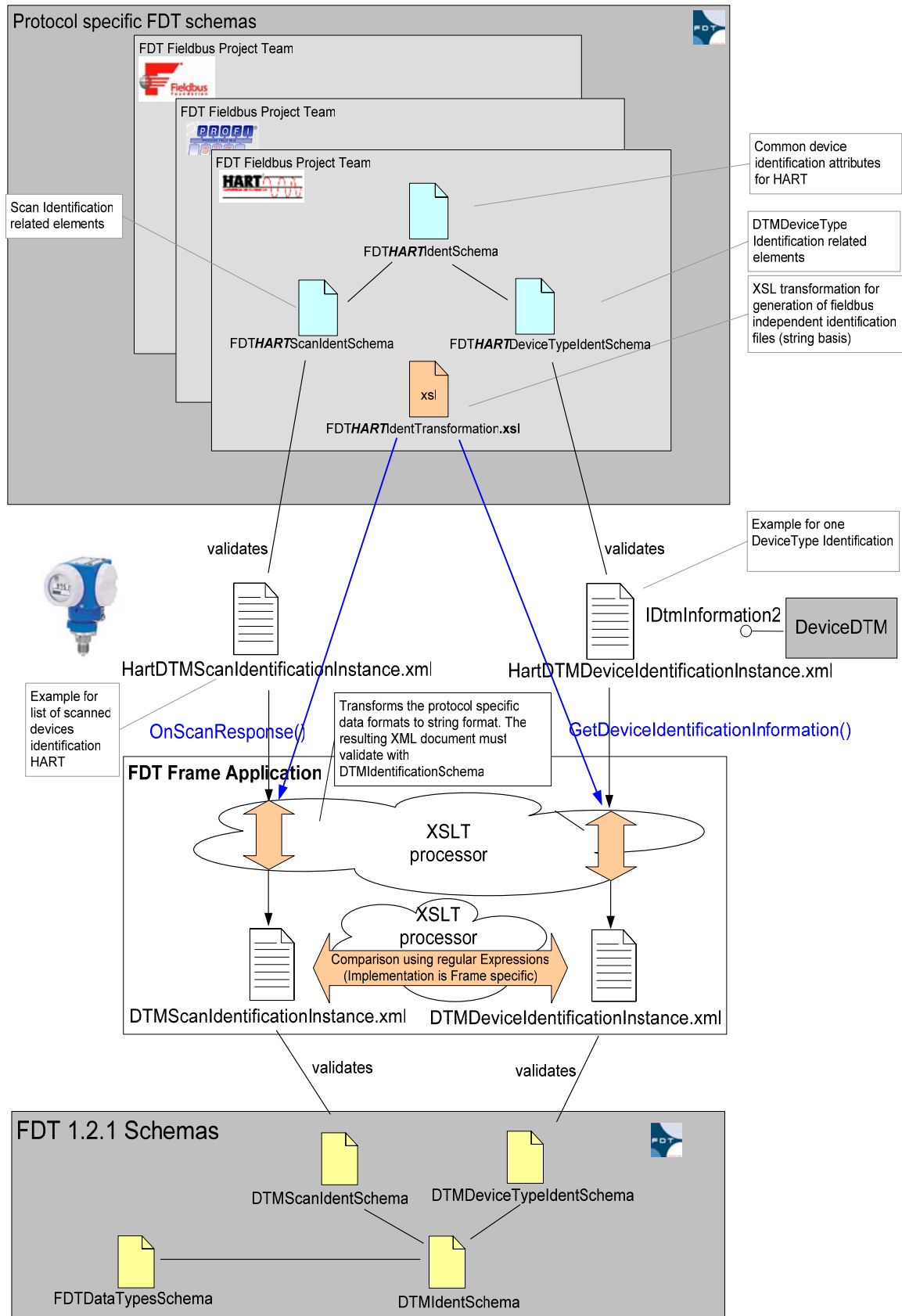
## 4.10.8 Field Bus Scanning and DTM Assignment

### 4.10.8.1 Device Identification



In order to identify a proper DTMDDeviceType, a Frame Application has to compare identification information of scanned physical devices with identification information of a DTMDDeviceType. These files must be converted to a field bus independent format using a fieldbus specific XSL transformation.

The following picture shows, how protocol specific schemas are integrated in the FDT specification and used by Frame Application and DTMs (Example HART is shown):



Picture: Structural overview

#### 4.10.8.2 Protocol Specific Transformation Style Sheet (xsl)

As shown in the structural overview picture above, the protocol specific FDT specification extension covers a transformation style sheet (.xsl) in addition to schemas. This xsl is to be used by a Frame Application in order to convert the protocol specific formats included in the identification (scan and DTM) XML documents into strings. The result must be validated against the protocol independent FDT Schemas ([DTMScanIdentSchema](#) and [DTMDeviceTypeIdentSchema](#)). The output can be used by a Frame Application to compare scan and DTM values based on string format. A DTM may define a range of supported physical device types by including regular expressions. In order to identify a matching DTMDeviceType, a Frame Application must implement a pattern matching according to the regular expression syntax defined below.

#### 4.10.8.3 Semantic Identification Information

The following table lists identification elements, which have to be provided by Scan and DTM identification. After xsl transformation, the following values must be available in string format with the elements listed below in [DTMScanIdentSchema](#) and [DTMDeviceTypeIdentSchema](#).

Semantic element name	Description	Scan	DTM
IdAddress	Fieldbus address of the scanned physical device.	x	-
IdBusProtocol	Protocol identification (enum)	x	x
IdBusProtocolVersion	Version of bus protocol		
IdManufacturer	Manufacturer identification	x	?
IdTypeID	Device type identification	x	?
IdSoftwareRevision	Tool relevant version of the physical device – Firmware Version	x	?
IdHardwareRevision	Hardware version of the physical device	x	?
IdTag	Tag name, which set in the physical device	x	-
IdSerialNumber	In order to get a common definition for all kind of protocols, a serial number is defined to be only unique for one manufacturer and device type. For world wide unique identification this attribute must always be combined with manufacturerID and deviceTypeID.	x	-
IdDTMSupportLevel	Enum : genericSupport, profileSupport, blockspecificProfileSupport, specificSupport(=default)	-	x

X must be provided, - is not to be provided, ? optional

If a semantic element cannot be defined for a fieldbus protocol, the value must be set to 'NOT\_APPLICABLE'.

#### 4.10.8.4 Device Assignment

The comparison of scan result with DTM device identification information and the device assignment is done by the Frame Application based on it's internal rules.

Each element from the scan document can be compared with the DTM device identification information.

The element of the DTM device identification document must match to the value within the scan document.

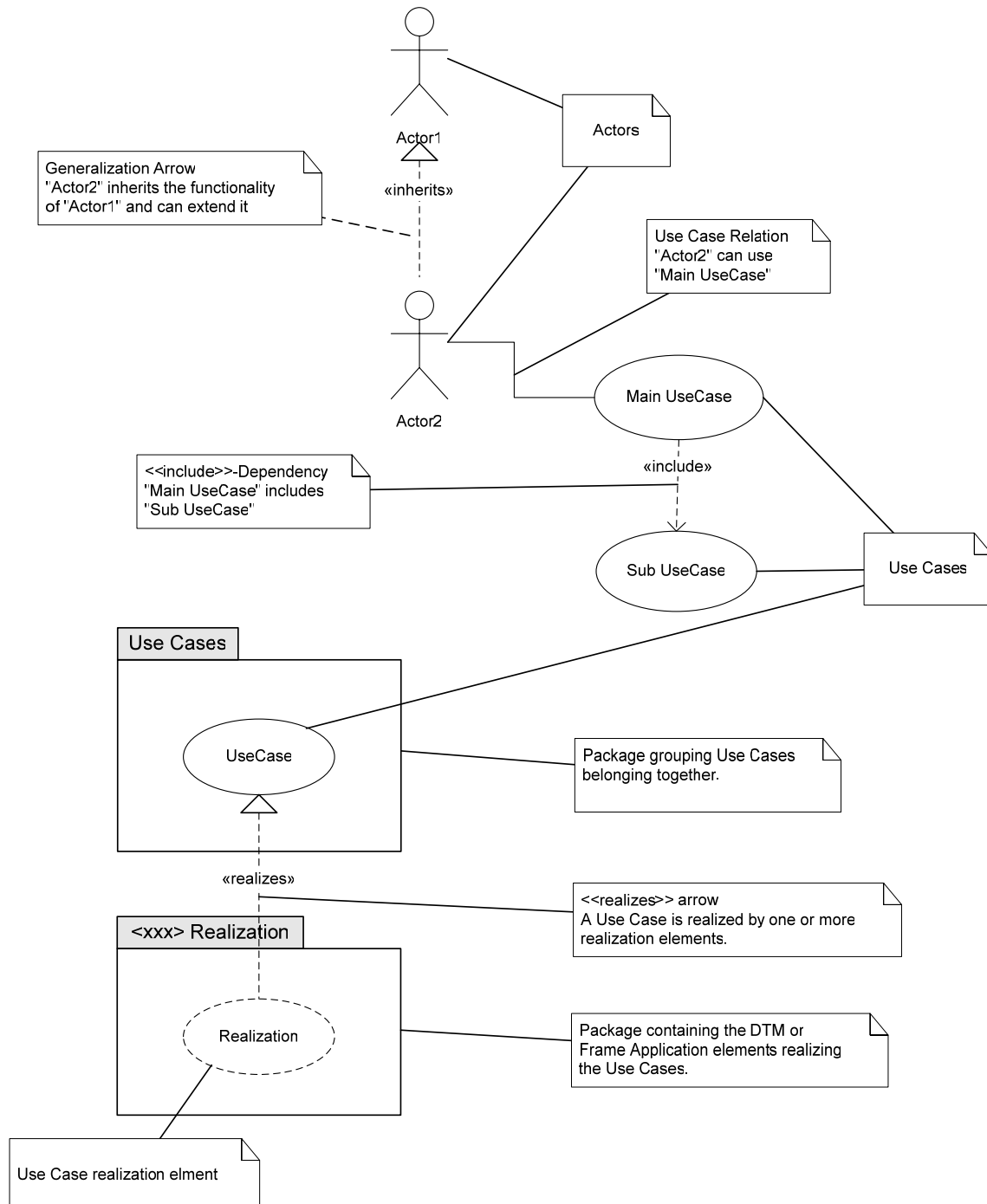
If the element of the DTM device identification document contains a pattern, the Frame Application must use regular expressions to compare scan and DTM device identification information.

#### 4.10.8.5 Regular Expression Specification

MetaCharacter	Description
.	Matches any single character.
[ ]	Indicates a character class. Matches any character inside the brackets (for example, [abc] matches "a", "b", and "c").
^	If this metacharacter occurs at the start of a character class, it negates the character class. A negated character class matches any character except those inside the brackets (for example, [^abc] matches all characters except "a", "b", and "c"). If ^ is at the beginning of the regular expression, it matches the beginning of the input (for example, ^[abc] will only match input that begins with "a", "b", or "c").
-	In a character class, indicates a range of characters (for example, [0-9] matches any of the digits "0" through "9").
?	Indicates that the preceding expression is optional: it matches once or not at all (for example, [0-9][0-9]? matches "2" and "12").
+	Indicates that the preceding expression matches one or more times (for example, [0-9]+ matches "1", "13", "666", and so on).
*	Indicates that the preceding expression matches zero or more times.
??, +?, *?	Non-greedy versions of ?, +, and *. These match as little as possible, unlike the greedy versions which match as much as possible. Example: given the input "<abc><def>", <.*?> matches "<abc>" while <.*> matches "<abc><def>".
( )	Grouping operator. Example: ([0-9]+,)*[0-9]+ matches a list of numbers separated by commas (such as "1" or "1,23,456").
\	Escape character: interpret the next character literally (for example, [0-9]+ matches one or more digits, but [0-9]\+ matches a digit followed by a plus character). Also used for abbreviations (such as \a for any alphanumeric character; see table below). If \ is followed by a number <i>n</i> , it matches the <i>n</i> th match group (starting from 0). Example: <{.*?}>.*?<\0> matches "<head>Contents</head>". Note that in C++ string literals, two backslashes must be used: "\\+", "\\a", "<{.*?}>.*?<\0>".
\$	At the end of a regular expression, this character matches the end of the input. Example: [0-9]\$ matches a digit at the end of the input.
	Alternation operator: separates two expressions, exactly one of which matches (for example, T the matches "The" or "the").
!	Negation operator: the expression following ! does not match the input. Example: a!b matches "a" not followed by "b".

## 5 FDT Session Model and Use Cases

The following figure shows the UML syntax that is used throughout this chapter.



The following figure shows the main use cases.

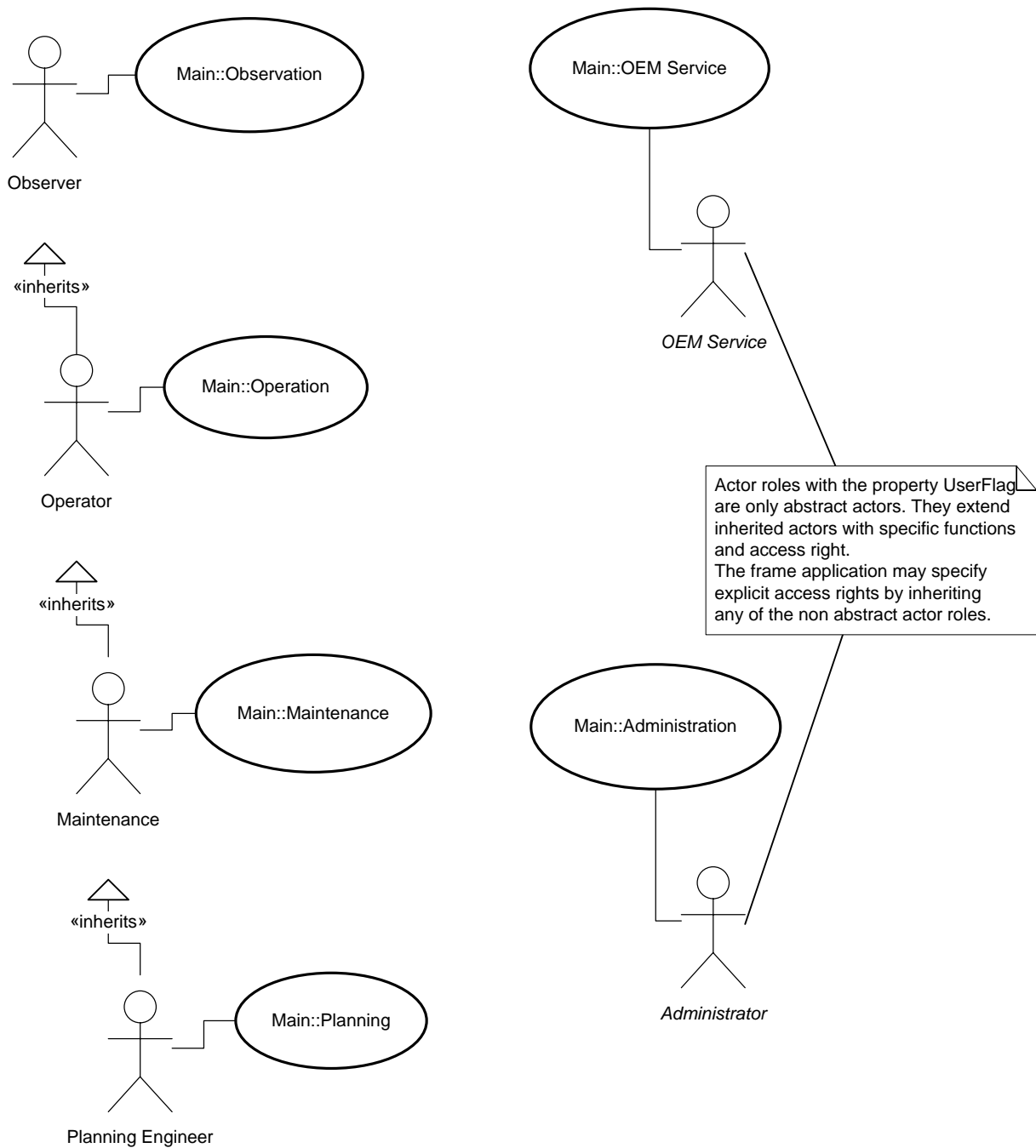


Figure 5: Use case - Main Use Case Diagram

They are specified in more detail in the following sections, including textual descriptions and optionally some necessary scenarios.



## 5.1 Actors

The actor types in the above displayed use case diagram are structured in a hierarchical way. The "Maintenance" actor inherits the "Operator" actor. The "Planning Engineer" actors inherits the use cases of the "Maintenance" actor. Both the "Administrator" and the "OEM Service" can extend the inherited actors by additional use cases. Use cases, which are passed on to an actor of higher level, may act in an extended way to match their intention.

The following tables will give a brief description of the actors' roles:

<b>Actor Name:</b>	Observer
<b>Brief Description:</b>	This actor stands for a person that may only observe the current process.
<b>Inherits:</b>	None
<b>User Level:</b>	FDTUserInformation.userLevel = observer
<b>Access Verification:</b>	The access as "Observer" actor may not have a password.
<b>Use Cases:</b>	None

<b>Actor Name:</b>	Operator
<b>Brief Description:</b>	This actor stands for a person who has to observe and manage the current process. The "Operator" may therefore check the current status of the device, modify set values and check if the device is well functioning. The use cases for this actor role should enable the user to perform a complete diagnosis, watch the actual status and parameter set as well as the current process variables.
<b>Inherits:</b>	Observer
<b>User Level:</b>	FDTUserInformation.userLevel = operator
<b>Access Verification:</b>	The access as "Operator" requires a password.
<b>Use Cases:</b>	User Login, Online View, Audit Trail, Archive, Report Generation, Asset Management

<b>Actor Name:</b>	Maintenance
<b>Brief Description:</b>	A "Maintenance" person should have the possibility to perform all necessary maintenance operations including device exchange, teaching, calibration, adjustment, ... 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.
<b>Inherits:</b>	Operator
<b>User Level:</b>	FDTUserInformation.userLevel = maintenance
<b>Access Verification:</b>	The access as "Maintenance" actor requires a password.
<b>Use Cases:</b>	Simulation, Online Operation, Repair, Offline Operation, Bulk Data Handling User Login*, Online View*, Audit Trail*, Archive*, Report Generation*, Asset Management* (* Inherited use cases)

<b>Actor Name:</b>	Planning Engineer
<b>Brief Description:</b>	The actor " <a href="#">Planning Engineer</a> " stands for person like a plant engineer, a specialist (s. VDI/VDE2187) or any fully authorized person. This Person has access to the complete set of functions of the Frame Application and can use DTM functions without any restrictions. Only OEM-specific operations are locked.
<b>Inherits:</b>	<a href="#">Maintenance</a>
<b>User Level:</b>	FDTUserInformation.userLevel = planningEngineer
<b>Access Verification:</b>	The access as " <a href="#">Planning Engineer</a> " actor requires a password.
<b>Use Cases:</b>	<a href="#">DTM Instance Handling</a> , <a href="#">Simulation</a> *, <a href="#">Online Operation</a> *, <a href="#">Repair</a> *, <a href="#">Offline Operation</a> *, <a href="#">Bulk Data Handling</a> *, <a href="#">User Login</a> *, <a href="#">Online View</a> *, <a href="#">Audit Trail</a> *, <a href="#">Archive</a> *, <a href="#">Report Generation</a> *, <a href="#">Asset Management</a> * (* Inherited use cases)

<b>Actor Name:</b>	Administrator (abstract actor)
<b>Brief Description:</b>	The " <a href="#">Administrator</a> " actor stands for person that has to perform administrative operations within an engineering environment. He is responsible for adding and removing of software components.
<b>Inherits:</b>	Depends on the Frame Application
<b>User Flag*:</b>	FDTUserInformation.administrator = TRUE
<b>Access Verification:</b>	The access as " <a href="#">Administrator</a> " actor requires a password.
<b>Use Cases:</b>	<a href="#">Integration</a> and all inherited use cases

<b>Actor Name:</b>	OEM Service (abstract actor)
<b>Brief Description:</b>	The actor " <a href="#">OEM Service</a> " may perform the complete set of DTM specific functions. OEM specific operation may be accessible to an " <a href="#">OEM Service</a> " actor, like resetting of internal counters and exchanging firmware. The " <a href="#">OEM Service</a> " has only inherited use cases, but the inherited use cases (especially the " <a href="#">Online Operation</a> " use case) may have significant extensions.
<b>Inherits:</b>	Depends on the Frame Application
<b>User Flag*:</b>	FDTUserInformation.oemService = TRUE
<b>Access Verification:</b>	The access verification for an " <a href="#">OEM Service</a> " must have two security levels: 1. Frame Application password: enables access to the Frame Application functionality 2. Device-specific password: enables access to OEM operation with the device. The verification of the device specific password lies in the responsibility of the DTM or even the device itself.
<b>Use Cases:</b>	Inherited use cases only

\* "User Flags" may be combined with any "User Level", to specify the inherited actor role of an "Administrator" or "OEM Service" actor.

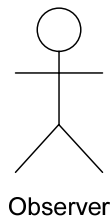
## 5.2 Use cases

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.

A realization diagram follows all use case descriptions in this section. These realization diagrams show the operations needed to build the related use case. An operation may either be a part of a DTM or a part of the Frame Application.

### 5.2.1 Observation



The "Observer" stands for a person, that has the lowest access level. No use case is associated with this person.

## 5.2.2 Operation

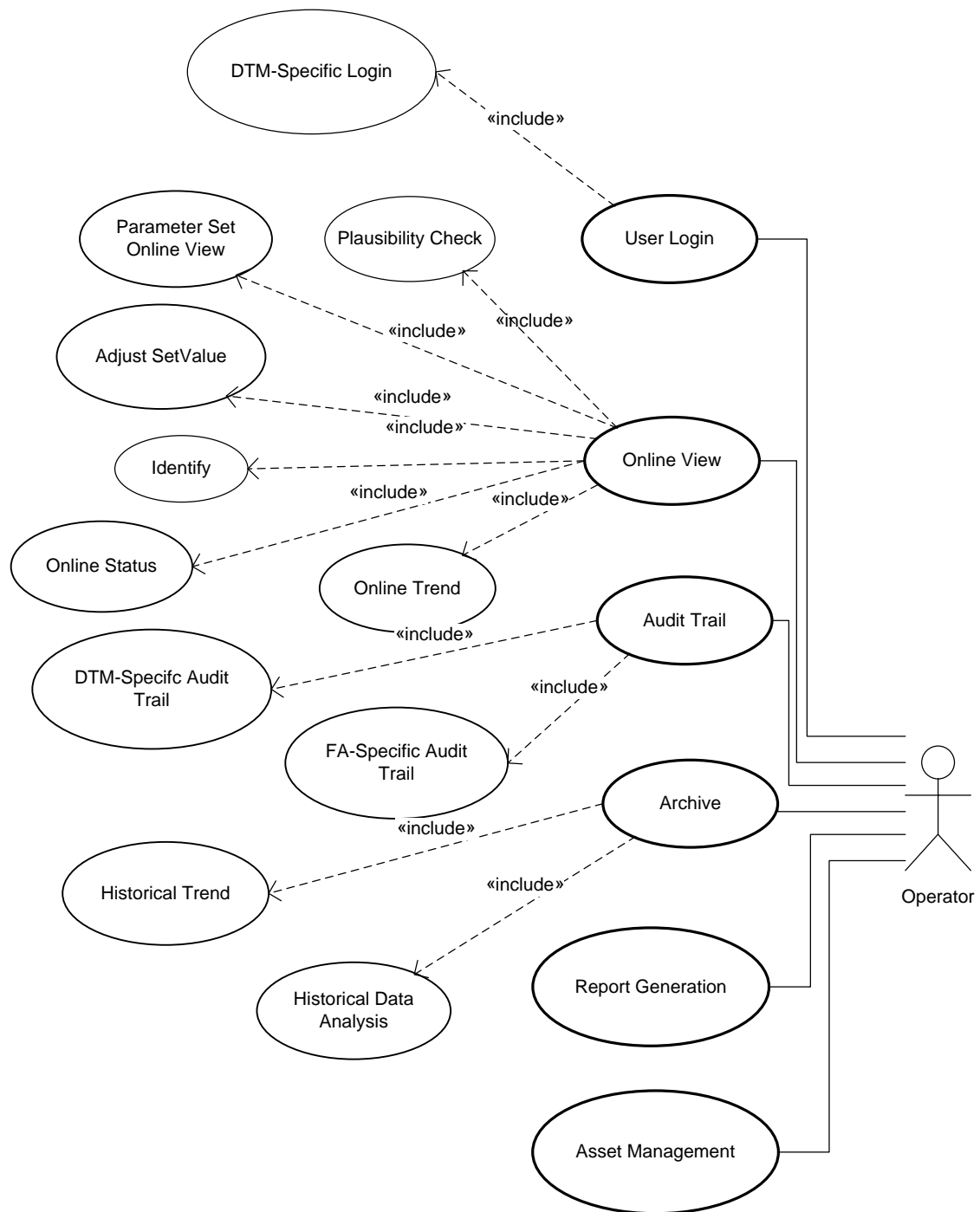


Figure 6: Use case - Operation

<b>Use Case:</b>		User Login
<b>Brief Description:</b>		A person of specified actor type logs into the Frame Application. If verification fails the person may act as an "Observer" actor only.
<b>GUI:</b>		Part of the Frame Application
<b>Print:</b>		No support
<b>Device Connection:</b>		Not established
<b>UserLevel</b>	Observer:	Depends on the Frame Application
	Operator:	Accessible
	Maintenance:	
	Planning Eng.:	
<b>UserFlag</b>	Administrator:	
	OEM Service:	Accessible with extended functionality (see below)
<b>Included Use Case:</b>		DTM-Specific Login
<b>Brief Description:</b>		Access to manufacturer specific information e.g. production codes, changes log
<b>GUI:</b>		Part of the DTM
<b>Print:</b>		No support
<b>Device Connection:</b>		Typically needs an established connection
<b>UserFlag</b>	OEM Service:	Accessible only for OEM Service

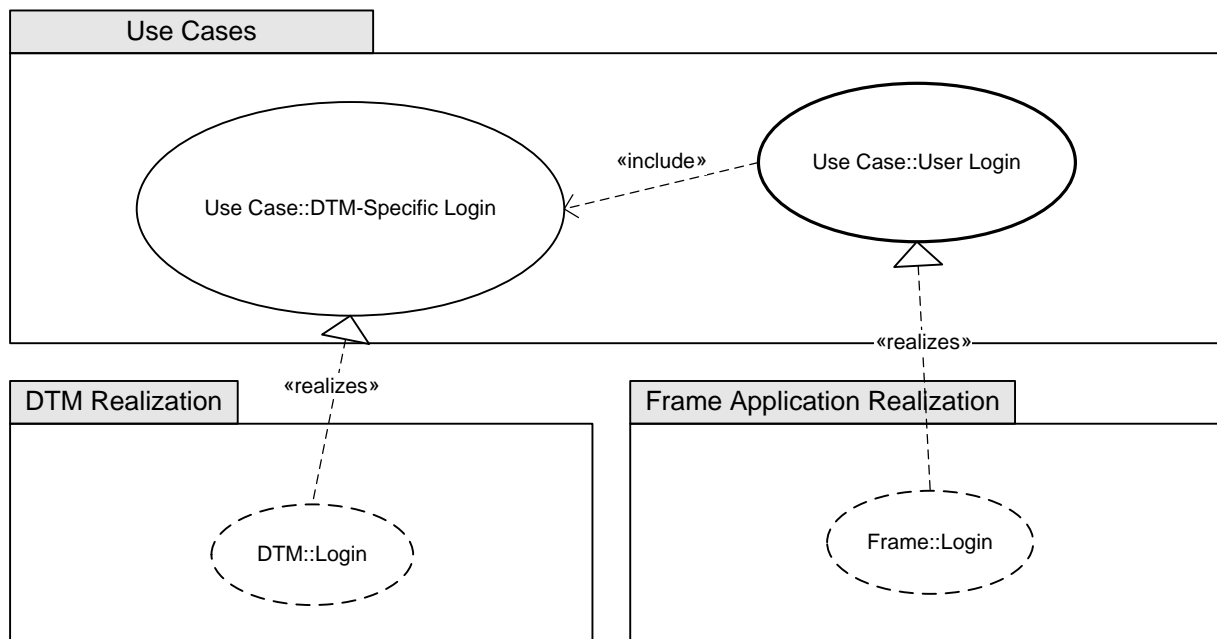


Figure 7: Realization of use case "User Login"

<b>Use Case:</b>		Online View
<b>Brief Description:</b>		This use case offers a complete set of operations for viewing of device parameters and status.
<b>GUI:</b>		The GUI is part of the DTM. The Frame Application can offer online views for a group of devices.
<b>Print:</b>		<a href="#">Online View</a> should always support print function to create documentation.
<b>Device Connection:</b>		Needs an established connection to one or more devices (Adjust SetValue may influence device data)
<b>UserLevel</b>	<a href="#">Observer:</a>	No access
	<a href="#">Operator:</a>	Accessible
	<a href="#">Maintenance:</a>	
	<a href="#">Planning Eng.:</a>	
<b>UserFlags:</b>		No changes
<b>Included Use Case:</b>		Parameter Set Online View
<b>Brief Description:</b>		Enables the actor to load parameters from the device for viewing and documenting (printing).
<b>Included Use Case:</b>		Adjust SetValue
<b>Brief Description:</b>		Enables the actor to adjust the SetValues of a device (e.g. positioner, controllers).
<b>Included Use Case:</b>		Online Status
<b>Brief Description:</b>		Use case for viewing and analyzing the current device status.
<b>Included Use Case:</b>		Online Trend
<b>Brief Description:</b>		Use case for generating and watching trends of dynamic online values.
<b>Included Use Case:</b>		Plausibility Check
<b>Brief Description:</b>		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.
<b>Included Use Case:</b>		Identify
<b>Brief Description:</b>		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 FDT-JIG-Annex specifications.

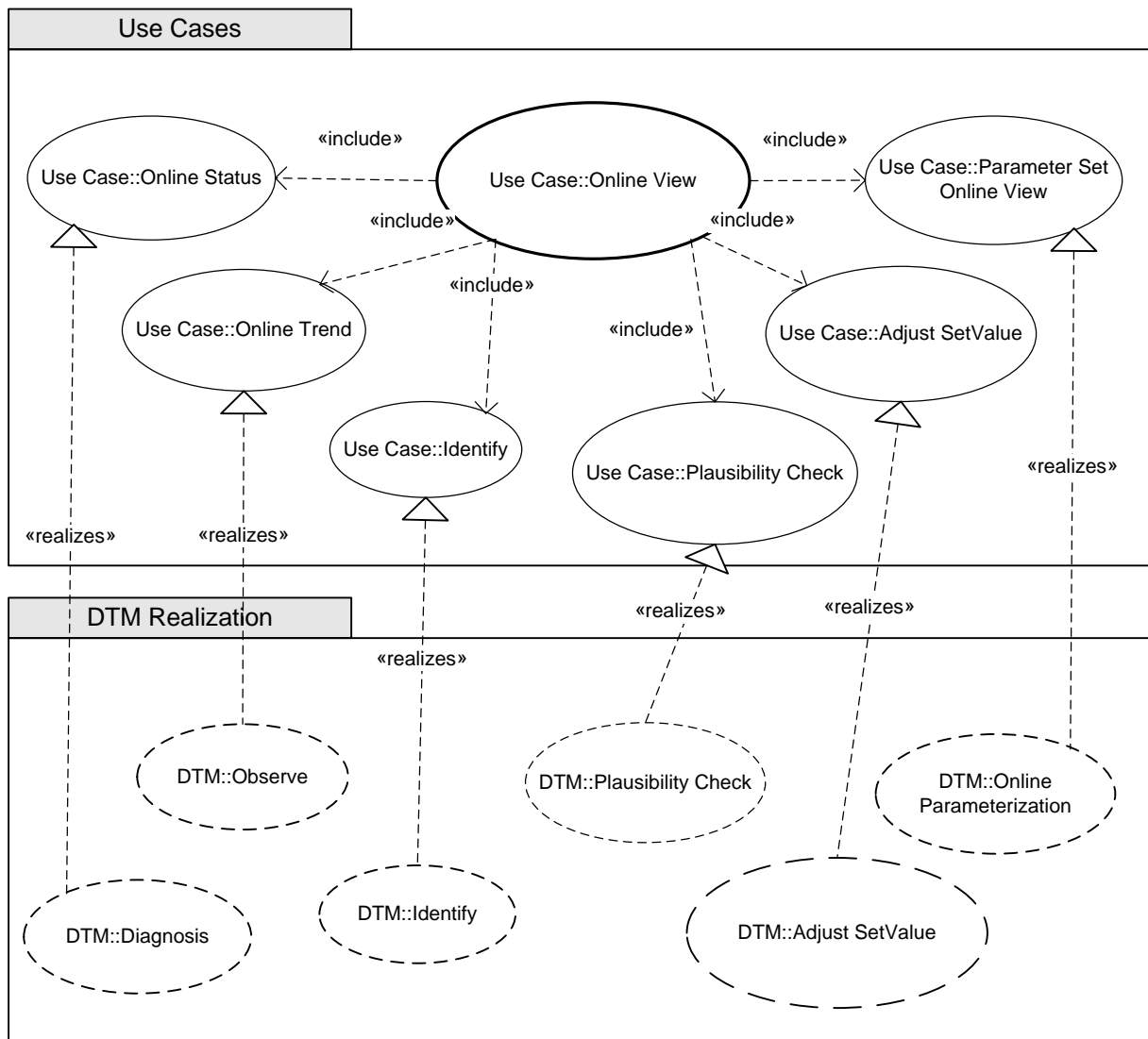


Figure 8: Realization of use case "Online View"

<b>Use Case:</b>		<b>Audit Trail</b>
<b>Brief Description:</b>		Use case to enable the actor viewing and deleting audit trail data.
<b>GUI:</b>		The component, which supports audit- trail, has to provide an adequate GUI.
<b>Print:</b>		Printing for documentation purpose as a need for audit trails and therefore has to be supported.
<b>Device Connection:</b>		No connection necessary
<b>UserLevel</b>	<b>Observer:</b>	No access
	<b>Operator:</b>	Accessible for viewing only
	<b>Maintenance:</b>	
	<b>Planning Eng.:</b>	View, export and delete
<b>UserFlags:</b>		No changes
<b>Included Use Case:</b>		<b>DTM-Specific Audit Trail</b>
<b>Brief Description:</b>		Every DTM might support an audit trail on its own.
<b>Included Use Case:</b>		<b>FA-Specific Audit Trail</b>
<b>Brief Description:</b>		The Frame Application may implement a system global audit trail using internal data and named DTM properties exported from the DTM via the IDtmAuditTrailEvents interface.

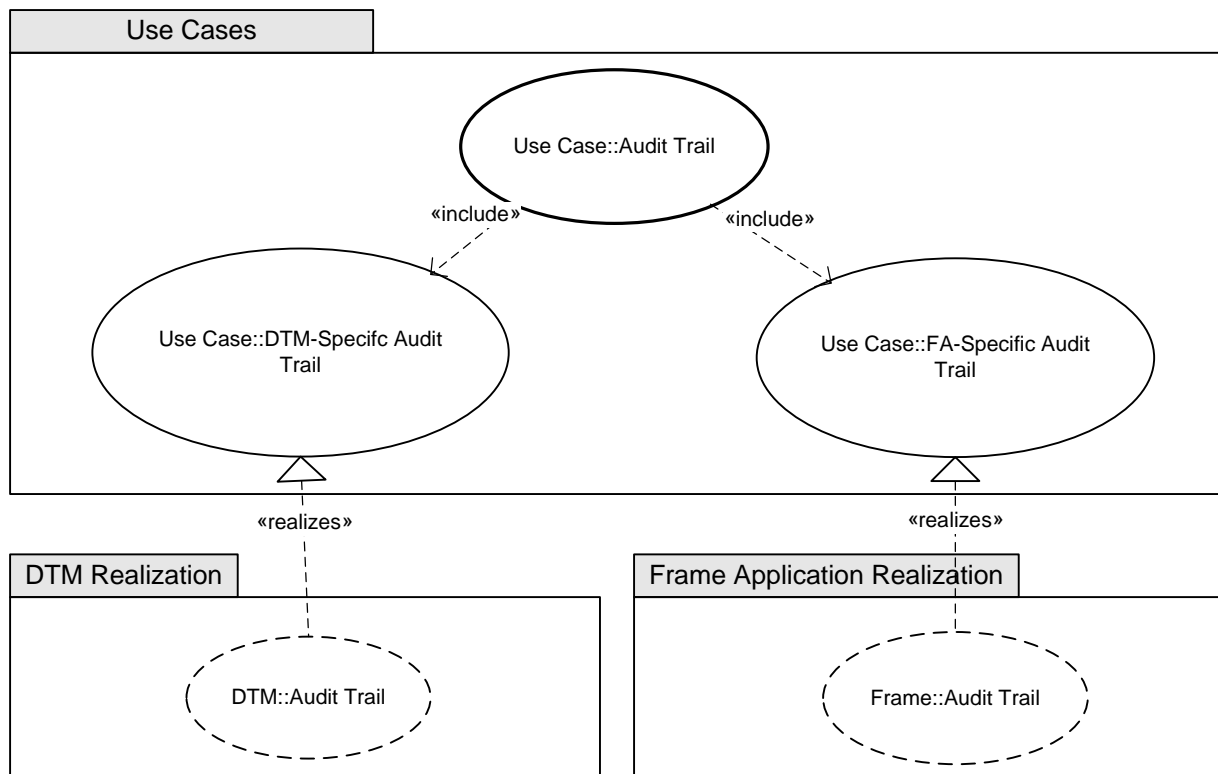


Figure 9: Realization of use case "Audit Trail"



<b>Use Case:</b>		Archive
<b>Brief Description:</b>		The "Archive" use case describes the access to a Frame Application archive for viewing and data analysis.
<b>GUI:</b>		The component, which supports archive functions, has to provide an adequate GUI.
<b>Print:</b>		Every archive component should support printing too.
<b>Device Connection:</b>		Not necessary
<b>UserLevel</b>	<b>Observer:</b>	No access
	<b>Operator:</b>	Accessible for viewing only
	<b>Maintenance:</b>	
	<b>Planning Eng.:</b>	View, export and delete
<b>UserFlags:</b>		No changes
<b>Included Use Case:</b>		Historical Trend
<b>Brief Description:</b>		The archive operations may support visualization of historical data (for example trending).
<b>Included Use Case:</b>		Historical Data Analysis
<b>Brief Description:</b>		Some Frame Applications and DTMs may support extended operations to analyze historical data.

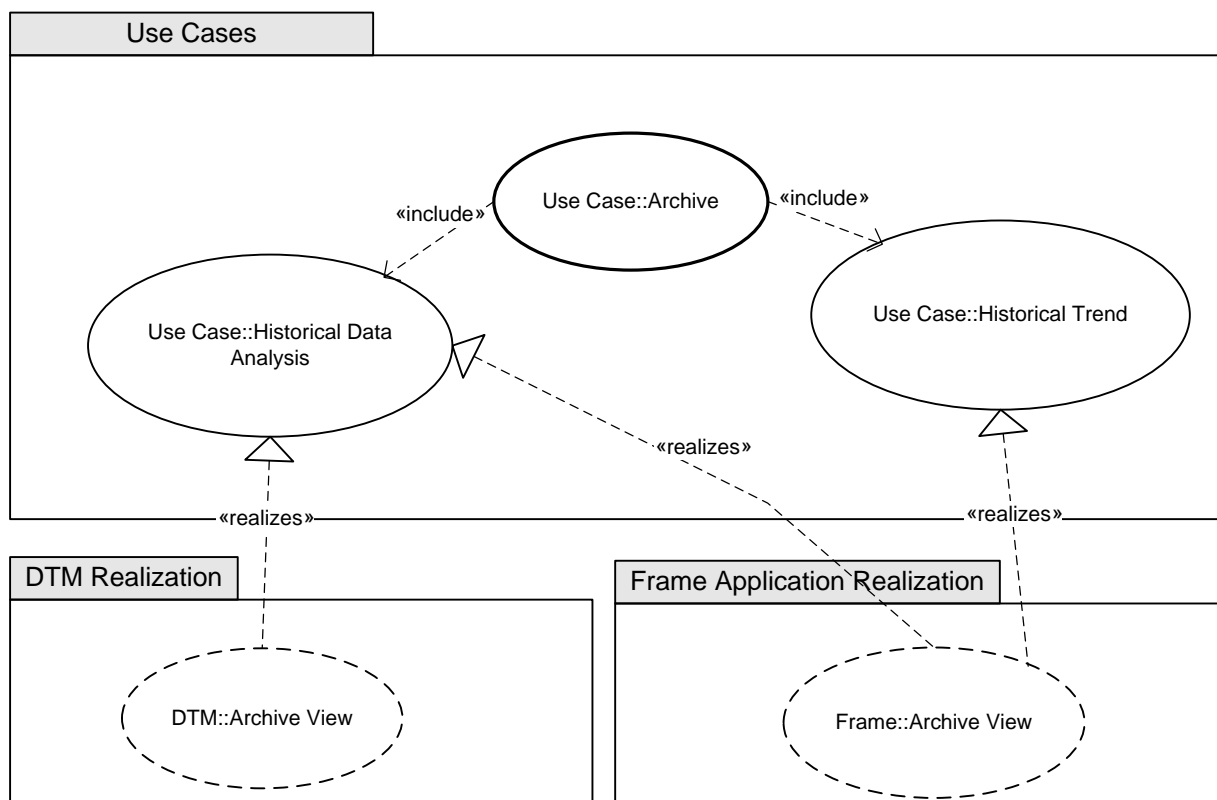


Figure 10: Realization of use case "Archive"

Use Case:	Report Generation
<b>Brief Description:</b>	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.
<b>GUI:</b>	Frame Application
<b>Print:</b>	Frame Application in combination with one or more DTMs
<b>Device Connection:</b>	Depends on the type of report
<b>UserLevel, UserFlag:</b>	The printed information may depend on user level and user flags.

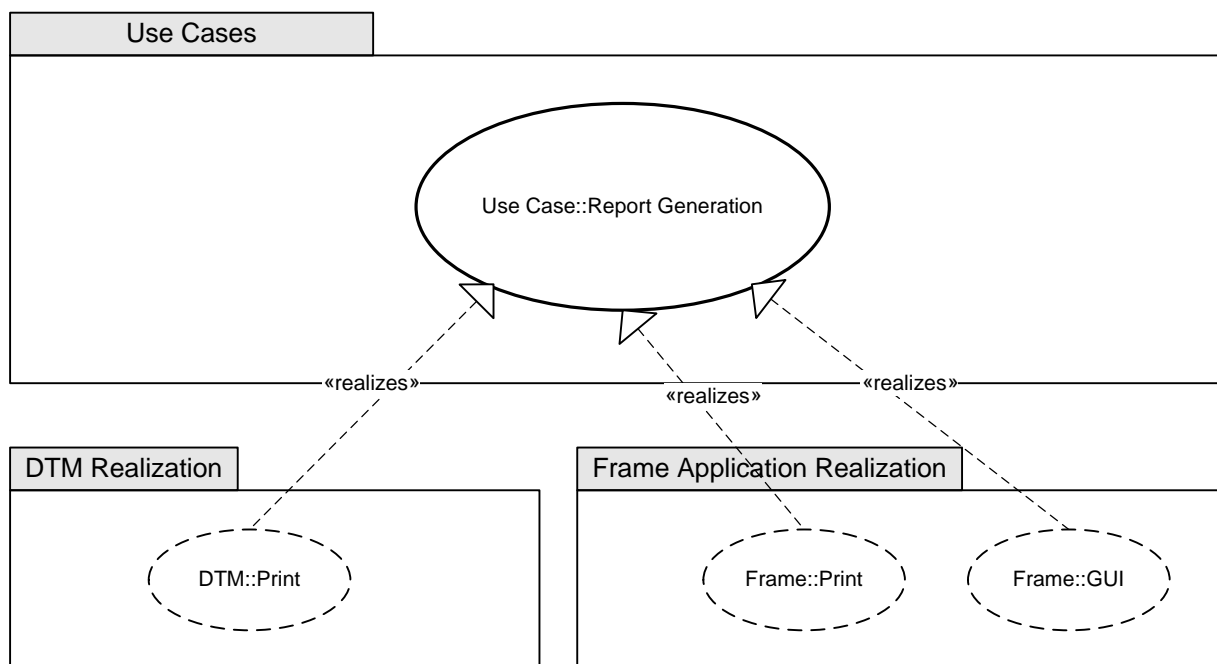


Figure 11: Realization of use case "Report Generation "

<b>Use Case:</b>	Asset Management	
<b>Brief Description:</b>	Frame Application provides mechanism for asset management	
<b>GUI:</b>	The component, which supports asset management functions, has to provide an adequate GUI.	
<b>Print:</b>	Frame Application supporting this use case should support printing, too.	
<b>Device Connection:</b>	Not necessary	
<b>UserLevel</b>	<b>Observer:</b>	No access
	<b>Operator:</b>	Accessible for viewing only
	<b>Maintenance:</b>	
	<b>Planning Eng.:</b>	View, export and delete
<b>UserFlags:</b>	No changes	

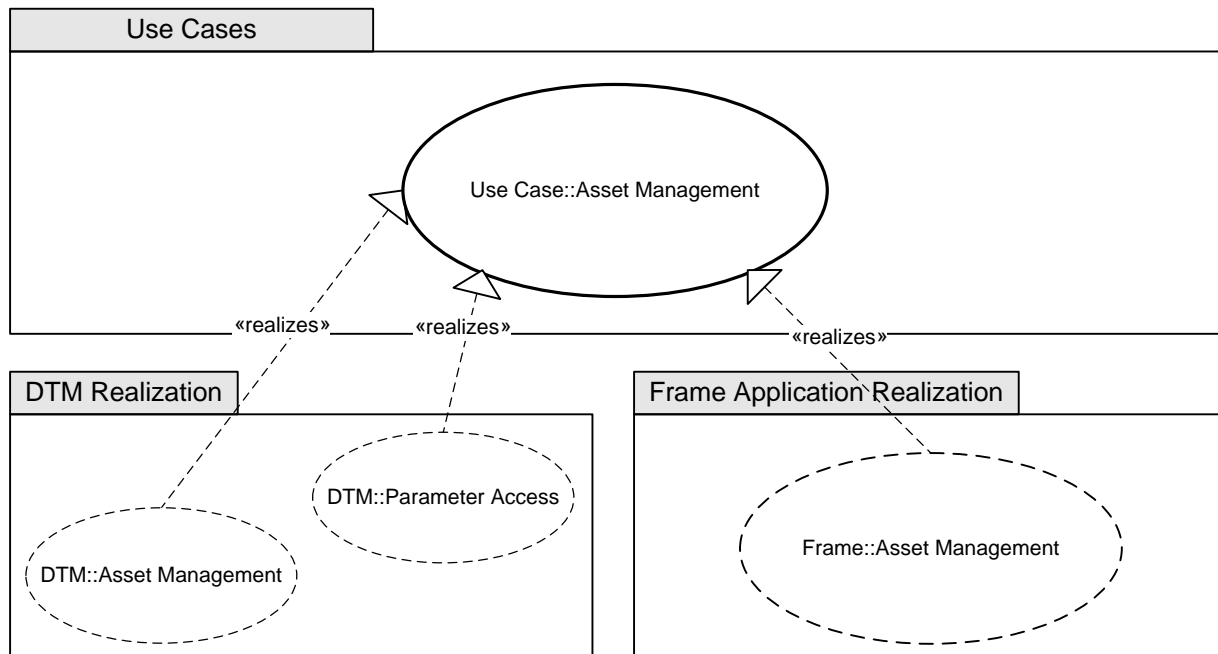


Figure 12: Realization of use case "Asset Management"

## 5.2.3 Maintenance

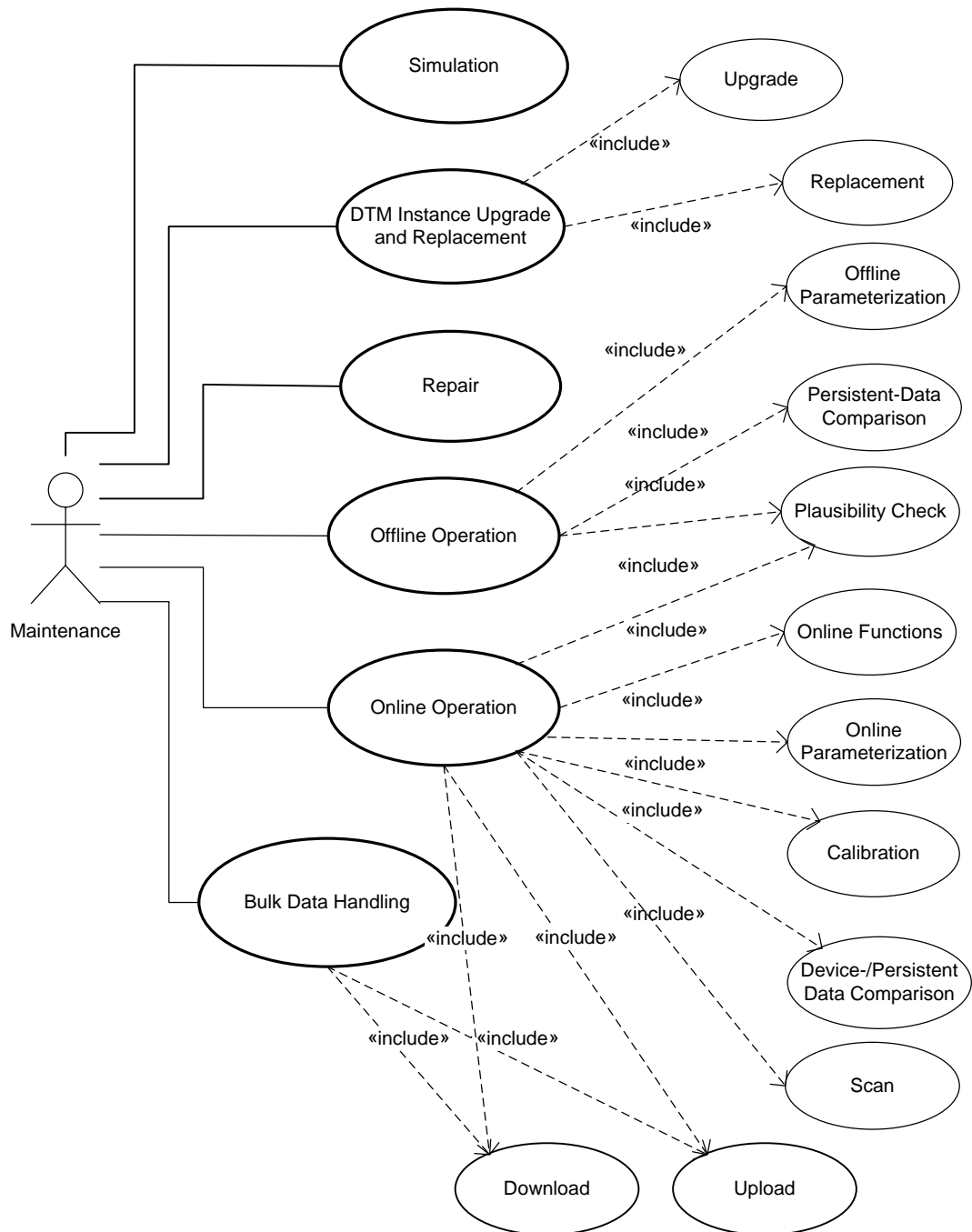


Figure 13: Use case - Maintenance

<b>Use Case:</b>		Simulation
<b>Brief Description:</b>		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.
<b>GUI:</b>		DTM-specific
<b>Print:</b>		DTM-specific
<b>Device Connection:</b>		Must have a connection established
<b>UserLevel</b>	<b>Observer:</b>	No access
	<b>Operator:</b>	
	<b>Maintenance:</b>	Accessible
	<b>Planning Eng.:</b>	
<b>UserFlags:</b>		No changes

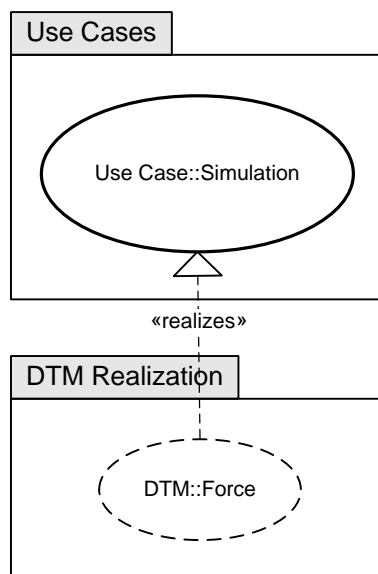


Figure 14: Realization of use case "Simulation"

<b>Use Case:</b>		Offline Operation
<b>Brief Description:</b>		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.
<b>GUI:</b>		DTM and Frame Application
<b>Print:</b>		DTM
<b>Device Connection:</b>		Depends on the included use case
<b>UserLevel</b>	<b>Observer:</b>	No access
	<b>Operator:</b>	
	<b>Maintenance:</b>	Accessible
	<b>Planning Eng.:</b>	
<b>UserFlags:</b>		No changes
<b>Included Use Case:</b>		Plausibility Check
<b>Brief Description:</b>		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: <ul style="list-style-type: none"> <li>• After display of a warning, inconsistent data may be stored</li> <li>• For safety reasons after display of a warning writing of data is prohibited.</li> </ul>
<b>Users:</b>		The plausibility check may be more tolerant for actors of higher level.
<b>Included Use Case:</b>		Offline Parameterization
<b>Brief Description:</b>		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.
<b>GUI:</b>		DTM
<b>Print:</b>		DTM
<b>Device Connection:</b>		No connection necessary
<b>Included Use Case:</b>		Persistent Data Comparison
<b>Brief Description:</b>		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 data sets may be editable and data may be transferred from one data set to the other.
<b>GUI:</b>		DTM
<b>Print:</b>		May be supported by the DTM
<b>Device Connection:</b>		Not necessary

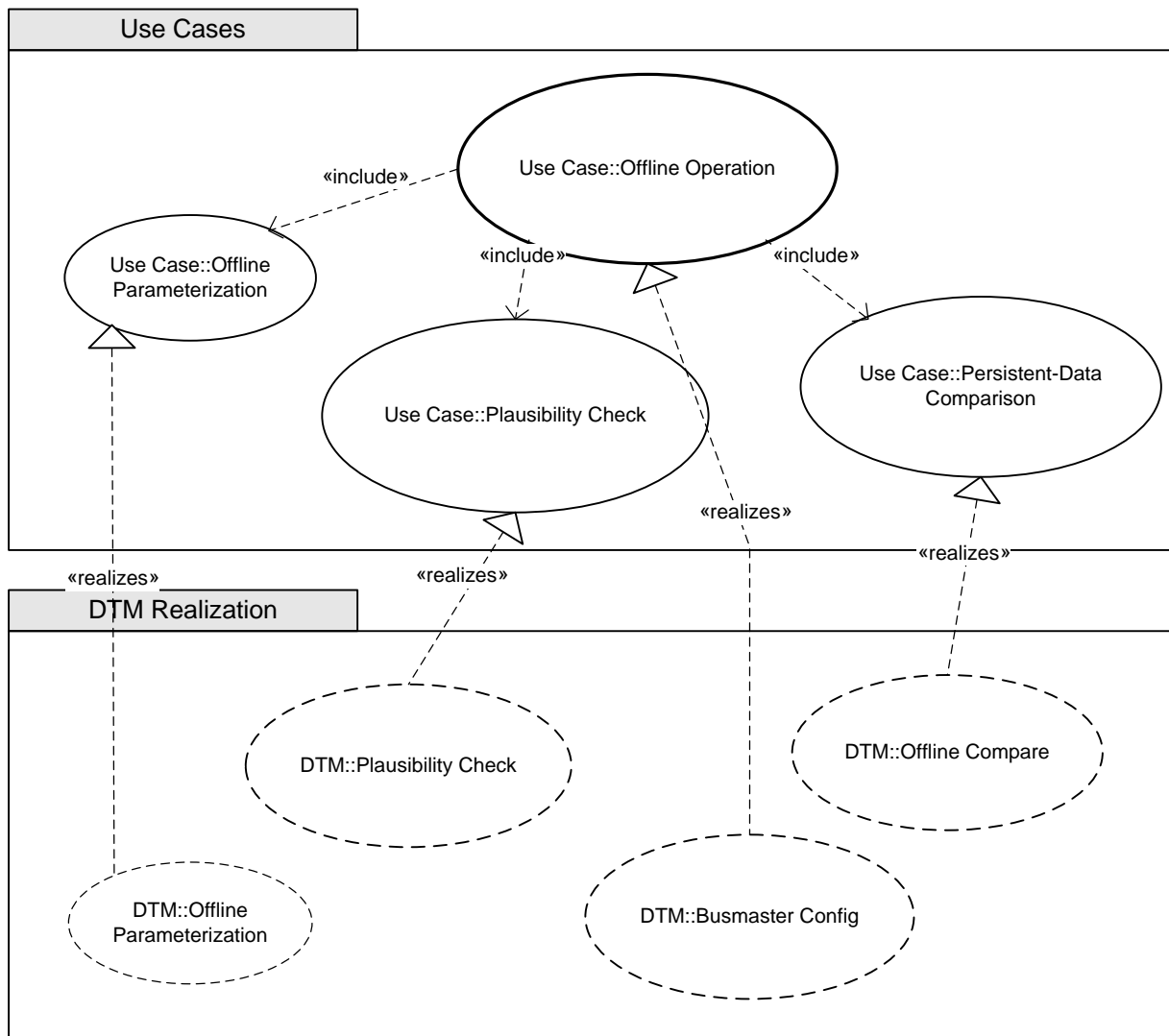


Figure 15: Realization of use case "Offline Operation"

Use Case:		Repair
<b>Brief Description:</b>		This use case stands for operations which must 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.
<b>UserLevel</b>	<b>Observer:</b>	No access
	<b>Operator:</b>	
	<b>Maintenance:</b>	Accessible
	<b>Planning Eng.:</b>	
<b>UserFlag</b>	<b>Administrator:</b>	No changes
	<b>OEM Service:</b>	Accessible with extended functionality (see below)

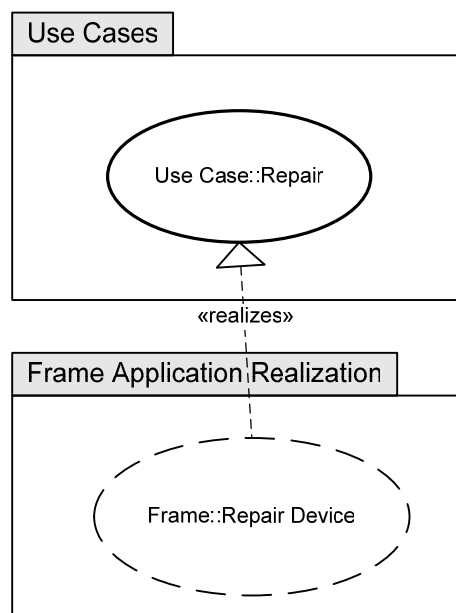
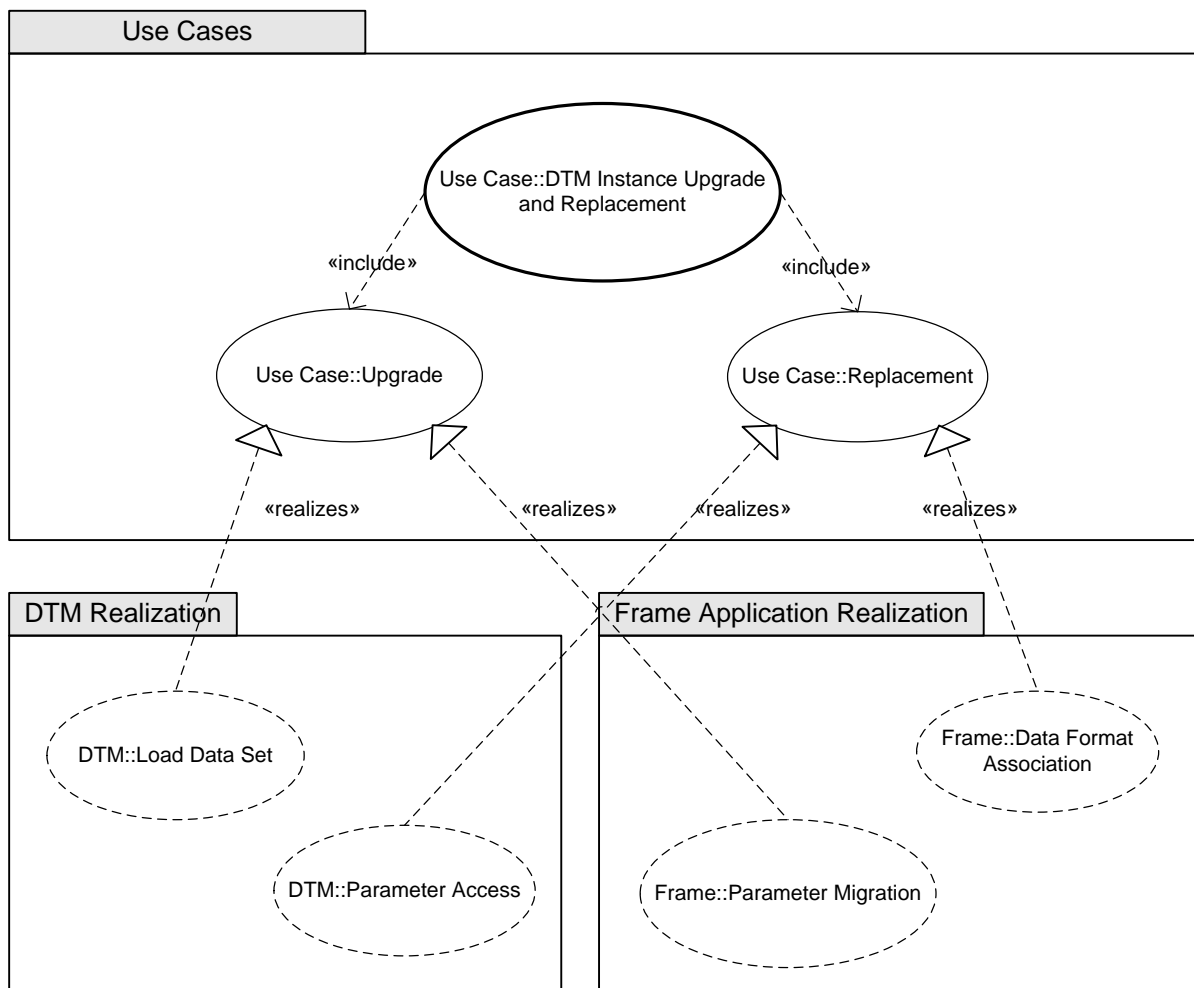


Figure 16: Realization of use case "Repair"



Use Case:		DTM Instance Upgrade and Replacement
<b>Brief Description:</b>		This use case stands for operations which must be performed to upgrade or to replace a DTM. Upgrade or replacement are used when the ProgId of the new DTM is different to the ProgId of 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.
<b>UserLevel</b>	<b>Observer:</b>	No access
	<b>Operator:</b>	
	<b>Maintenance:</b>	Accessible
	<b>Planning Eng.:</b>	
<b>UserFlag</b>	<b>Administrator:</b>	No changes
	<b>OEM Service:</b>	No changes
<b>Included Use Case:</b>		Replacement
<b>Brief Description:</b>		This use case stands for operations which must be performed to replace a DTM in the case of incompatible data set format.
<b>Included Use Case:</b>		Upgrade
<b>Brief Description:</b>		This use case stands for operations which must be performed to upgrade a DTM in the case of compatible data set format. Refer to chapter 3.6 and 6.25



<b>Use Case:</b>		Online Operation
<b>Brief Description:</b>		This use case includes all operations which are performed directly with the device.
<b>GUI:</b>		DTM-specific
<b>Print:</b>		DTM-specific
<b>Device Connection:</b>		Connected or disconnected
<b>Users:</b>	<b>Operator:</b>	No access
<b>UserLevel</b>	<b>Observer:</b>	No access
	<b>Operator:</b>	
	<b>Maintenance:</b>	Accessible
	<b>Planning Eng.:</b>	Fully accessible excluding OEM-specific parameter
<b>UserFlag</b>	<b>Administrator:</b>	No changes
	<b>OEM Service:</b>	Accessible with extended functionality (see below)
<b>Included Use Case:</b>		Online Functions
<b>Brief Description:</b>		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 sections like calibration or teach in.
<b>Included Use Case:</b>		Online Parameterization
<b>Brief Description:</b>		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.
<b>Included Use Case:</b>		Device-/Persistent Data Comparison
<b>Brief Description:</b>		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.
<b>Included Use Case:</b>		Calibration
<b>Brief Description:</b>		This use case invokes calibration procedures to adjust the input for e.g. pressure transmitters or the current for the output..
<b>Included Use Case:</b>		Plausibility Check
<b>Brief Description:</b>		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.
<b>UserLevel, UserFlag:</b>		The plausibility check may be more tolerant for actors of higher level.
<b>Included Use Case:</b>		Upload
<b>Brief Description:</b>		Reads the whole parameter set from the device into the Frame Application database. An upload started by an "OEM Service" actor may upload additional OEM parameters.
<b>GUI:</b>		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.
<b>Print:</b>		No support
<b>Device Connection:</b>		Needs a temporary connection
<b>Included Use Case:</b>		Download
<b>Brief Description:</b>		Like upload, but in the opposite direction.
<b>GUI:</b>		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.
<b>Print:</b>		No support
<b>Device Connection:</b>		Needs a temporary connection
<b>Included Use Case:</b>		Scan
<b>Brief Description:</b>		Retrieves the list of available devices via the IFdtSubtopology interface from a channel object. The channel object may be provided by a DTM or by Frame Application.
<b>GUI:</b>		The Frame Application may offer a GUI for representation of the list of

		devices.
<b>Print:</b>		No support
<b>Device Connection:</b>		Needs a temporary connection. (The channel must have online access.)

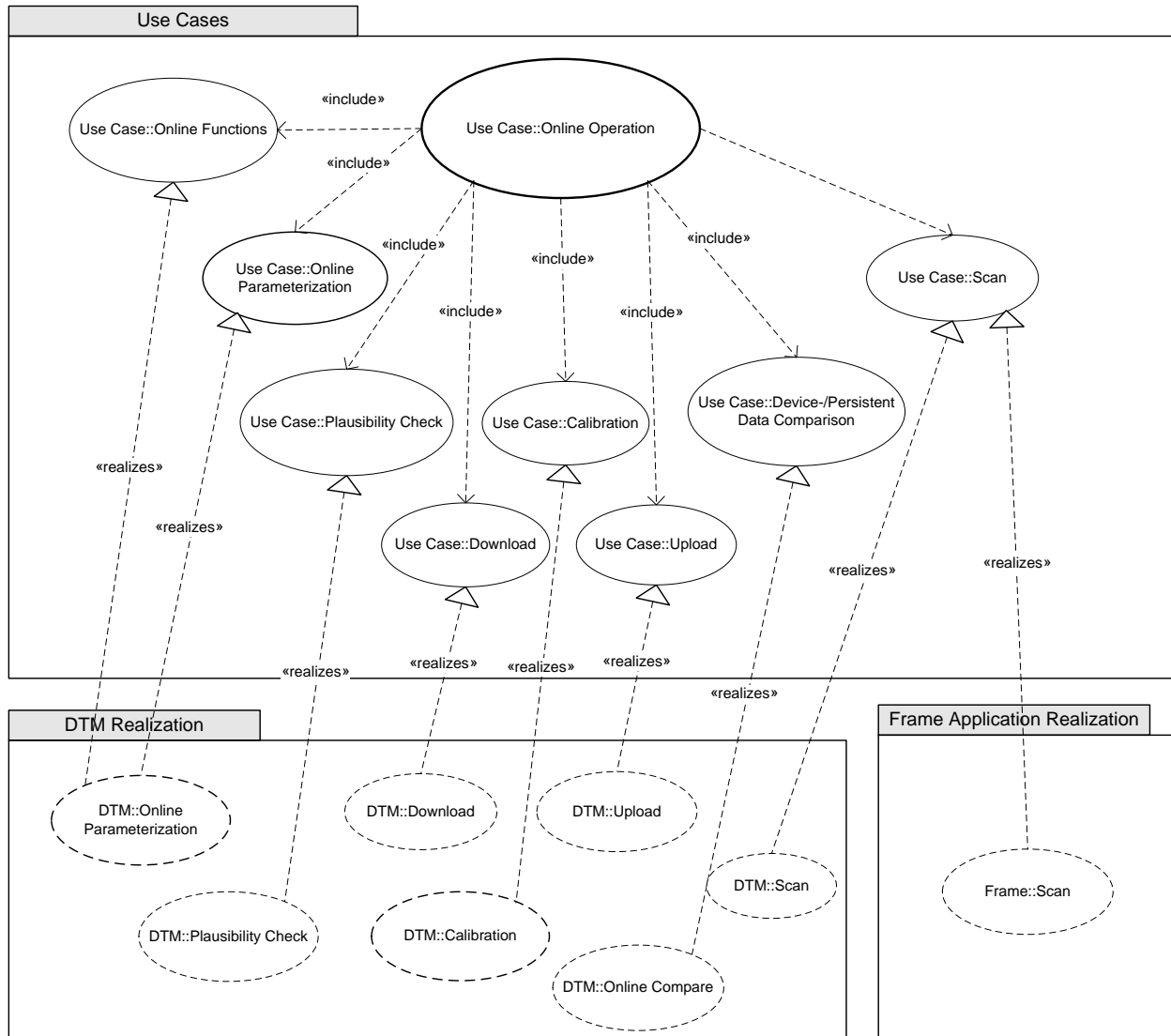


Figure 17: Realization of use case "Online Operation"

<b>Use Case:</b>		<b>Bulk Data Handling</b>
<b>Brief Description:</b>		This use case stands for the possibility to handle (e.g. up- and download, parameter adjustment or report generation) a group of instruments at once. This use case handles bulk data on system level.
<b>GUI:</b>		Frame Application
<b>Print:</b>		Not supported
<b>Device Connection:</b>		Needs a connection
<b>UserLevel</b>	<b>Observer:</b>	No access
	<b>Operator:</b>	
	<b>Maintenance:</b>	Accessible
	<b>Planning Eng.:</b>	
<b>UserFlags:</b>		No changes
<b>Included Use Case:</b>		Upload
<b>Brief Description:</b>		Reads the whole parameterization from the devices of the group into the Frame Application database.
<b>Included Use Case:</b>		Download
<b>Brief Description:</b>		Like upload, but in the opposite direction.

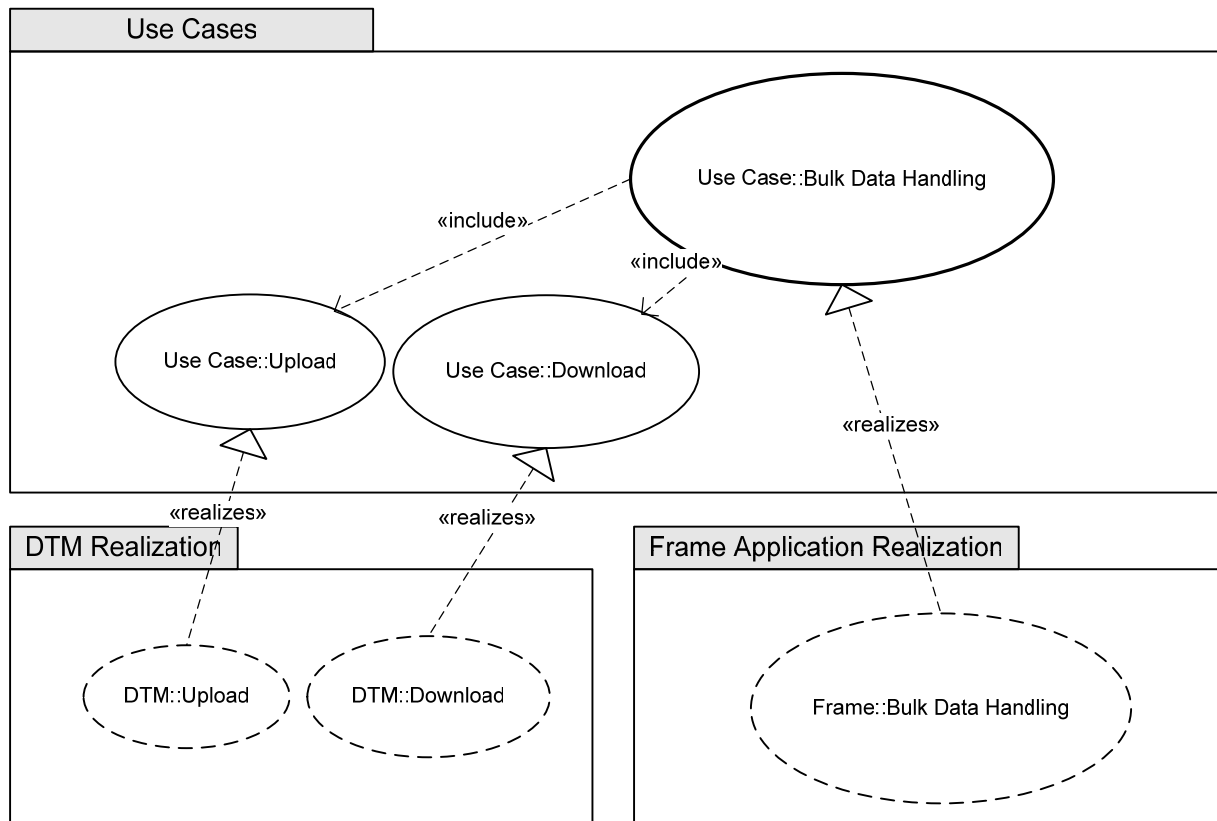


Figure 18: Realization of use case "Bulk Data Handling"

## 5.2.4 Planning

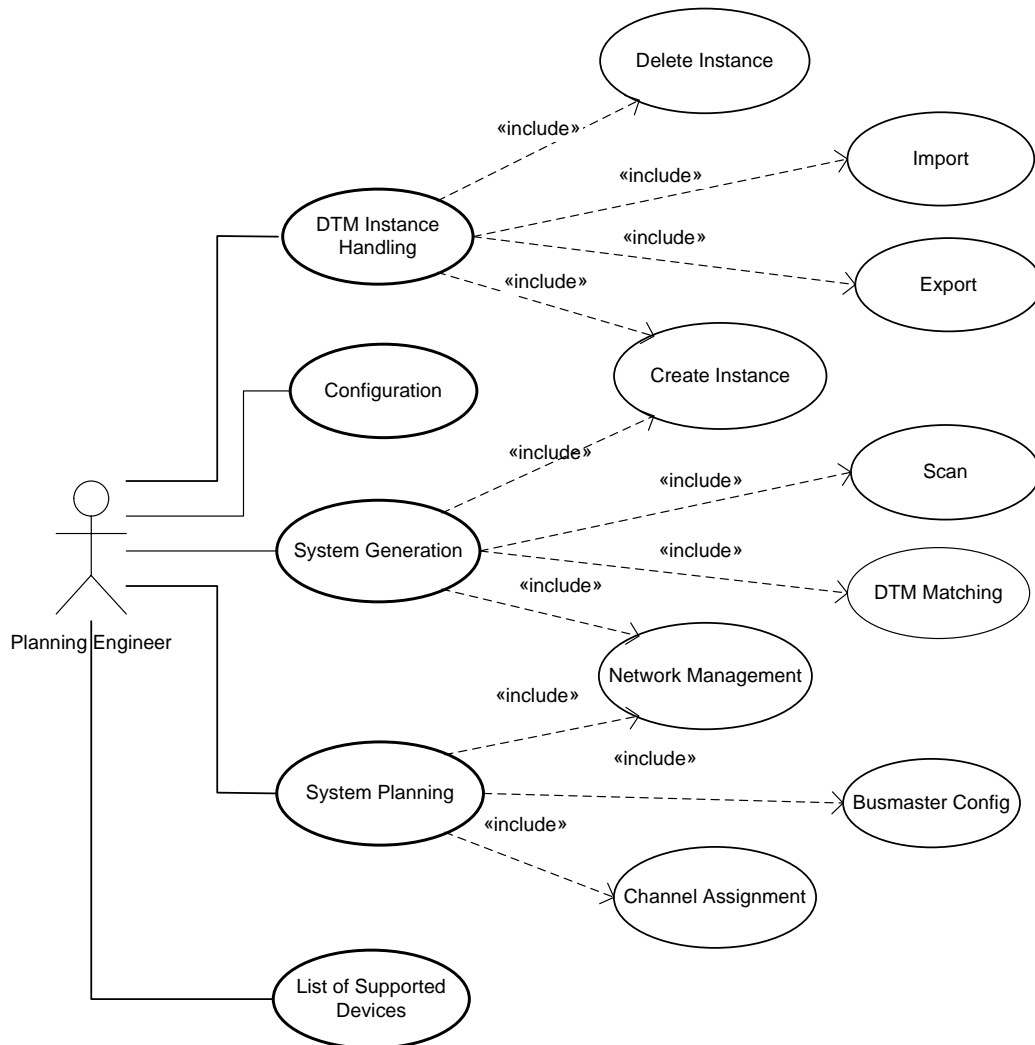


Figure 19: Use case - Planning Engineer

<b>Use Case:</b>		DTM Instance Handling
<b>Brief Description:</b>		With this use case a "Planning Engineer" actor can handle (E.g. instantiate, import, export, delete) a device instance in the Frame Application.
<b>GUI:</b>		Frame Application and DTM
<b>Print:</b>		Not supported
<b>Device Connection:</b>		Not necessary
<b>UserLevel</b>	<b>Observer:</b>	No access
	<b>Operator:</b>	
	<b>Maintenance:</b>	
	<b>Planning Eng.:</b>	Accessible
<b>UserFlags:</b>		No changes
<b>Included Use Case:</b>		Create Instance
<b>Brief Description:</b>		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 must be instantiated. This action is performed by the DTM.
<b>GUI:</b>		Frame Application and DTM (if it supports device default selection)
<b>Included Use Case:</b>		Import
<b>Brief Description:</b>		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.
<b>GUI:</b>		FA
<b>Included Use Case:</b>		Export
<b>Brief Description:</b>		To copy data from one device instance to another, the device instance data can be exported.
<b>GUI:</b>		FA
<b>Included Use Case:</b>		Delete Instance
<b>Brief Description:</b>		Deletion of a device instance
<b>GUI:</b>		FA

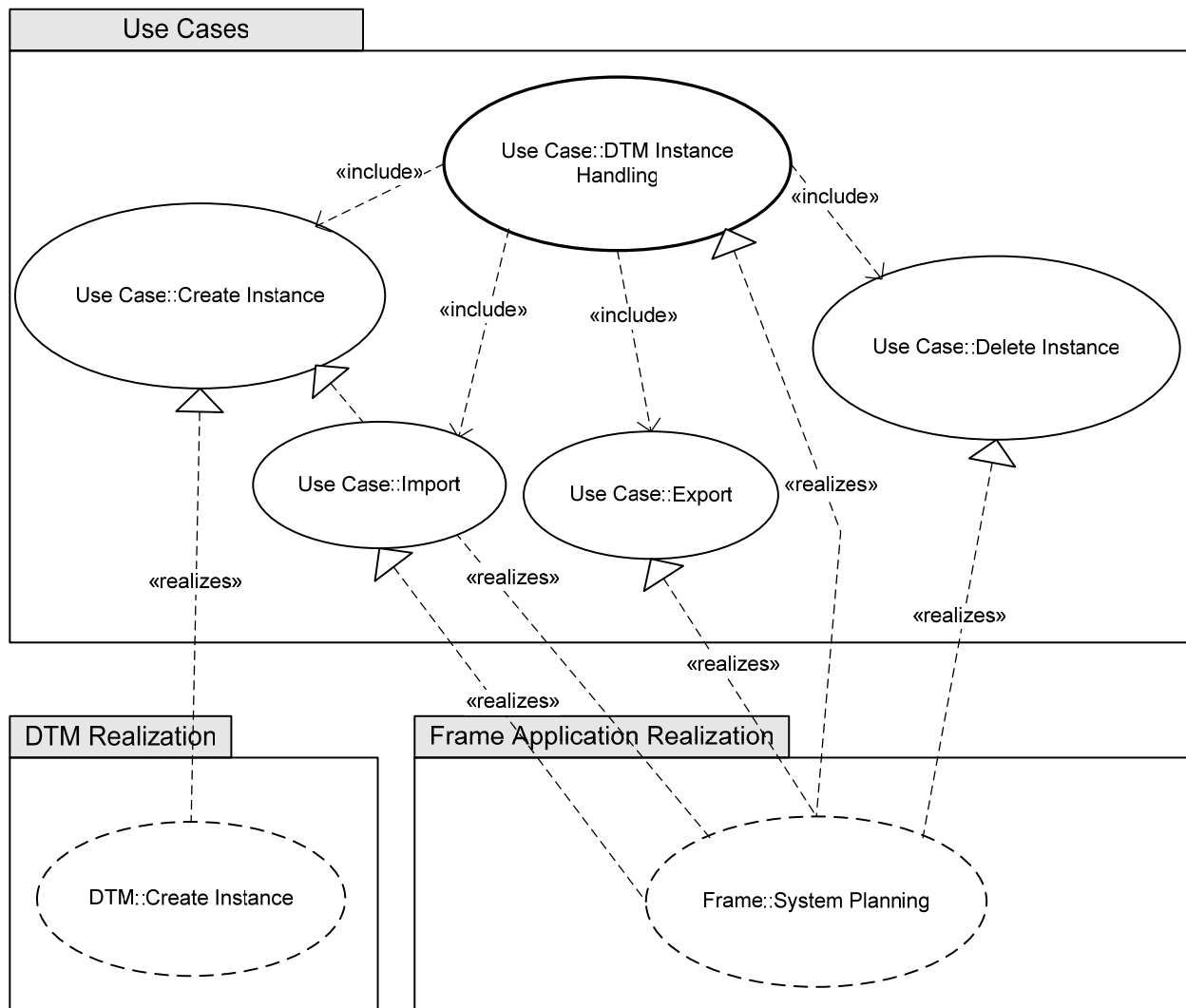


Figure 20: Realization of use case “DTM Instance Handling”

Use Case:		Configuration
<b>Brief Description:</b>		This use case enables the “ <a href="#">Planning Engineer</a> ” actor to configure a complex device.
<b>GUI:</b>		DTM
<b>Print:</b>		May be supported
<b>Device Connection:</b>		Must not have a connection established
<b>UserLevel</b>	<a href="#">Observer:</a>	No access
	<a href="#">Operator:</a>	
	<a href="#">Maintenance:</a>	
	<a href="#">Planning Eng.:</a>	Accessible
<b>UserFlags:</b>		No changes

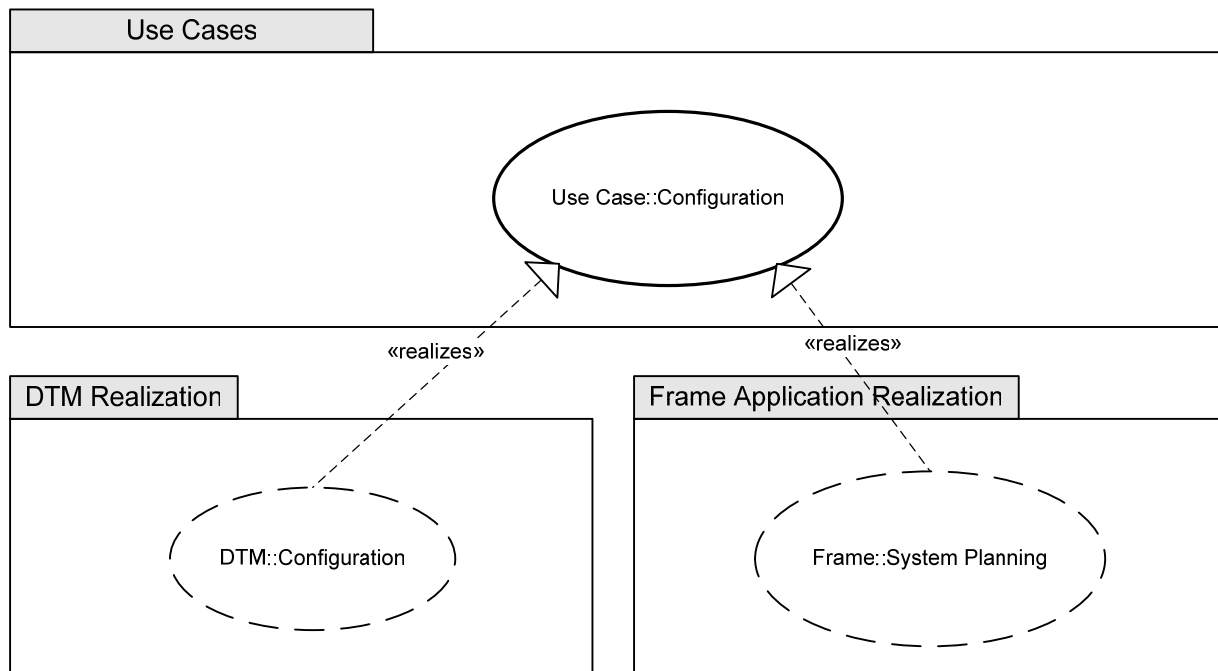


Figure 21: Realization of use case “Configuration”



<b>Use Case:</b>		System Generation
<b>Brief Description:</b>		Use case to perform all necessary actions for the creatuib of topology based on the result of scan.
<b>GUI:</b>		Frame Application and DTM
<b>Print:</b>		Frame Application and DTM
<b>Device Connection:</b>		Necessary
<b>UserLevel</b>	<b>Observer:</b>	No access
	<b>Operator:</b>	
	<b>Maintenance:</b>	
	<b>Planning Eng.:</b>	Accessible
<b>UserFlags:</b>		No changes
<b>Included Use Case:</b>		Scan
<b>Brief Description:</b>		Retrieves the list of available devices via the IFdtSubtopology interface from a channel object. The channel object may be provided by a DTM or by Frame Application.
<b>GUI:</b>		The Frame Application may offer a GUI for representation of the list of devices.
<b>Print:</b>		No support
<b>Device Connection:</b>		Needs a temporary connection. (The channel must have online access.)
<b>Included Use Case:</b>		Network Management
<b>Brief Description:</b>		Use case for network management purposes e.g. address setting as a function of a Communication-DTM.
<b>Included Use Case:</b>		DTM matching
<b>Brief Description:</b>		Use case for finding a DTM that matches best the information retrieved in the Scan use case.
<b>Included Use Case:</b>		Create Instance
<b>Brief Description:</b>		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 must be instantiated. This action is performed by the DTM.
<b>GUI:</b>		Frame Application and DTM (if it supports device default selection)

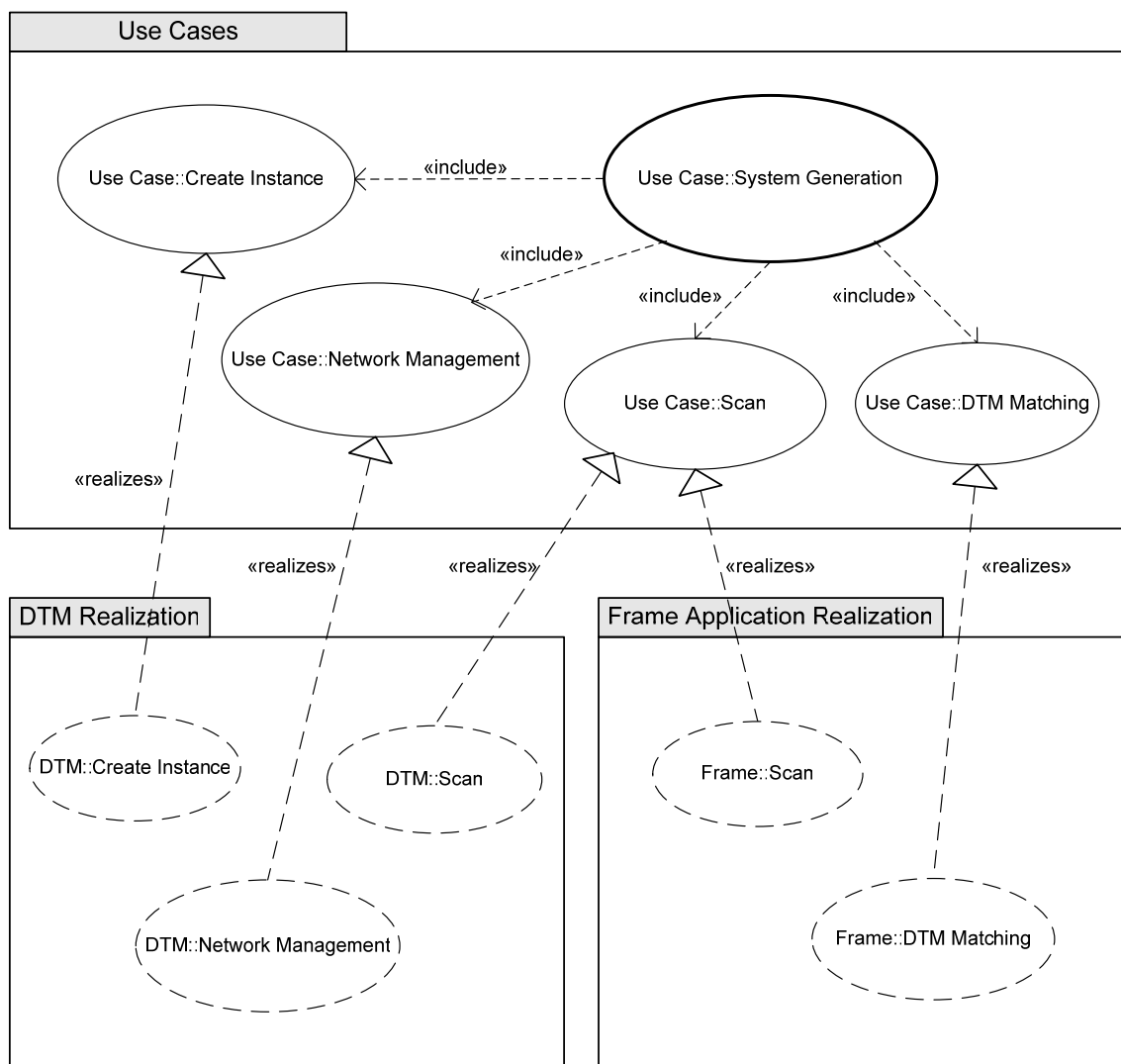


Figure 22: Realization of use case "System Generation"

<b>Use Case:</b>		<b>System Planning</b>
<b>Brief Description:</b>		Use case to perform all necessary actions for the editing of topology information.
<b>GUI:</b>		Frame Application and DTM
<b>Print:</b>		Frame Application and DTM
<b>Device Connection:</b>		Not necessary
<b>UserLevel</b>	<b>Observer:</b>	No access
	<b>Operator:</b>	
	<b>Maintenance:</b>	
	<b>Planning Eng.:</b>	Accessible
<b>UserFlags:</b>		No changes
<b>Included Use Case:</b>		<b>Network Management</b>
<b>Brief Description:</b>		Use case for network management purposes e.g. address setting as a function of a Communication-DTM.
<b>Included Use Case:</b>		Bus master Configuration
<b>Brief Description:</b>		Use case for configuration of bus master DTMs
<b>Included Use Case:</b>		Channel Assignment
<b>Brief Description:</b>		Use case for editing the FDT channel assignments

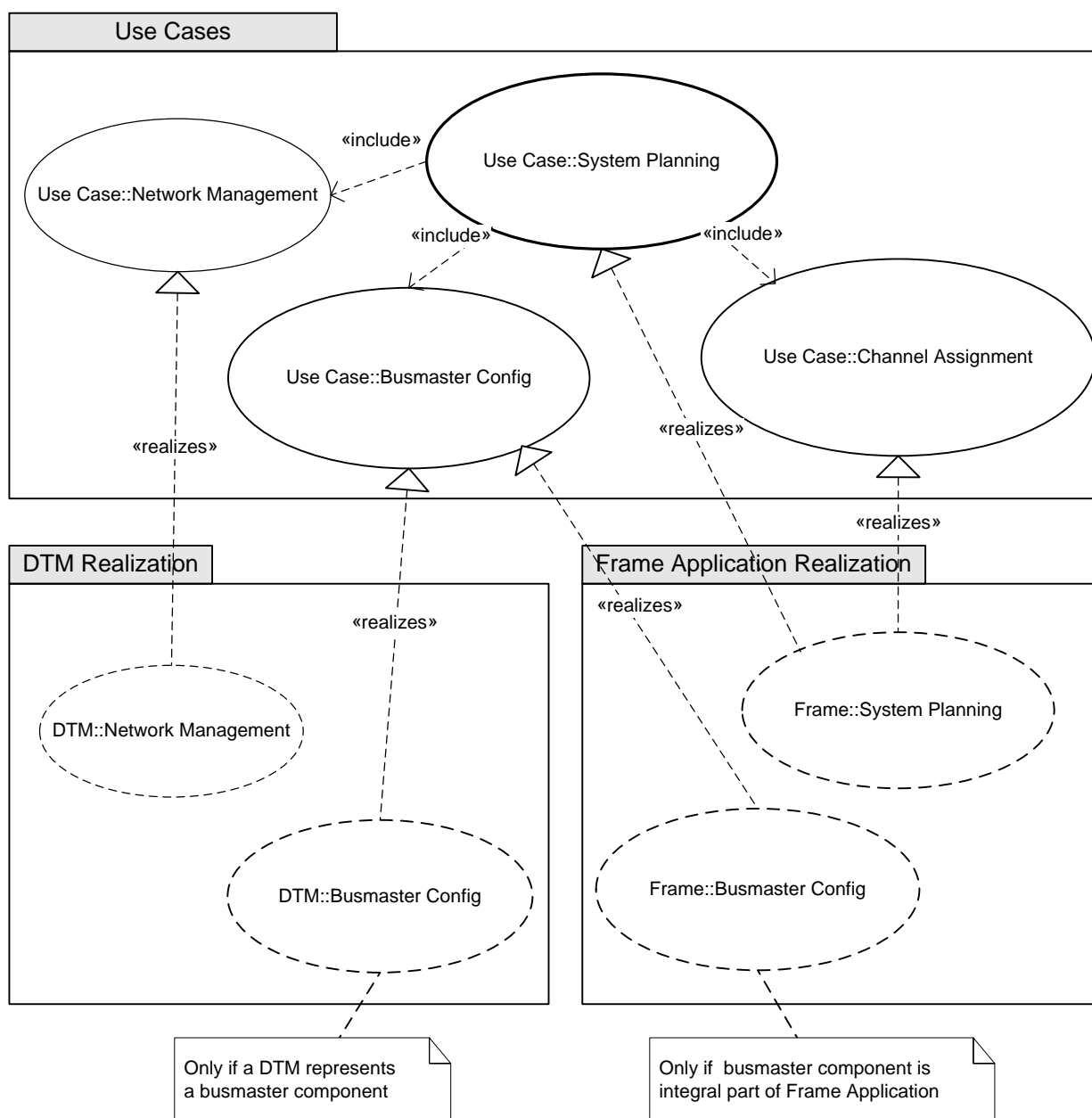


Figure 23: Realization of use case "System Planning"

Use Case:		List of Supported Devices
<b>Brief Description:</b>		In this use case a list of all supported devices within a project can be viewed and edited. A “ <b>Planning Engineer</b> ” actor can select a device from this list to create a new device instance. An actor with the “ <b>Administrator</b> ” actor flag set has access to change this list.
<b>GUI:</b>		FA
<b>Print:</b>		No support
<b>Device Connection:</b>		No connection
<b>UserLevel</b>	<b>Observer:</b>	No access
	<b>Operator:</b>	
	<b>Maintenance:</b>	
	<b>Planning Eng.:</b>	Accessible for viewing only
<b>UserFlag</b>	<b>Administrator:</b>	Accessible for editing
	<b>OEM Service:</b>	No changes

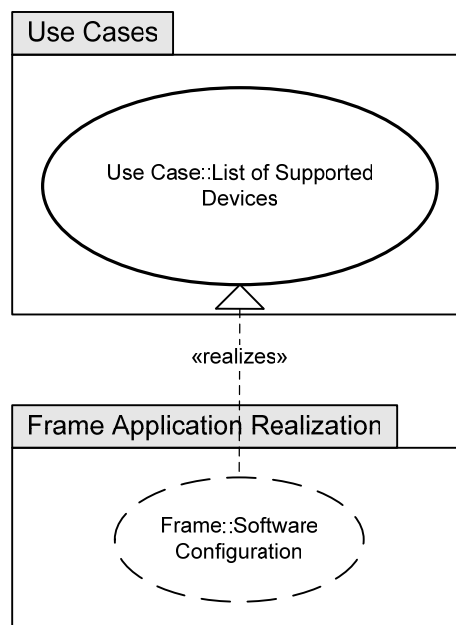


Figure 24: Realization of use case “List of Supported Devices”

## 5.2.5 OEM Service

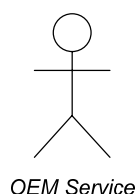


Figure 25: Use case - OEM Service

The “ **OEM Service**” actor may only have extended access to use cases of other actor roles.

## 5.2.6 Administrator

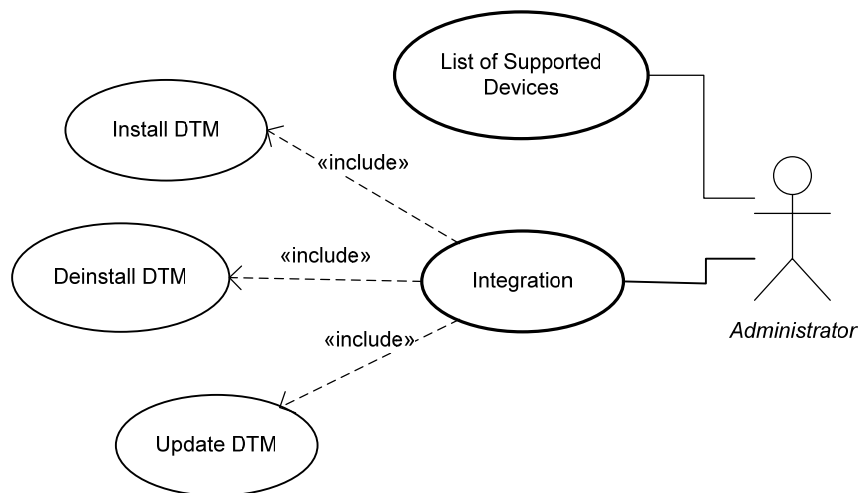


Figure 26: Use case – Administrator

Use Case: Integration	
<b>Brief Description:</b>	This use case enables the actor to install, deinstall or update new DTMs. Installation and deinstallation are not controlled by the Frame Application.
<b>GUI:</b>	The Frame Application may support the management of DTMs.
<b>Print:</b>	Not supported
<b>Device Connection:</b>	Must not have a connection established
<b>Users:</b>	Not accessible
Operator:	
Maintenance:	
Planning Eng.:	
Administrator:	
OEM Service:	Not accessible
Included Use Case: Install	
<b>Brief Description:</b>	Installation of a new DTM.
<b>GUI:</b>	Responsibility of the DTM
Included Use Case: Deinstall	
<b>Brief Description:</b>	dito
<b>GUI:</b>	responsibility of the DTM
Included Use Case: Update DTM	
<b>Brief Description:</b>	dito
<b>GUI:</b>	responsibility of the DTM

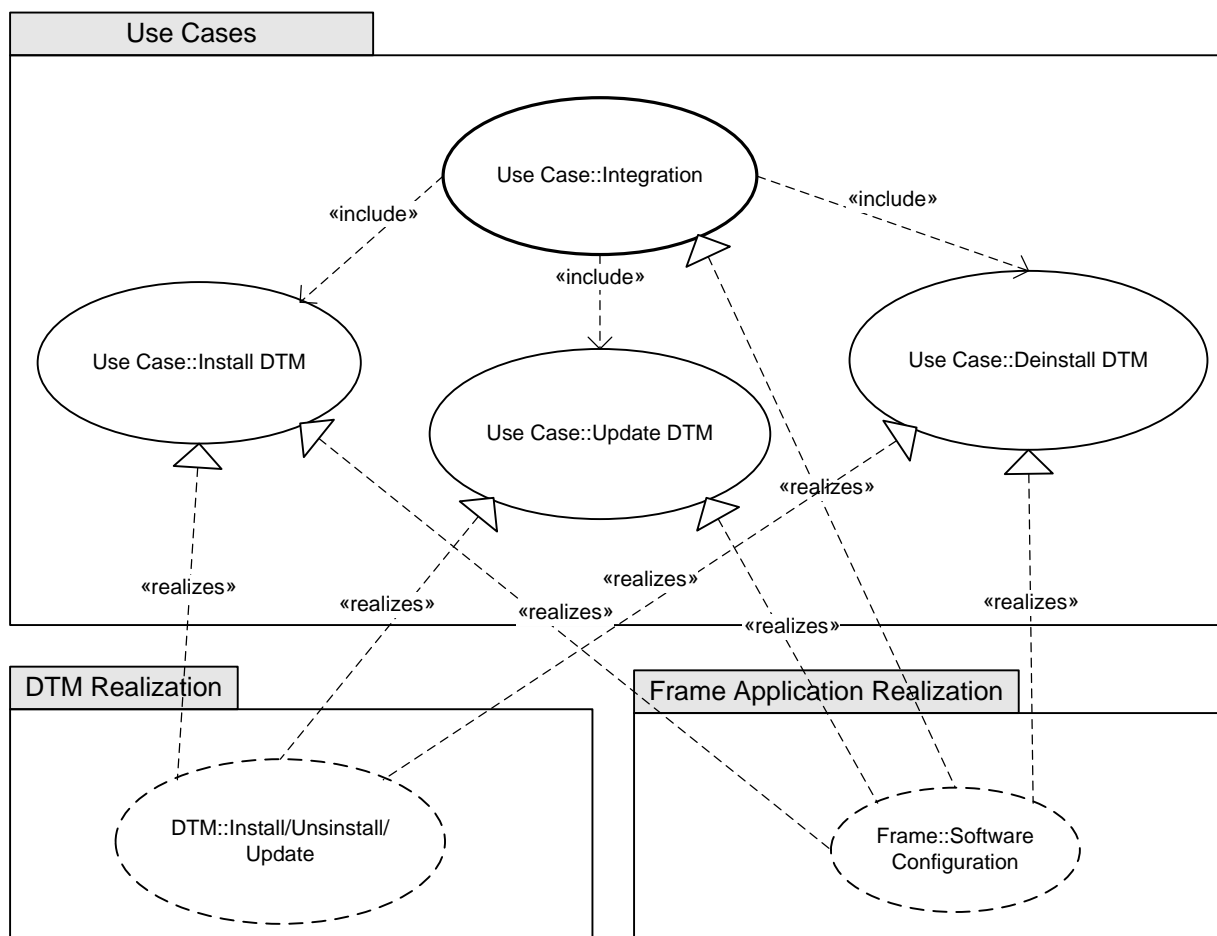


Figure 27: Realization of use case "Integration"

### 5.3 DTM Use Case Realization

As shown in the preceding sections, all Use Cases are realized operations which are part of a DTM or Part of a Frame Application. In this section the operations contributing to the realization of the DTM related Use Cases are explained.

The operations of the DTM may contain a graphical interface as well as a print function. Refer to 5.2

The following description of the operations elements of the DTM is presented in tabular form. Each table contains the following entries:

- Brief Description: Brief description of the operations
- Extends Use Case(s): Listing of all Use Cases whose realization involves this operation.
- Invocation: Description showing how this operation is called
- Application ID: Identification for application context.

<b>Operation:</b>	Adjust SetValue
<b>Brief Description:</b>	Enables the actor to adjust the SetValue of a device (e.g. positioner, controllers).
<b>Realizes Use Case(s):</b>	<a href="#">Adjust SetValue</a>
<b>Invocation:</b>	The Adjust-SetValue-Operation is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtAdjustSetValue

<b>Operation:</b>	Archive View
<b>Brief Description:</b>	A DTM may provide an archive for historical data. This realization element is for viewing the archived data
<b>Realizes Use Case(s):</b>	Archive
<b>Invocation:</b>	The Archive View is called directly by the Frame Application
<b>Application ID:</b>	-- not supported --

<b>Operation:</b>	Asset Management
<b>Brief Description:</b>	The contents of the Asset Management have not been defined yet.
<b>Realizes Use Case(s):</b>	<a href="#">Asset Management</a>
<b>Invocation:</b>	The Asset-Management-Operation is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtAssetManagement

<b>Operation:</b>	Audit Trail
<b>Brief Description:</b>	The Audit Trail Operation (see bulk-audit-trail) is used for the realization of a device-specific control and documentation of changes in the device parameters.
<b>Realizes Use Case(s):</b>	<a href="#">Device-Specific Audit Trail</a>
<b>Invocation:</b>	Is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtAuditTrail

<b>Operation:</b>	Bus Master Configuration
<b>Brief Description:</b>	GUI for configuration of the bus master.
<b>Realizes Use Case(s):</b>	<a href="#">Bus Master Configuration</a>
<b>Invocation:</b>	Is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtManagement

<b>Operation:</b>	Calibration
<b>Brief Description:</b>	Calibration procedures to adjust the input for e.g. pressure transmitters or the current for the output.
<b>Realizes Use Case(s):</b>	<a href="#">Online Functions</a>
<b>Invocation:</b>	Is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtCalibration

<b>Operation:</b>	Configuration
<b>Brief Description:</b>	The configuration operation lets a user define the structure of complex devices. Configuration is used online and offline.
<b>Realizes Use Case(s):</b>	<a href="#">Configuration</a>
<b>Invocation:</b>	is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtConfiguration



<b>Operation:</b>	Create Instance
<b>Brief Description:</b>	This operation generates the data set a DTM requires for the management of a device and feeds it with preset data.
<b>Realizes Use Case(s):</b>	<a href="#">Create Instance</a>
<b>Invocation:</b>	is called directly by the Frame Application
<b>Application ID:</b>	-- not supported --

<b>Operation:</b>	Delete Instance
<b>Brief Description:</b>	Deletes the data
<b>Realizes Use Case(s):</b>	<a href="#">Delete Instance</a>
<b>Invocation:</b>	is called directly by the Frame Application
<b>Application ID:</b>	-- not supported --

<b>Operation:</b>	Diagnosis
<b>Brief Description:</b>	This operation realizes all diagnostic methods a DTM provides for a device check.
<b>Realizes Use Case(s):</b>	<a href="#">Online Status</a>
<b>Invocation:</b>	is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtDiagnosis

<b>Operation:</b>	Download
<b>Brief Description:</b>	The Download Operation writes the current parameterization to the field device.
<b>Realizes Use Case(s):</b>	<a href="#">Download</a>
<b>Invocation:</b>	is called directly by the Frame Application
<b>Application ID:</b>	-- not supported --

<b>Operation:</b>	Force
<b>Brief Description:</b>	This operation allows the user to modify the device and change his parameters within a defined time limit. On termination of this operation at the least the device is reset to its original state.  This operation realizes for example the simulation of sensor output values.
<b>Realizes Use Case(s):</b>	<a href="#">Simulation</a>
<b>Invocation:</b>	Is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtForce

<b>Operation:</b>	Identify
<b>Brief Description:</b>	Displays identification of a device instance information e.g. address, TAG, Fieldbus-Rev., DD-Rev., Firmware. These information is protocol dependant. It also may contain device type specific information. Further information may be provided: references to documentation, area to add comments to the device instance.
<b>Realizes Use Case(s):</b>	Parameter Set Online View
<b>Invocation:</b>	The Identify is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtIdentify

<b>Operation:</b>	Install/Uninstall/Update
<b>Brief Description:</b>	The installation and deinstallation of the DTM-Software forms part of the DTM-Operation. Its software-technical realization will however be carried out outside the proper DTM-Module by suitable installation software. The installation of DTM-Software does not necessarily also signify the notification in the Frame Application.
<b>Realizes Use Case(s):</b>	<a href="#">Install DTM</a> , <a href="#">Deinstall DTM</a> , <a href="#">Update DTM</a>
<b>Invocation:</b>	Can be started directly by the Frame Application or by the surface of the operating system.
<b>Application ID:</b>	-- not supported --

<b>Operation:</b>	Load Data Set
<b>Brief Description:</b>	DTM loads data from a storage with a compatible data set format. Additional check for data compatibility must be done by the DTM.
<b>Realizes Use Case(s):</b>	Upgrade
<b>Invocation:</b>	is called directly by the Frame Application
<b>Application ID:</b>	-- not supported --

<b>Operation:</b>	Login
<b>Brief Description:</b>	When an operation with OEM-specific functions is started by an actor of the “ <a href="#">OEM Service</a> ” user level, free accessibility to those functions can be enabled by another password inquiry. This password inquiry is realized by operation “ <a href="#">Login</a> ”. As in most cases OEM-specific functions only have an effect on online connected devices, it is possible to verify the password through the device and in addition make the device-internal functionality accessible. This type of login is active during the entire session.
<b>Realizes Use Case(s):</b>	<a href="#">DTM-Specific Login</a>
<b>Invocation:</b>	is called during other operations, when an “ <a href="#">OEM Service</a> ” actor accesses operations with OEM-specific function elements.
<b>Application ID:</b>	-- not supported --

<b>Operation:</b>	Main Operation
<b>Brief Description:</b>	Entry point for a combination of different functions of a DTM (e.g. combination of parametrization, configuration, diagnosis, measured values).
<b>Realizes Use Case(s):</b>	
<b>Invocation:</b>	The Main Operation is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtMainOperation

<b>Operation:</b>	Network Management
<b>Brief Description:</b>	Function of a Communication-DTM to manage the network information
<b>Realizes Use Case(s):</b>	Network Management
<b>Invocation:</b>	The Network Management is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtNetworkManagement

<b>Operation:</b>	Observe
<b>Brief Description:</b>	This operation includes all software elements which support the observation of a device without affecting the functions or the parameter set. The realization of an “ <a href="#">Online Trend</a> ” can be named as an example of such an operation.
<b>Realizes Use Case(s):</b>	<a href="#">Online Trend</a>
<b>Invocation:</b>	is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtObserve

<b>Operation:</b>	Offline Compare
<b>Brief Description:</b>	The Compare-Operation implements the comparison of offline (persistent, transient, projected, default) parameters of two instances. The Compare GUI of a DTM is switched (or extended) to Offline Compare using the <a href="#">IDtmDiagnosis::Compare()</a> function
<b>Realizes Use Case(s):</b>	<a href="#">Persistent Data Comparison</a>
<b>Invocation:</b>	is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtOfflineCompare

<b>Operation:</b>	Offline Parameterize
<b>Brief Description:</b>	This operation enables editing of the instance persistent data set.
<b>Realizes Use Case(s):</b>	<a href="#">Offline Parameterization</a>
<b>Invocation:</b>	is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtOfflineParameterize

<b>Operation:</b>	Online Compare
<b>Brief Description:</b>	The Compare-Operation implements the comparison of offline (persistent, transient, projected, default) parameters with online parameters of the device. A DTM may also permit a comparison with standard configurations or previously exported configurations of other devices through access to intrinsic databases.
<b>Realizes Use Case(s):</b>	<a href="#">Device-/Persistent Data Comparison</a>
<b>Invocation:</b>	is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtOnlineCompare

<b>Operation:</b>	Online Parameterization
<b>Brief Description:</b>	This operation enables the editing of parameters directly on the device. When this operation starts, the current parameterization of the device is read out and made available to the user for editing. Modified parameters are immediately entered into the device on successful completion of the plausibility check of the data set. The edit function is disabled for actor “ <a href="#">Operator</a> ” in order to realize the “ <a href="#">Parameter Set Online View</a> ” Use Case.
<b>Realizes Use Case(s):</b>	<a href="#">Online Parameterization</a> , <a href="#">Online Functions</a> , <a href="#">Parameter Set Online View</a>
<b>Invocation:</b>	is called directly by the Frame Application
<b>Application ID:</b>	FDTApplicationIdSchema:fdtOnlineParameterize

<b>Operation:</b>	Parameter Access
<b>Brief Description:</b>	DTM provides read and write access via SingleDataAccess interfaces
<b>Realizes Use Case(s):</b>	Replacement
<b>Invocation:</b>	is only called indirectly via other operations
<b>Application ID:</b>	-- not supported --

<b>Operation:</b>	Plausibility Check
<b>Brief Description:</b>	Operation “ <a href="#">Plausibility Check</a> ” conducts a consistency check for all parameters of the device. Only offline data is contained in the consistency check. This operation works as partial realization of other operations of the DTM
<b>Realizes Use Case(s):</b>	<a href="#">Plausibility Check</a>
<b>Invocation:</b>	is only called indirectly via other operations
<b>Application ID:</b>	-- not supported --

<b>Operation:</b>	Print
<b>Brief Description:</b>	Print Operation stands for an action in which a report is created for current view following the generation of a printout. The report generation is an essential part of all operations during which data is edited or produced. The print operation therefore contributes to the part realization of almost all operations also implying a direct call of the GUI Operations.
<b>Realizes Use Case(s):</b>	The print operation participates in the realization of several operations of the DTM. Every application context a DTM supports with a GUI, should also support printing.
<b>Invocation:</b>	If necessary, it should be possible to start the Print Operation from the DTM's GUI. In this case the Print Operation is directly called by the DTM. The Frame Application starts the Print Operation of the DTM directly for the realization of the Use Cases “ <a href="#">Report Generation</a> ”. In this case the Frame Application invokes the documentation interface of the DTM.
<b>Application ID:</b>	-- not supported --

<b>Operation:</b>	Scan
<b>Brief Description:</b>	A channel provides a list of available devices.

<b>Operation:</b>	Upload
<b>Brief Description:</b>	Loads the current configuration from the device to the device instance data set of the DTM
<b>Realizes Use Case(s):</b>	<a href="#">Upload</a>
<b>Invocation:</b>	is called directly by the Frame Application
<b>Application ID:</b>	-- not supported --

## 5.4 Frame Application Use Case Realization

This section describes any operation the Frame Application must provide for the realization of the FDT Uses Cases. The operations are again listed in tabular form and introduced in a brief description explaining their function in the context of the FDT-concept.

<b>Operation:</b>	GUI
<b>Brief Description:</b>	This main operation logically links all GUIs the Frame Application realizes into an element of the software concept. Also in this case, two types of realization are viewed like in the realization of the GUI operation.

<b>Operation:</b>	Print
<b>Brief Description:</b>	The print operation is a central part in the Frame Application which realizes Frame Application-internal print operations and communicates with the interfaces in order to print DTM-specific data.
<b>Operation:</b>	Archive View
<b>Brief Description:</b>	A Frame Application may provide an archive for storing sensor data for example. The realization of display and analysis of archive data is performed by this operation.
<b>Operation:</b>	Asset Management
<b>Brief Description:</b>	Frame Application provides asset management functions.
<b>Operation:</b>	Audit Trail
<b>Brief Description:</b>	Frame Application provides mechanism for storage and display of audit trail information.
<b>Operation:</b>	Bulk Data Handling
<b>Brief Description:</b>	Frame Applications provide an option to apply individual functions like the Upload and Download of groups of devices. This operation realizes the parameterization and management of a wide range of devices.
<b>Operation:</b>	Busmaster Config
<b>Brief Description:</b>	If Frame application provides access to Fieldbus communication this operation allows configuration of the communication.
<b>Operation:</b>	Data Format Association
<b>Brief Description:</b>	Frame Application manages the format information of stored data sets and is able to associate DTM to the data sets. Refer to chapter 6.25.2
<b>Operation:</b>	DTM matching
<b>Brief Description:</b>	Frame Application compares the results from Scan to the information that is available in the Device Catalogue and finds the correct DTM for a physical device. Refer to chapter 6.2.1.1
<b>Operation:</b>	Login
<b>Brief Description:</b>	Before an actor is entitled to work within the Frame Application and one of his/her DTMs, his/her user role must have been specified and verified. Contrary to the <b>Device-Specific Login</b> the user's role can also remain valid for several sessions.
<b>Operation:</b>	Parameter Migration
<b>Brief Description:</b>	Frame Application provides mechanism to copy data from one DTM instance to an other DTM instance. This mechanism is handled with SingleDataAccess interfaces and may need additional user interaction.

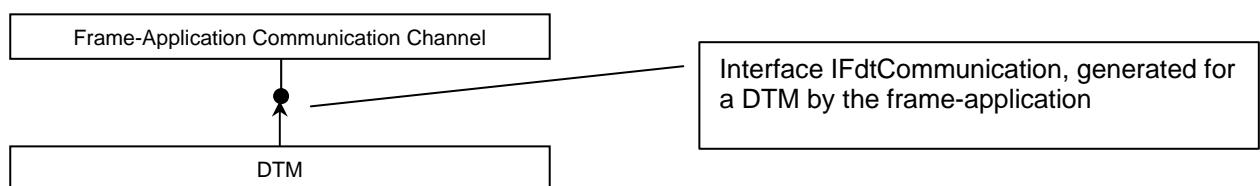
<b>Operation:</b>	Repair Device
<b>Brief Description:</b>	This use case stands for operations which must be performed to repair or change a device. Example: A Frame Application supports a temporary deletion of a device with automatically parameterization download when the device has been reinstalled.
<b>Operation:</b>	Software Configuration
<b>Brief Description:</b>	The configuration operation lets a user define the structure of complex devices. Configuration is used online and offline.
<b>Operation:</b>	System Planning
<b>Brief Description:</b>	The operation element is responsible for bracketing all functions that the Frame Application must provide in order to define the data structure, the communication and the communication links of the entire system.

## 6 FDT Sequence Charts

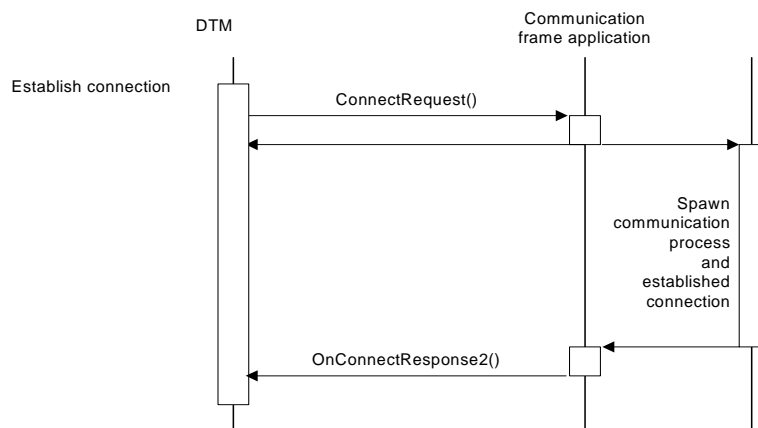
### 6.1 DTM Peer To Peer Communication

For a DTM each connection is established as a peer-to-peer connection. This chapter describes the communication function calls from a DTM developer's point of view.

#### 6.1.1 Establish a Connection between DTM and Device



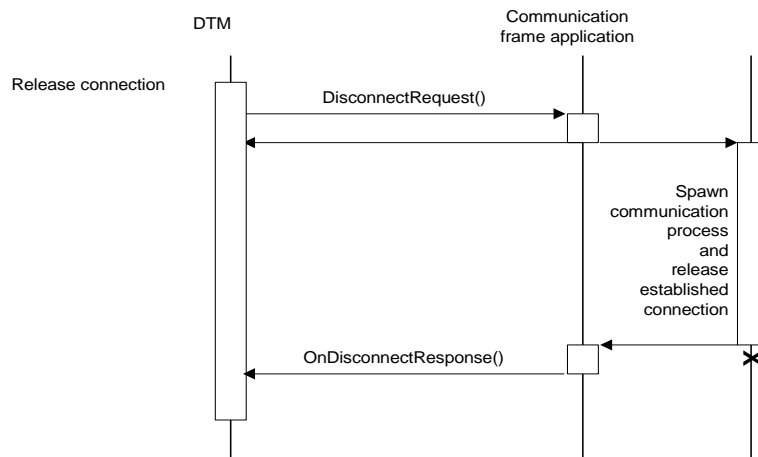
#### 6.1.2 Asynchronous Connect



#### Used methods:

[IFdtCommunication::ConnectRequest\(\)](#)  
[IFdtCommunicationEvents2::OnConnectResponse2\(\)](#)

### 6.1.3 Asynchronous Disconnect



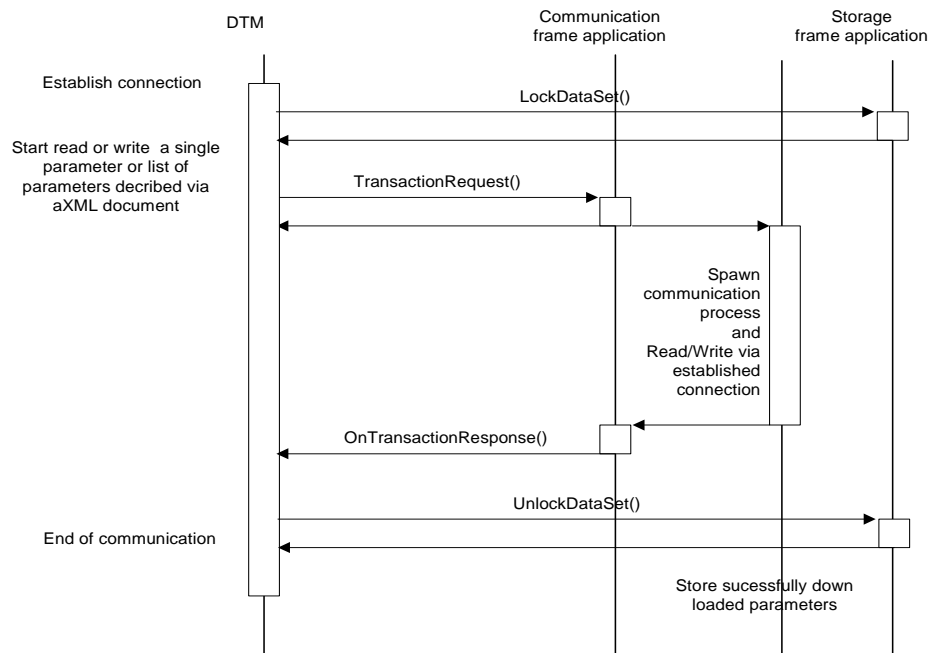
**Used methods:**

[IFdtCommunication::DisconnectRequest\(\)](#)

[IFdtCommunicationEvents::OnDisconnectResponse\(\)](#)



## 6.1.4 Asynchronous Transaction



### Used methods:

`IFdtCommunication::TransactionRequest()`

`IFdtCommunicationEvents::()`

`IFdtContainer::LockDataSet()`

`IFdtContainer::UnlockDataSet()`

## 6.2 Nested Communication

This chapter is important for DTM developers who support a device with gateway functionality (e.g. Remote I/O). This chapter describes the communication function calls from the point of view of a developer of a communication component.

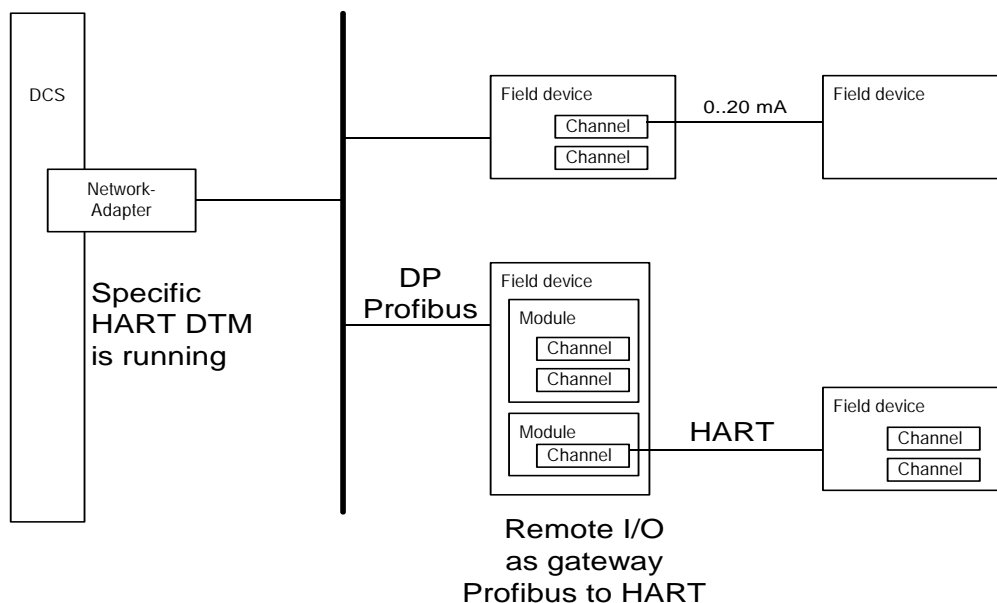
Nested communication is used to establish the connection to a device on a sub system. For example, a DTM calls a field device which is connected to a channel of a Remote I/O.

The requirement to this architecture is that a DTM must not know anything about the kind of the overlaying system. Nevertheless, the structure of the sub system is well known to Frame Application and DTMs.

The DTMs which have gateway functionality (Remote I/O) have to provide an FdtChannel with communication interfaces for each channel with gateway functionality.

Furthermore always the parent ( DTM with gateway functionality or, at least, the Frame Application) is responsible for the communication addressing of its sub-devices. Therefore it has to set parameters like 'tag' and 'BusInformation' according to the communication protocol. (see also: [IDtmParameter::SetParameters\(\)](#))

System-topology for the following examples:



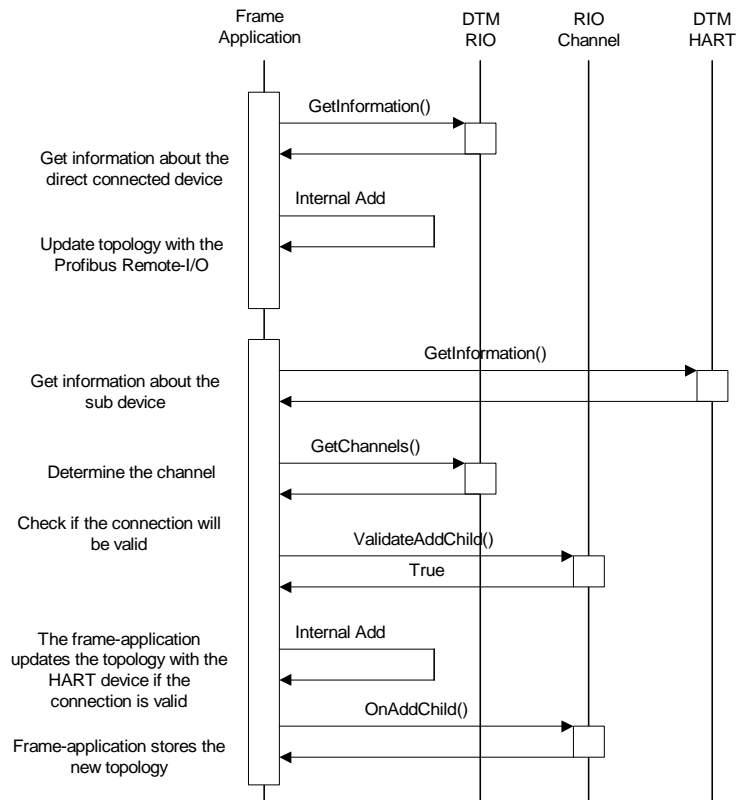
### 6.2.1 Generate Systemtopology

Following information reflects this topology:

- The instance data set of the HART-Device
- The instance data set of the Remote-I/O
- The reference of the data sets HART-Device to Remote-I/O

### 6.2.1.1 Frame Applications point of view

The Frame Application is responsible to generate and manage the topology. The sequence shows how the Frame Application manages the relation between DTMs and Communication-Channels.



#### **Used methods:**

`IDtmInformation::GetInformation()`

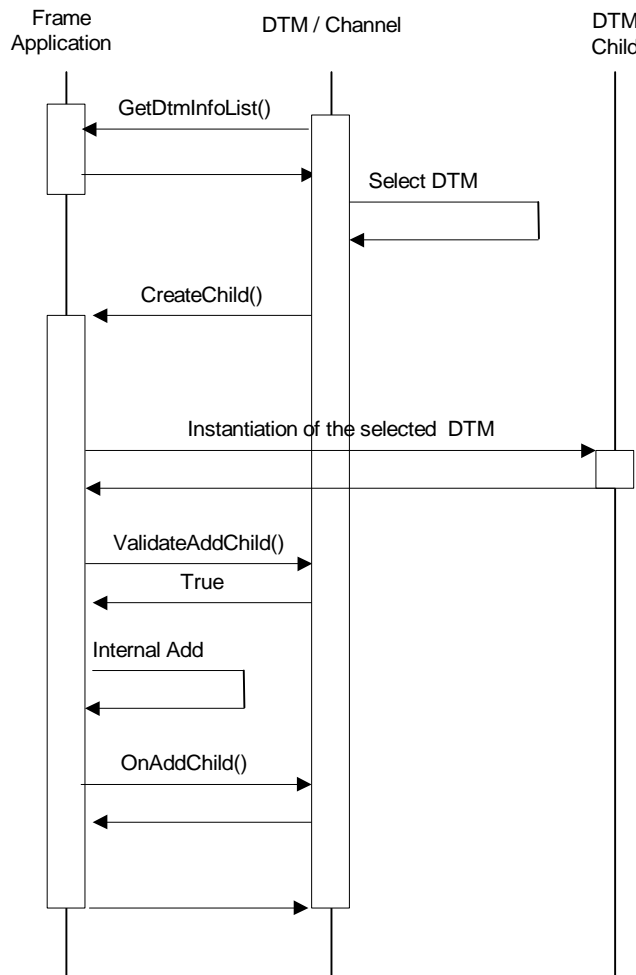
`IDtmChannel::GetChannels()`

`IFdtChannelSubTopology::ValidateAddChild()`

`IFdtChannelSubTopology::OnAddChild()`

## 6.2.1.2 DTMs point of view

This sequence shows the generation of the topology triggered by a DTM.



### **Used methods:**

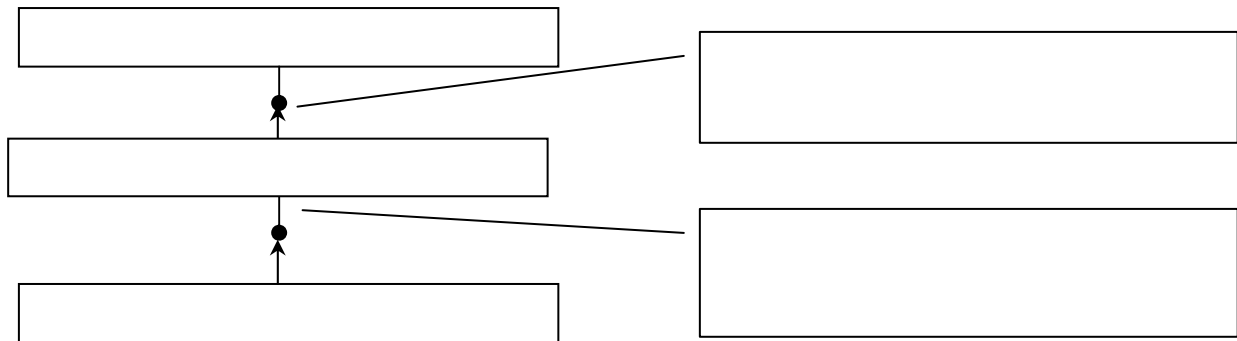
`IFdtTopology::GetDtmInfoList()`

`IFdtTopology::CreateChild()`

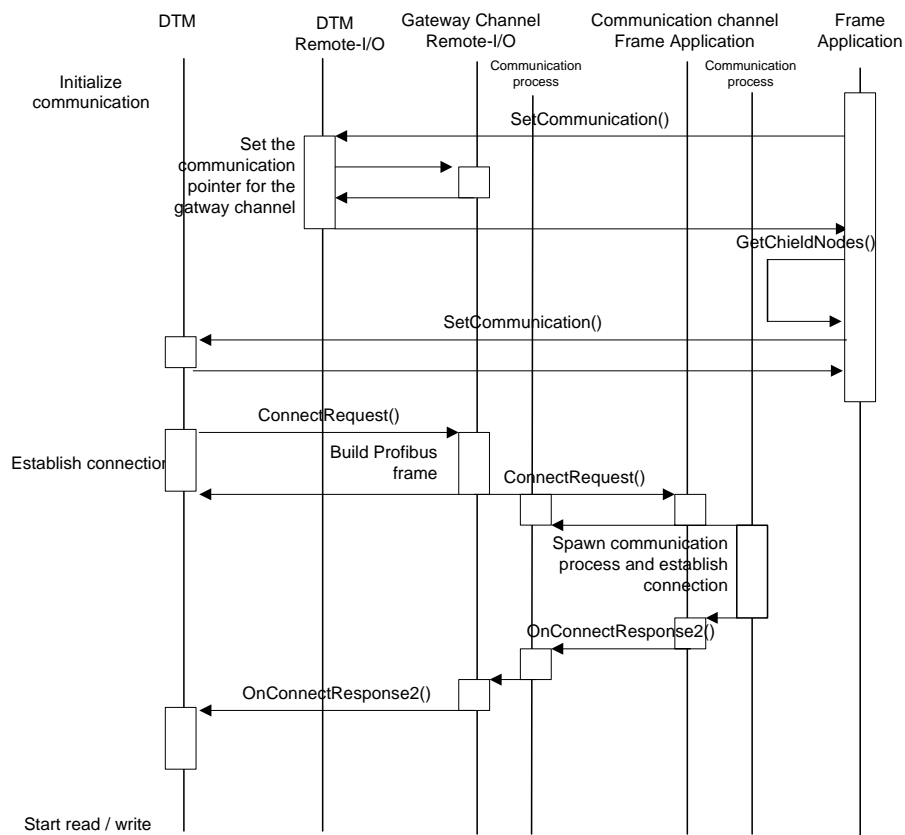
`IFdtChannelSubTopology::ValidateAddChild()`

`IFdtChannelSubTopology::OnAddChild()`

## 6.2.2 Establish a Connection between DTM and Device



The topology information must be available to create the proper communication interface hierarchy



### Used methods:

[IDtm::SetCommunication\(\)](#)

[IFdtTopology::GetChildNodes\(\)](#)

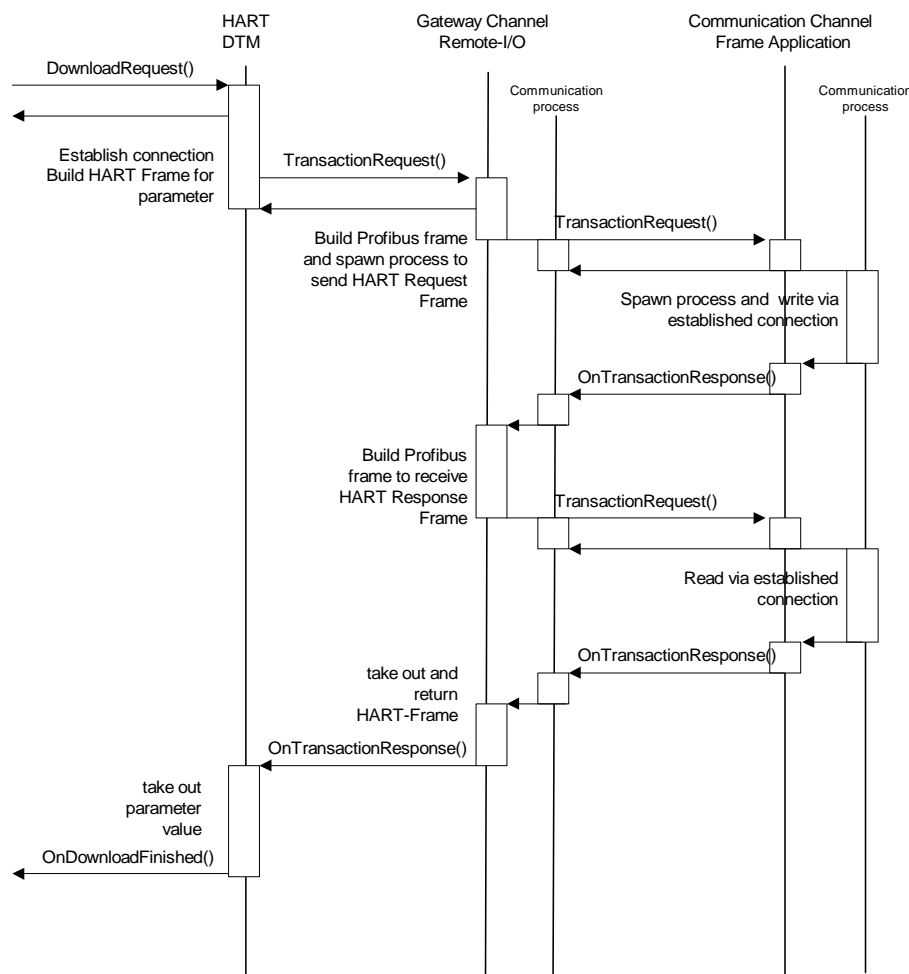
[IFdtCommunication::ConnectRequest\(\)](#)

[IFdtCommunicationEvents2::OnConnectResponse2\(\)](#)

### 6.2.3 Asynchronous Transaction

#### Example HART/PROFIBUS

The transaction-function-call of the HART DTM is realized as a read and write-function at the PROFIBUS communication component of the Remote I/O. With the PROFIBUS write function the communication component transfers the HART data to the HART Master at the Remote I/O. The answer of the HART device can be received from the HART Master by the according PROFIBUS read function call. The addressing for the read and write function call depends on the hardware and the configuration of the Remote I/O and is well known to the communication channel.



#### Used methods:

[IFdtCommunication::TransactionRequest\(\)](#)

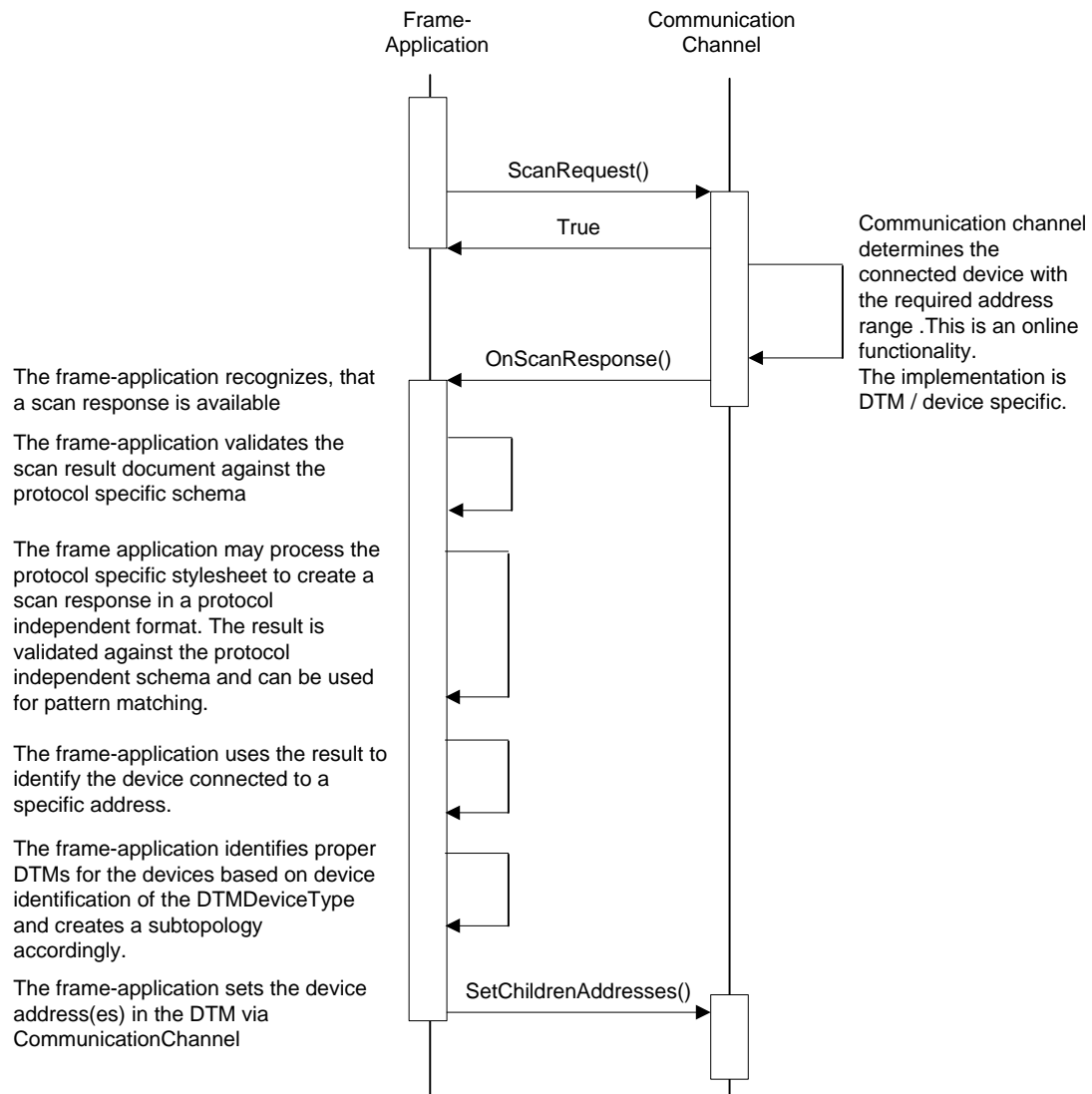
[IFdtCommunicationEvents::OnTransactionResponse\(\)](#)

[IDtmOnlineParameter::DownloadRequest\(\)](#)

[IDtmEvents::OnDownloadFinished\(\)](#)

## 6.3 Topology Scan

### 6.3.1 Scan Network



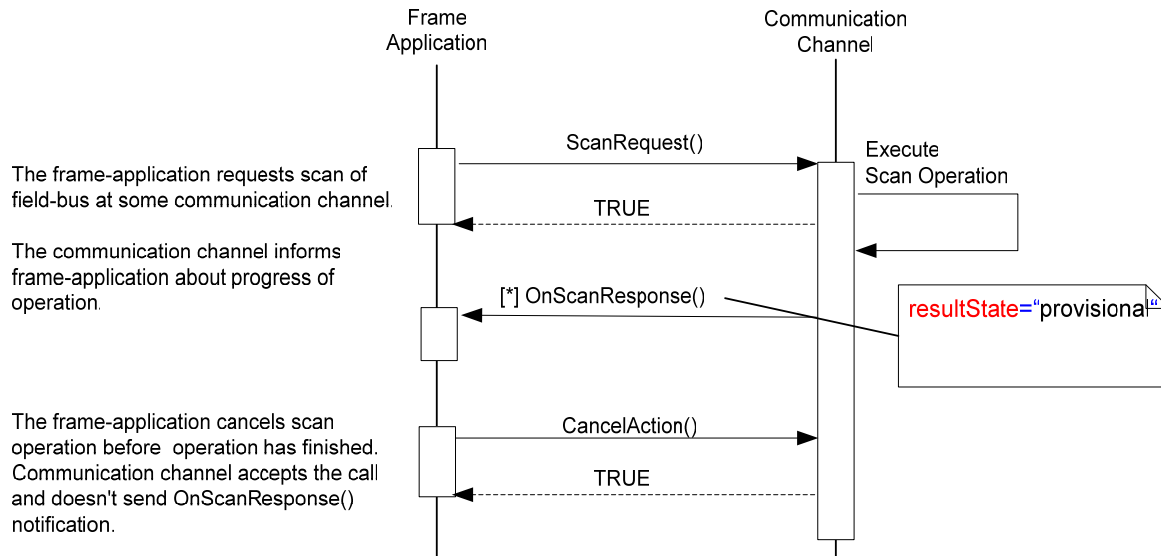
Sequence diagram: scan network topology

#### Used methods:

IFdtChannelScan::ScanRequest()  
 IDtmScanEvents::OnScanResponse()  
 IFdtChannelSubTopology2::SetChildrenAddresses()

### 6.3.2 Cancel Topology Scan

In this scenario Frame Application cancels an active scan request.



#### Used methods:

[IFdtChannelScan::ScanRequest\(\)](#)

[IFdtChannelScan:CancelAction\(\)](#)

[IDtmScanEvents::OnScanResponse\(\)](#)

#### Behavior

In case of problems it is up to the communication channel to handle problems in a way that the communication channel returns final response with an error information.

If the final response is not returned in expected time, user or Frame Application can cancel this action. No further `OnScanResponse()` should be called.

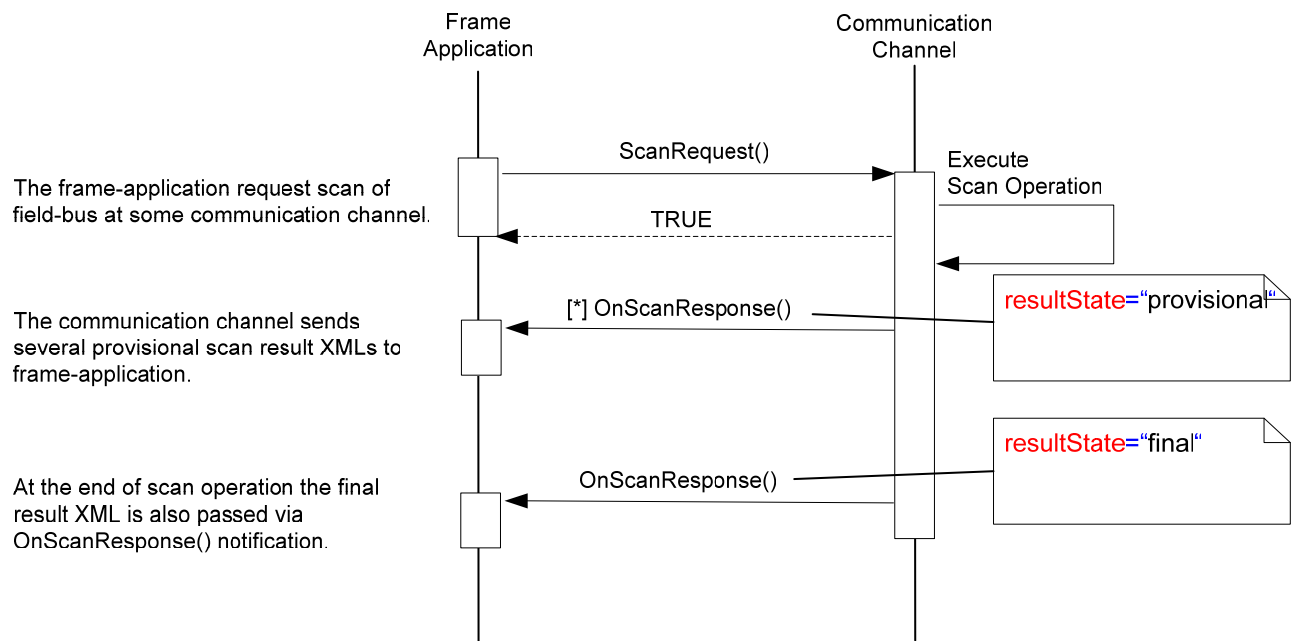
In case of an error in a provisional scan result, it is up to the Frame Application to cancel the scanning or to handle the particular error in the result and continue the scan.

Progress events must be fired by communication channel while scanning is performed.

### 6.3.3 Provisional Scan Result Notifications

In this scenario communication channel sends provisional TopologyScan result XMLs to the Frame Application.





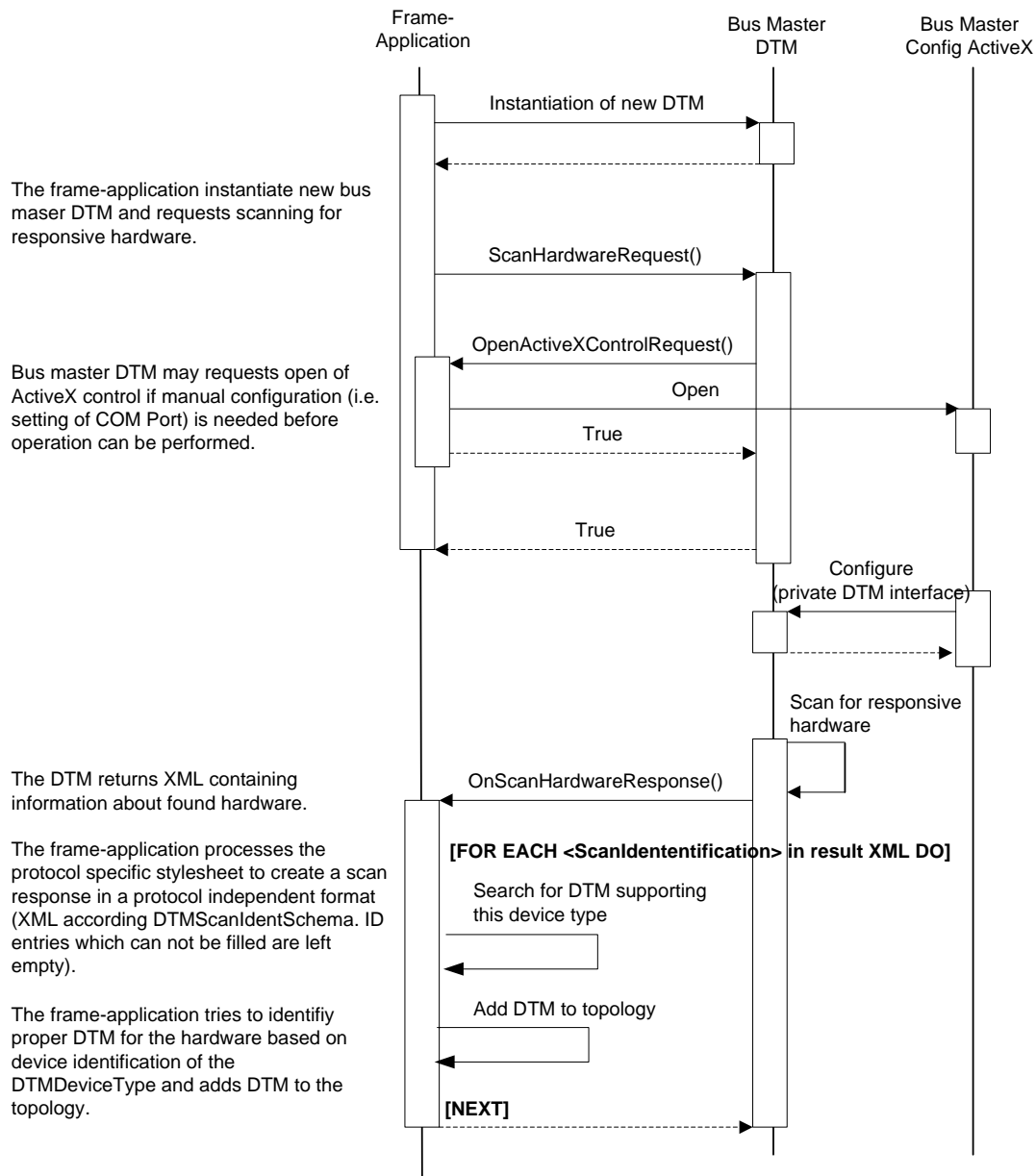
#### Used methods:

[IFdtChannelScan::ScanRequest\(\)](#)

[IDtmScanEvents::OnScanResponse\(\)](#)

### 6.3.4 Scan for Communication Hardware

Scan for communication hardware is needed for full automatic creation of a system topology out of an existing bus topology. Frame Application needs to check available communication hardware first and instantiate corresponding Bus Master DTM before scan for sub-topology is feasible.

**Used methods:**

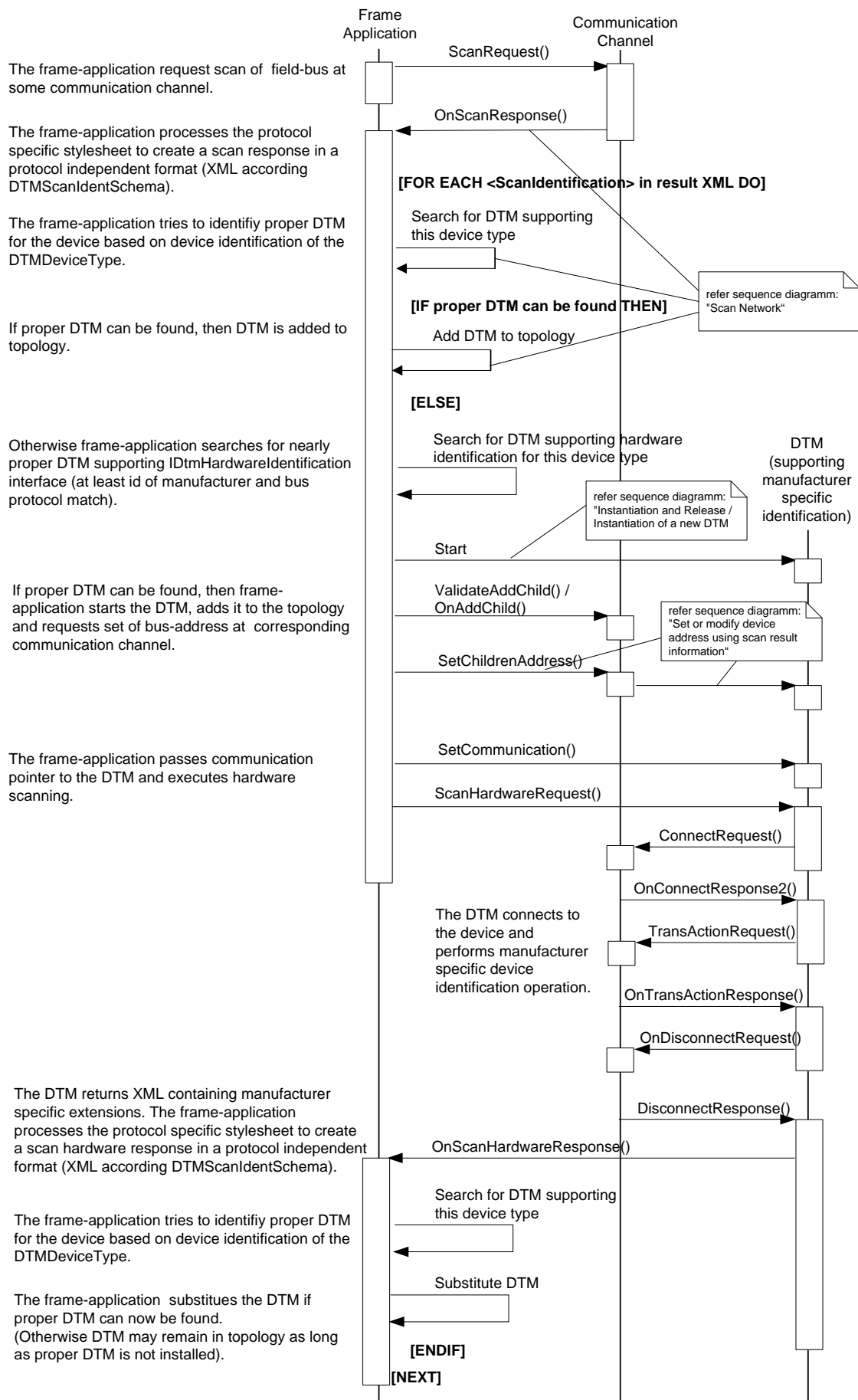
[IDtmHardwareIdentification::ScanHardwareRequest\(\)](#)

[IDtmScanEvents::OnScanHardwareResponse\(\)](#)

[IFdtActiveX::OpenActiveXControlRequest\(\)](#) (or [IFdtActiveX2::OpenDialogActiveXControlRequest\(\)](#))

### 6.3.5 Manufacturer specific Device Identification

In this scenario a Frame Application scans an existing field-bus network and uses DTM implementing IDtmHardwareIdentification interface to identify devices for which manufacturer specific operation must be performed.



**Used methods:**

[IDtmHardwareIdentification:ScanHardwareRequest\(\)](#)  
[IDtmScanEvents::OnScanHardwareResponse\(\)](#)  
[IDtm::SetCommunication\(\)](#)  
[IFdtChannelSubTopology:ValidateAddChild\(\)](#)  
[IFdtChannelSubTopology::OnAddChild\(\)](#)  
[IFdtChannelScan::ScanRequest\(\)](#)  
[IFdtChannelSubTopology2::SetChildrenAddresses\(\)](#)  
[IDtmScanEvents::OnScanResponse\(\)](#)  
[IFdtCommunication::ConnectRequest\(\)](#)  
[IFdtCommunication::TransActionRequest\(\)](#)  
[IFdtCommunication::DisconnectRequest\(\)](#)  
[IFdtCommunicationEvents2::OnConnectResponse2\(\)](#)  
[IFdtCommunicationEvents::OnTransactionResponse\(\)](#)  
[IFdtCommunicationEvents::OnDisconnectResponse\(\)](#)

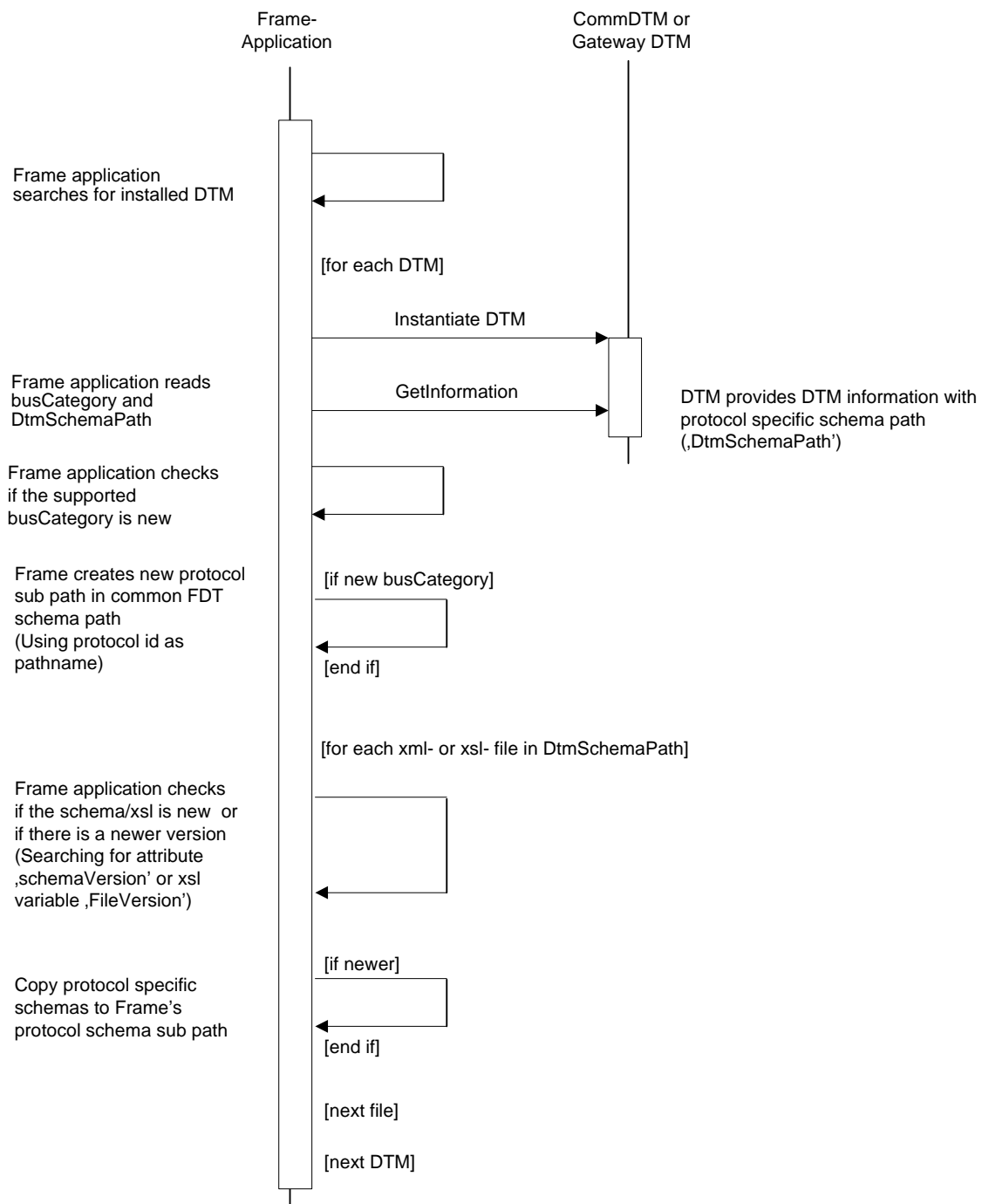
## ***6.4 Registration of Protocol Specific FDT Schemas***

In order to add new protocol support to the FDT specification and to existing Frame Application installations, a CommDTM installation must provide protocol specific schemas. There are two cases:

1. Protocol specific schemas accepted and published by FDT-JIG. These schemas may be released independent on a FDT specification release.
2. Schemas of a proprietary protocol. In this case all CommDTMs and Device-DTMs are provided in one consistent package. Consistency of all related data must be ensured internally by DTM.

The following structure explains an overall workflow:

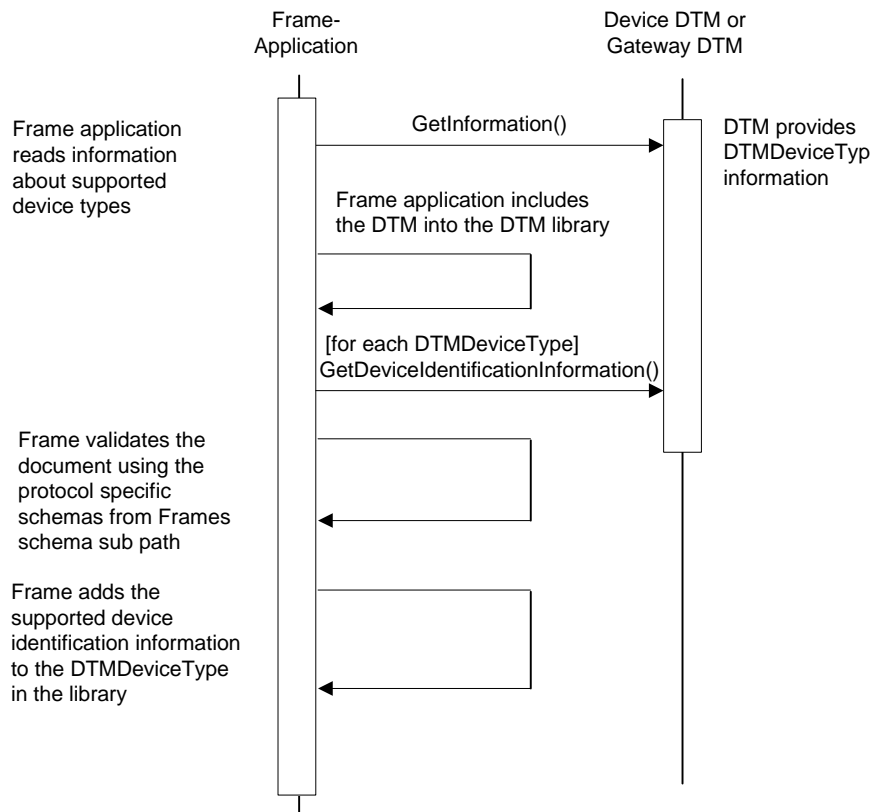
- Installation of DTMs with a channel implementing IFdtCommunication.
  - CommDTM is installed and contains in its setup the merge module of the protocol specific schemas.
  - CommDTM copies the schemas to a CommDTM specific certain path.
- DTM library update  
DTMDeviceType information is used for DTMCatalog information.
  - [IDtmInformation::GetInformation\(\)](#), returns an XML document containing schema version and path of protocol specific schemas.
  - Frame Application compares this information with the already known schemas.
  - Frame Application checks, if schemas are already available in Frame Applications sub schema path and copies the schemas if not, or if higher version is offered.
  - Frame Application is responsible to synchronize the schema files over all subdirectories.
  - Frame Application gets additional protocol specific device identification information from a DTM by calling [IDtmInformation2::GetDeviceIdentificationInformation](#).



#### Used methods:

[IDtmInformation::GetInformation\(\)](#)

Sequence diagram: Add new protocol specific schemas to Frame Applications schema sub path



Sequence diagram: Frame Application reads protocol specific device identification information of DTMDDeviceTypes

## 6.5 Configuration of a Fieldbus Master

Device-specific bus parameters are needed to configure the Frame Application's fieldbus master or communication scheduler. To retrieve these parameters an interaction between DTMs and a master configuration tool is required. To provide a standard access to this bus-specific data, it is stored as a public data accessible by the predefined XML tag

`<busMasterConfigurationPart>`

`<busMasterConfigurationPart>` is a binary stream which contains the device specific bus information according to the Fieldbus-Protocol-Specification.

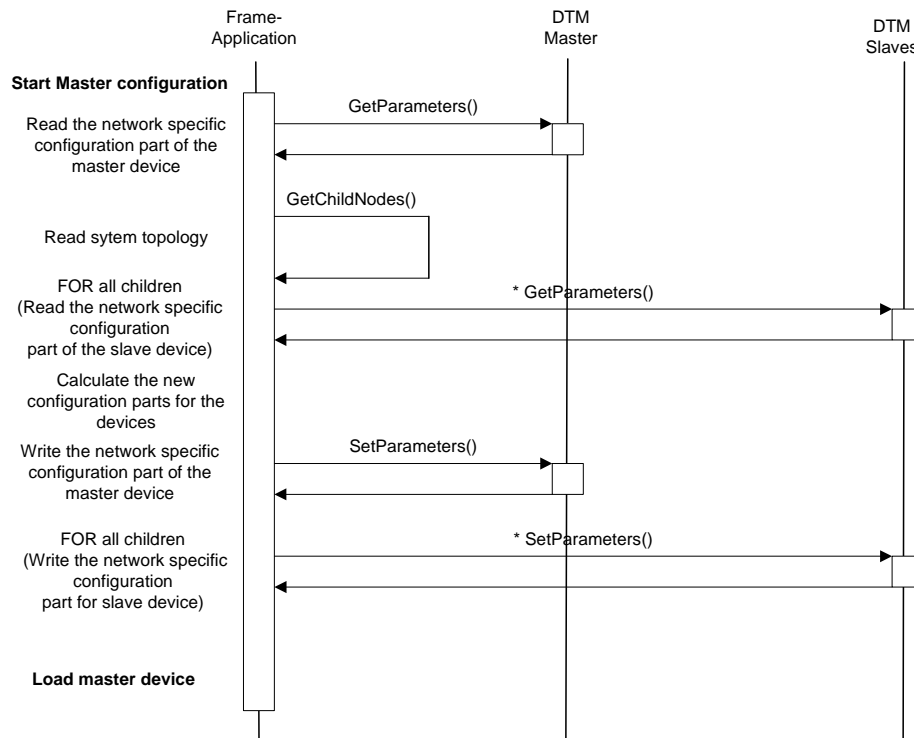
Each DTM must at least fill in the device specific parameters and all parameters which can be changed by its application.

All other entries may be filled up with substitute values like zeros. The substituted values of the `<busMasterConfigurationPart>` structure will be set by the environment's master configuration tool according to the requirements of the complete bus.

Independent of the values filled in, it is very important that the structure generated by the DTM adheres to the definitions of the fieldbus-specification.

After all network participants have written their instance data, the master configuration tool can commission the fieldbus. For that purpose it collects the <busMasterConfigurationPart> of each network participant and calculates the bus parameters of the corresponding master device.

The master configuration tool can be part of the DTM of the master device or like in the following example part of the Frame Application. If a DTM for a master device exists, the master configuration will be downloaded by [IDtmOnlineParameter::DownloadRequest\(\)](#).



### Used methods:

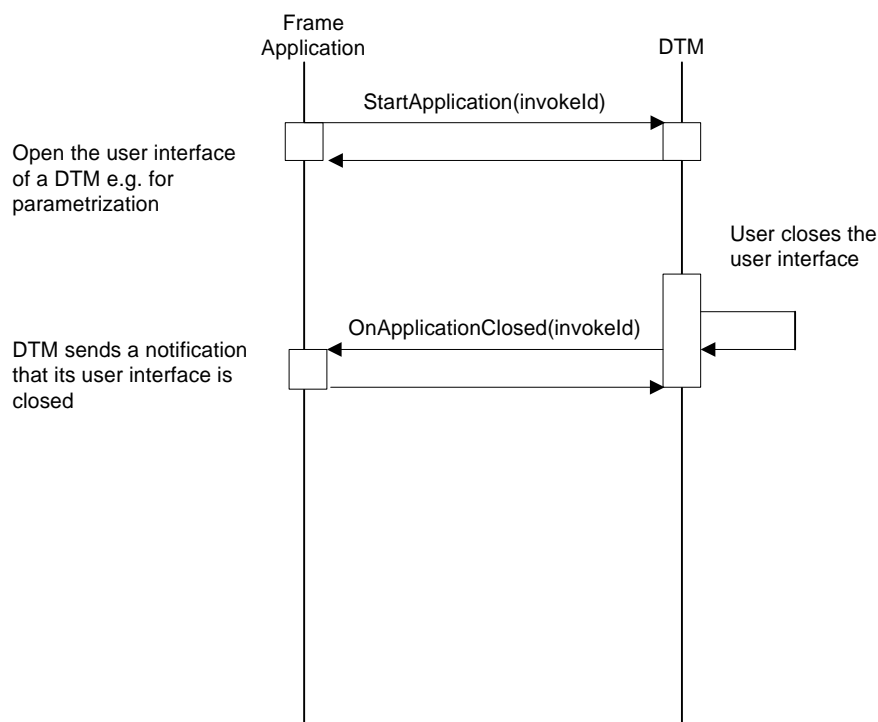
[IDtmParameter::GetParameters\(\)](#)

[IDtmParameter::SetParameters\(\)](#)

[IFdtTopology::GetChildNodes\(\)](#)

## 6.6 Starting and Releasing Applications

In general the Frame Application uses the interface [IDtmApplication](#) to start an application or uses [IDtmActiveXInformation](#) to get the information about an ActiveX control for the required application. The following sequence chart shows how the Frame Application starts an application and how it handles the asynchronous behavior of the user interface via the invoke id. The same mechanism is used for ActiveX controls. The association between user interface and invoke id can always be used to synchronize DTM and Frame Application, independent whether the user closes the user-interface or it is closed by the Frame Application via [ExitApplication\(\)](#) or [PrepareToRelease\(\)](#)



### Used methods:

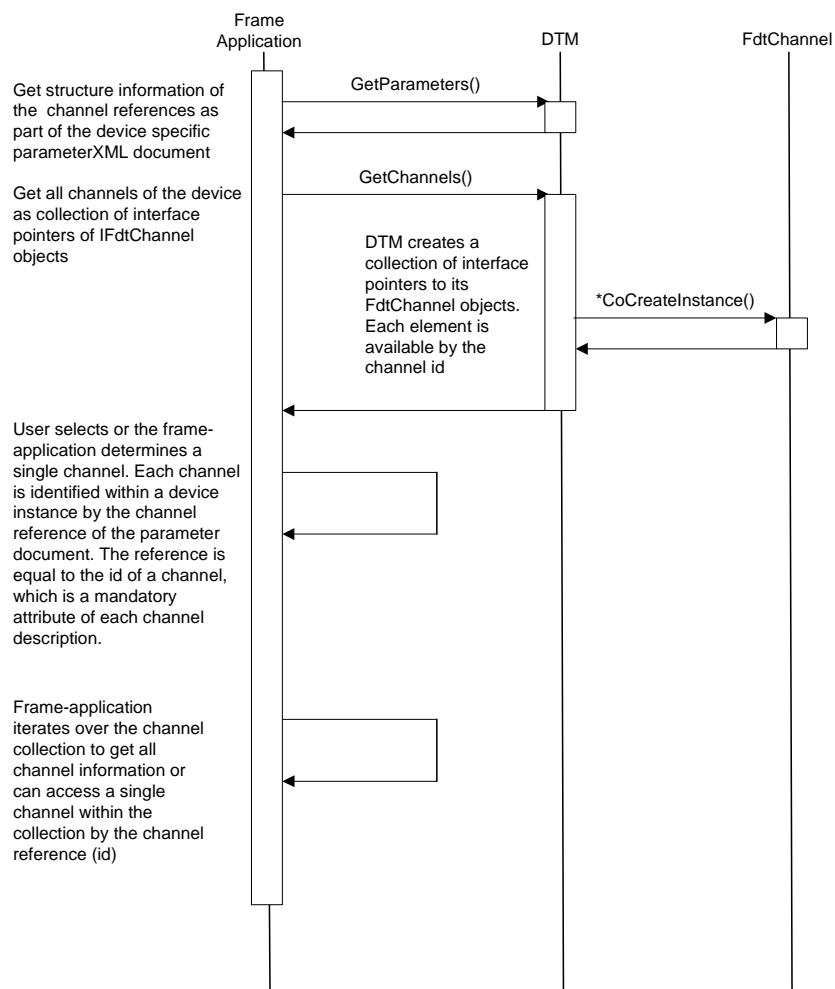
[IDtmApplication::StartApplication\(\)](#)

[IDtmEvents::OnApplicationClosed\(\)](#)



## 6.7 Channel Access

The information for the access of I/O data of a device within a Frame Application and the communication interfaces for nested communication are available via FdtChannel objects. These objects carry all address information for the configuration of a fieldbus master or a connected fieldbus controller. How to access such a channel object is fieldbus independent. But the information which is accessible as an XML document depends on the specific fieldbus protocol.



### Used methods:

Standard Microsoft

[IDtmParameter::GetParameters\(\)](#)

[IDtmChannel::GetChannels\(\)](#)

## 6.8 DCS Channel Assignment

During the channel assignment the relationship between a channel provided by the DTM and a channel within the Frame Application (DCS channel) is established by the Frame Application. To do this, the Frame Application has to get the channel properties from the DTM. To get this information it asks the DTM for the channels of the current instance and iterates over the received channels.

For the assigned channels the Frame Application sets the read-only-flag within the channel parameters. This flag causes that the channel is not deleted until the channel assignment is released.

If a DTM instance represents a complete device, all information for channel assignment is available at this DTM.

In case of a modular device like remote I/Os which is represented by one DTM, the internal structure is also available via the parameter interface. From the channel assignment point of view this information allows the Frame Application to generate a structured presentation, so that there is no difference for a user whether he works with a single DTM or with several DTMs which represent the parts of the modular device.

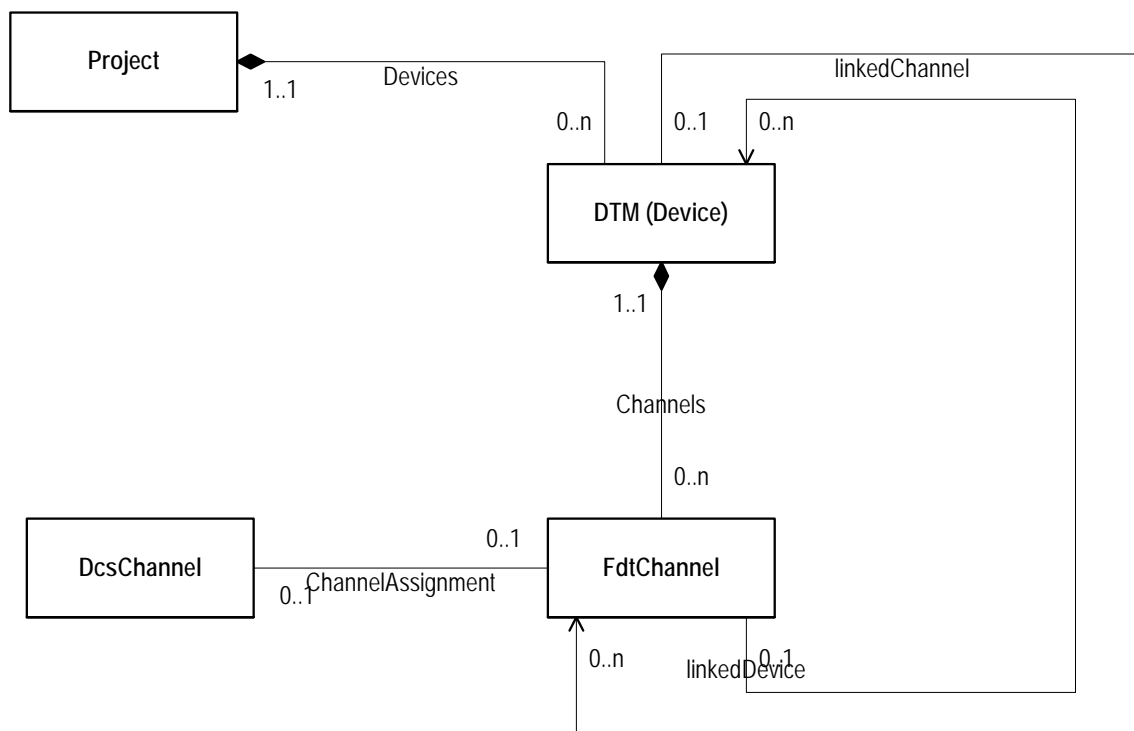
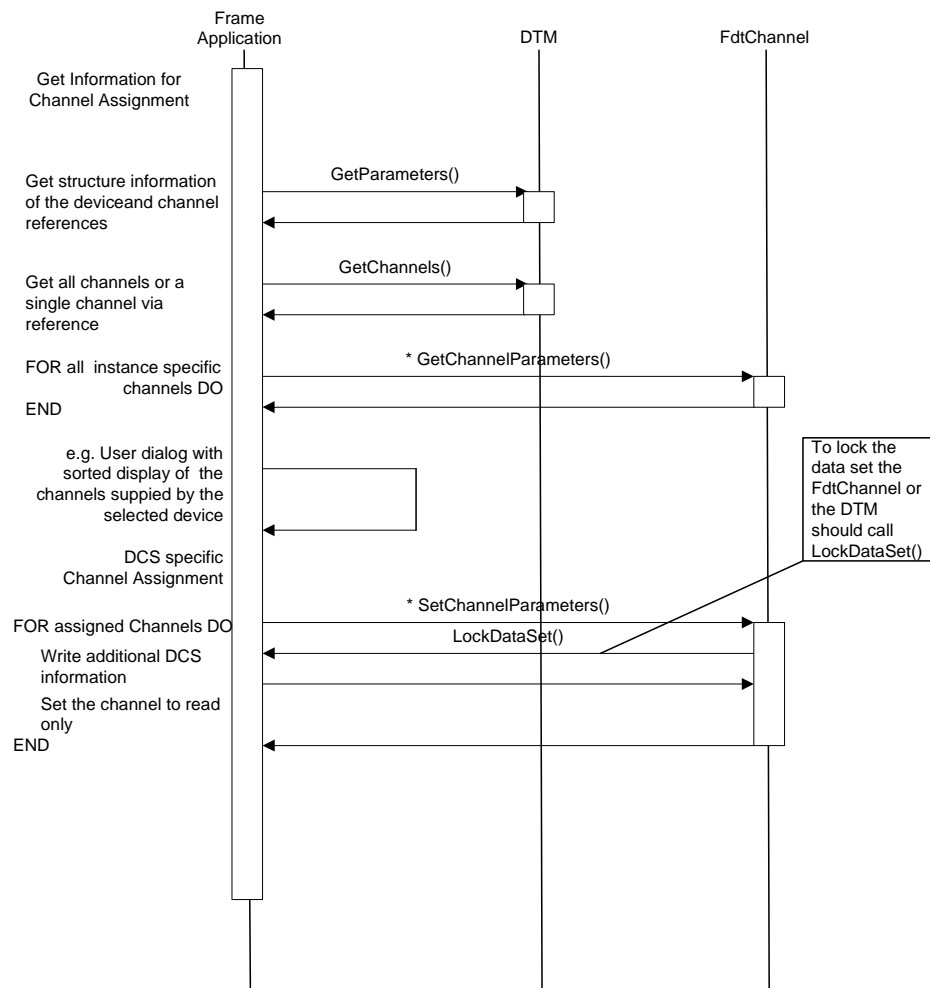


Figure 4-6 – Single DTM



### Used methods:

[IFdtContainer::LockDataSet\(\)](#)  
[IFdtContainer::UnlockDataSet\(\)](#)  
[IDtmParameter::GetParameters\(\)](#)  
[IDtmChannel::GetChannels\(\)](#)  
[IFdtChannel::GetChannelParameters\(\)](#)  
[IFdtChannel::SetChannelParameters\(\)](#)

The following figure shows a sub structure with DTMs for modular devices especially for remote I/Os with modules of different vendor. The connection of the Device-DTM and its module DTMs is established via the standard topology methods. At this sub structure the Device-DTM is the gateway between the fieldbus and the proprietary backplane bus. So the communication can be realized by the mechanisms of nested communication.

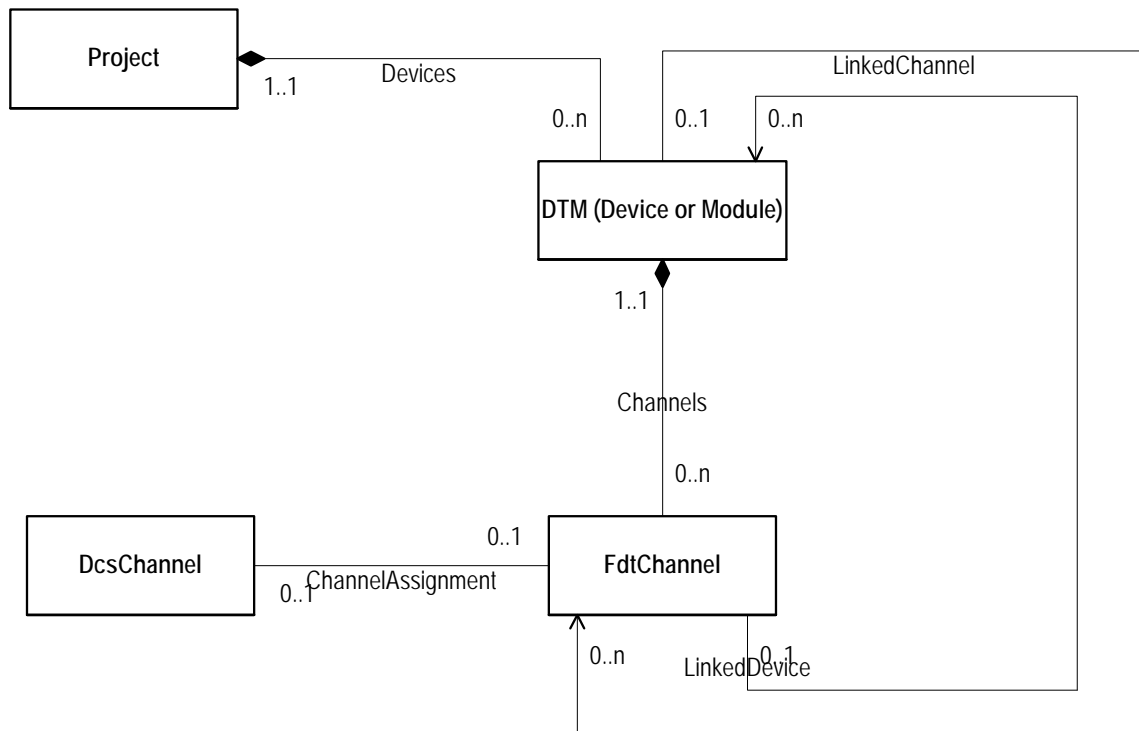
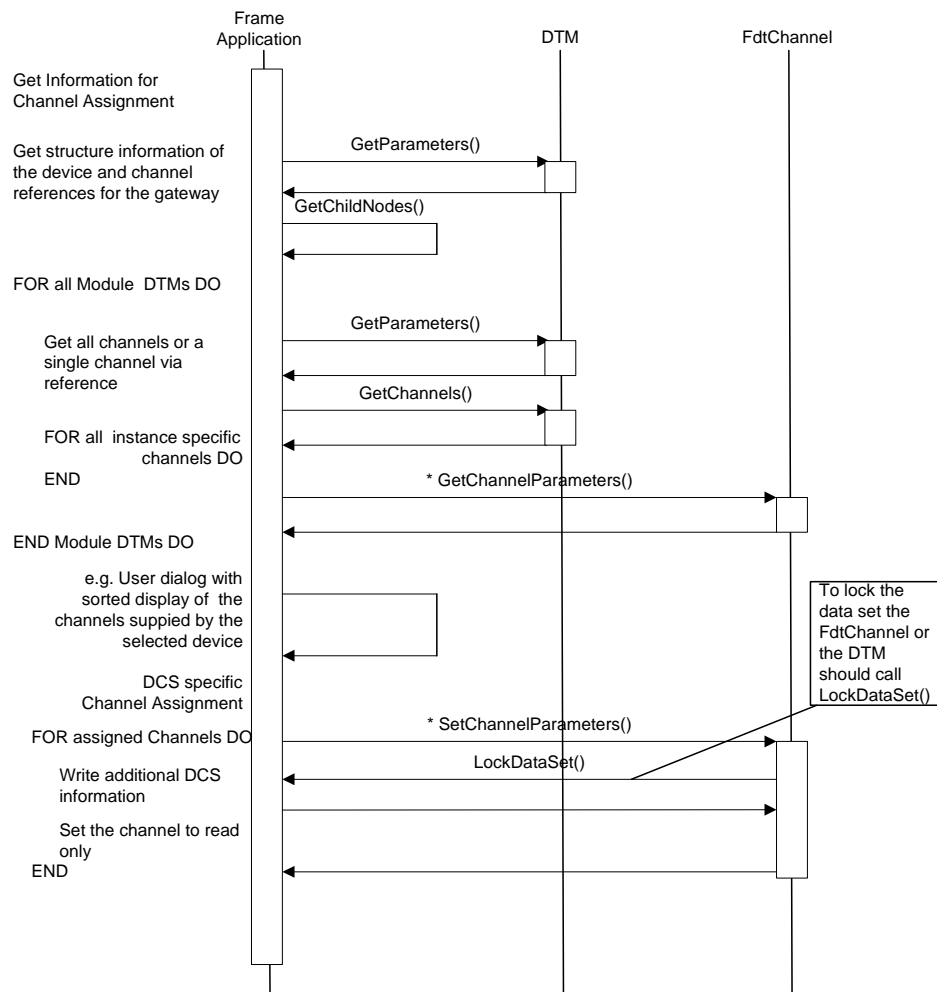


Figure 4-7 – Modular DTM Structure

In this case 'LinkedDevice' specifies the connection between a communication channel of the bus coupler (DTM Device) and the modules (DTM Module) as well as the connection between a communication channel of a module and a connected device. Especially for a remote I/O the FdtChannels are similar to the slots at the backplane.

So if a DTM instance represents only a part of a device, the information for channel assignment has to be collected by the Frame Application

In case of a modular device like remote I/Os, the gateway is signed as main DTM and is the start point to collect the information of the sub DTMs which at least belong to the same device. So the internal structure is represented by the topology. From the channel assignment point of view the channel information together with the topology allows the Frame Application to generate a structured presentation, so that there is no difference for a user whether he works with a single DTM or with several DTMs which represent the parts of the modular device.

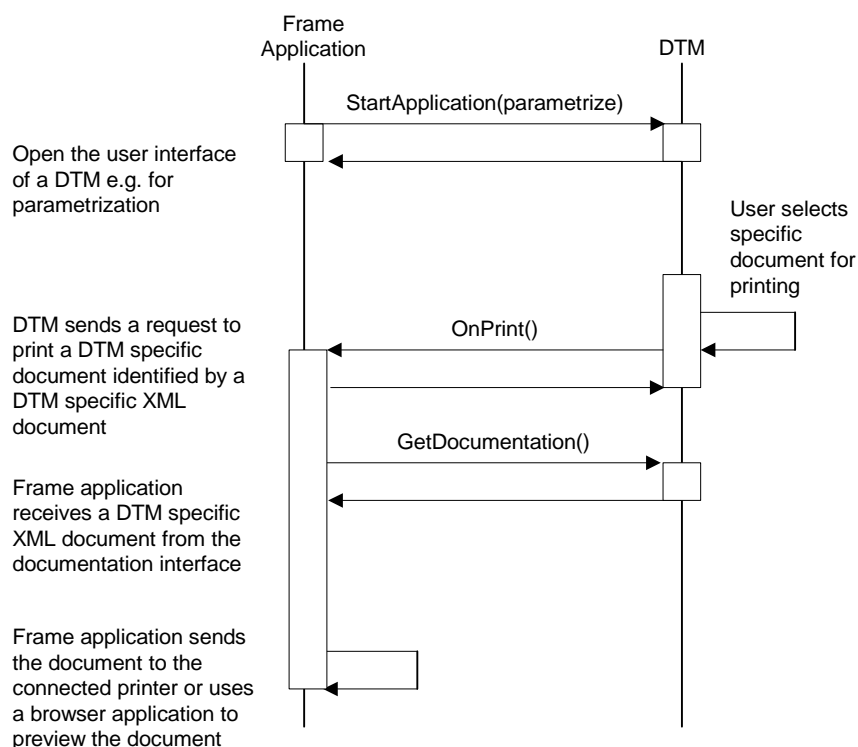


### Used methods:

[IFdtContainer::LockDataSet\(\)](#)  
[IFdtContainer::UnlockDataSet\(\)](#)  
[IDtmParameter::GetParameters\(\)](#)  
[IDtmChannel::GetChannels\(\)](#)  
[IFdtChannel::GetChannelParameters\(\)](#)  
[IFdtChannel::SetChannelParameters\(\)](#)  
[IFdtTopology::GetChildNodes\(\)](#)

## 6.9 Printing of DTM Specific Documents

In general the Frame Application uses [IDtmDocumentation](#) for its documentation. This interface allows the Frame Application to ask a DTM for context specific documents identified by the information passed via an XML document. Beneath the documents defined by application id of a Frame Application a DTM can have device- or task-specific documents. These documents are not necessary for the integration itself but are mandatory for special environments like failsafe.



### Used methods:

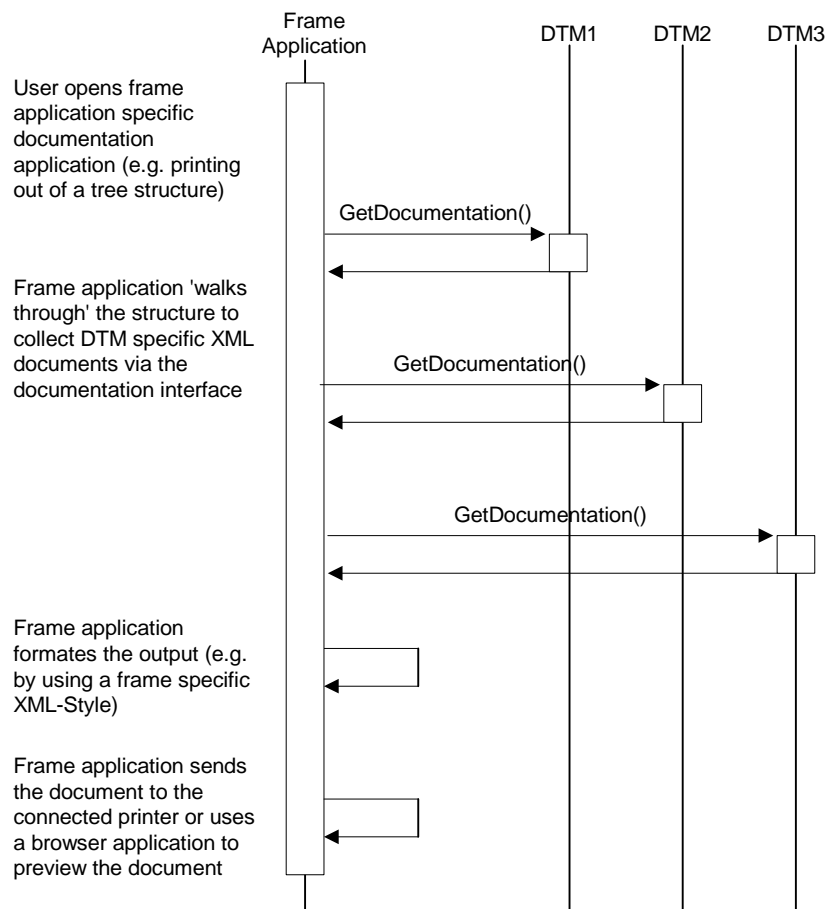
[IDtmApplication::StartApplication\(\)](#)

[IDtmEvents::OnPrint\(\)](#)

[IDtmDocumentation::GetDocumentation\(\)](#)

## 6.10 Printing of Frame Application Specific Documents

[IDtmDocumentation](#) allows the Frame Application to ask a DTM for context specific documents identified by the information passed via an XML document.

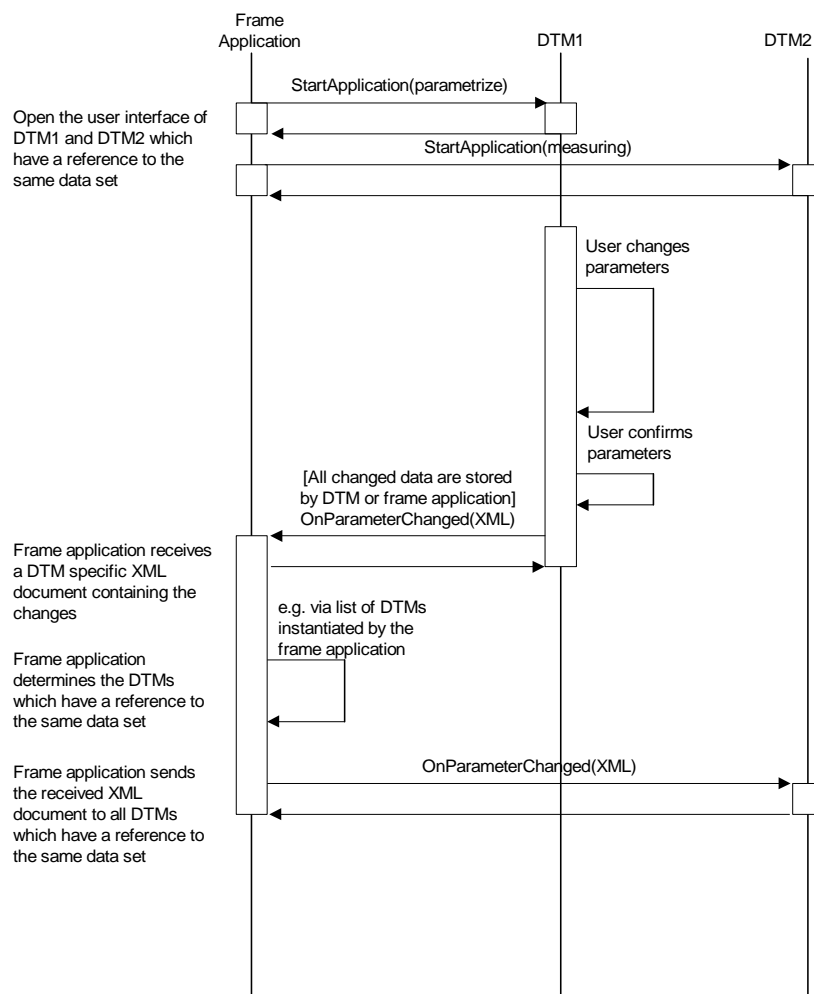


### Used methods:

[IDtmDocumentation::GetDocumentation\(\)](#)

## 6.11 Propagation of Changes

Within a multi user environment it is common, that more than one DTM have access to the same data set. To synchronize DTMs which are started by several users on different workplaces FDT provides a notification mechanism via [OnParameterChanged\(\)](#). Precondition for this event mechanism is that all changed data are stored by the DTM or by the Frame Application. All other DTMs have read access only. Furthermore all DTMs which are responsible for a device instance must have a common agreement about the contents and schema of the XML document send for propagation of changes. The Frame Application only passes the document to all DTMs which have a reference to the same data set.



### Used methods:

[IDtmApplication::StartApplication\(\)](#)

[IDtmEvents::OnParameterChanged\(\)](#)

[IFdtEvents::OnParameterChanged\(\)](#)



## 6.12 Locking

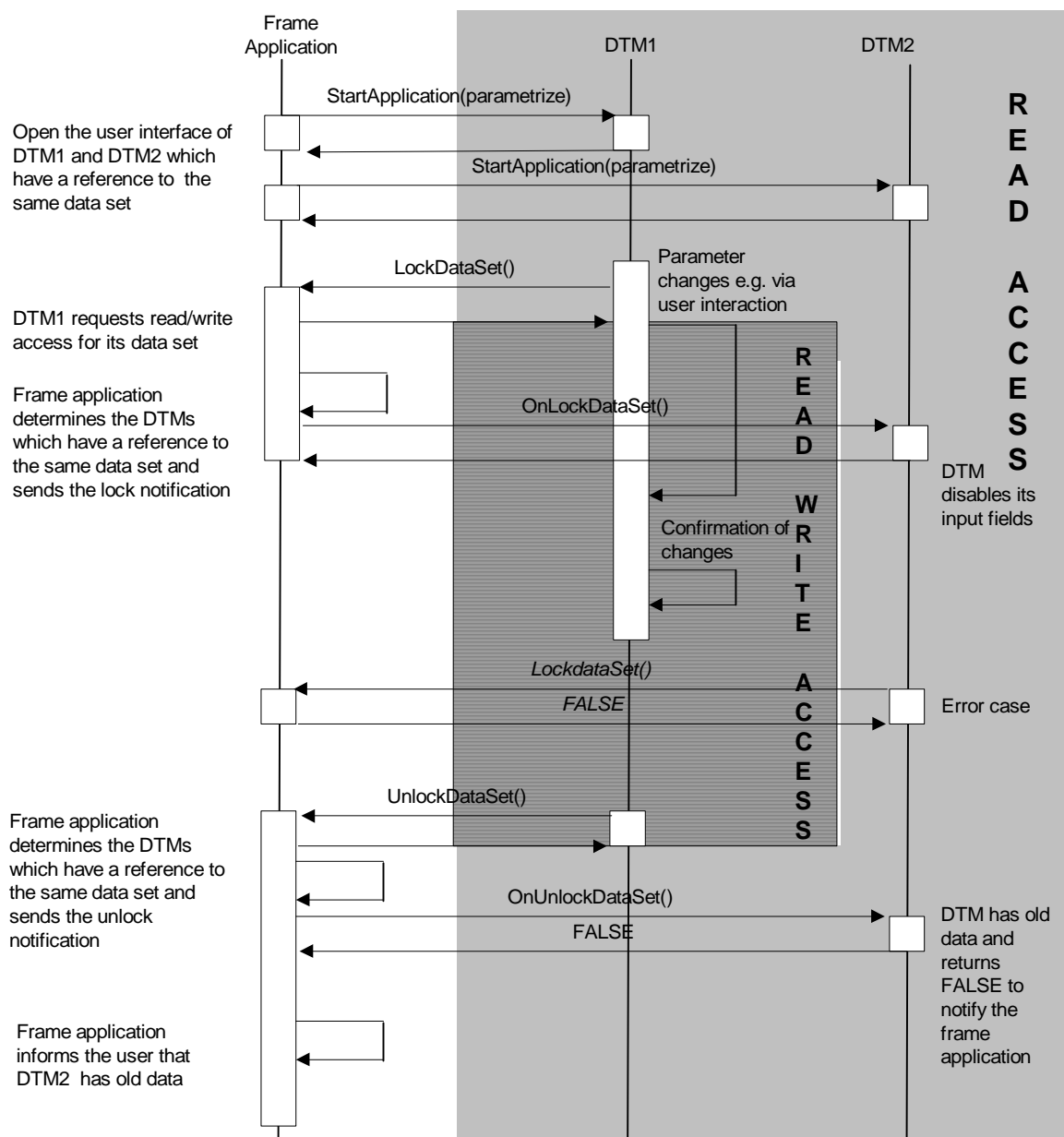
Within a multi user environment it is common, that more than one DTM have access to the same data set. To synchronize DTMs which are started by several users on different workplaces FDT provides a locking mechanism. Target for this event mechanism is that only one DTM has read/write access to the data set. All other DTMs have read access only.

For this reason a DTM must lock it's data set only if required and only during modification of the data. After the instance data is saved by the Frame Application and is not further under modification the DTM must unlock it's data set immediately to enable concurrent access to the device data by other DTM instances within a multiuser environment.

- Before opening an ActiveX the DTM must try to lock the dataset. If successful all user input fields can be enabled, in the other case user input fields must be disabled. After closing all ActiveX controls in case of a locked dataset the DTM should request to save the dataset and unlock the data set after Frame Application has saved the dataset.
- Before an upload the DTM must also try to lock the dataset, after upload DTM should call [IFdtContainer::SaveRequest\(\)](#) and unlock dataset after Frame Application has saved dataset.
- A DTM can only unlock the dataset if no user interface with write-access is opened.
- If a DTM locks it's dataset after instantiation and unlocks during e.g. a [IDtm::PrepareToRelease\(\)](#) this DTM is not suitable for use within a multiuser environment.

### 6.12.1 Locking for Non Synchronized DTMs

This locking mechanism is suitable for all DTMs which do not implement [IFdtEvents::OnParameterChanged\(\)](#) ([IFdtEvents::OnParameterChanged\(\)](#) returns E\_NOTIMPL). The mechanism must be implemented to support multi-user environments.



### Used methods:

`IDtmApplication::StartApplication()`

`IFdtContainer::LockDataSet()`

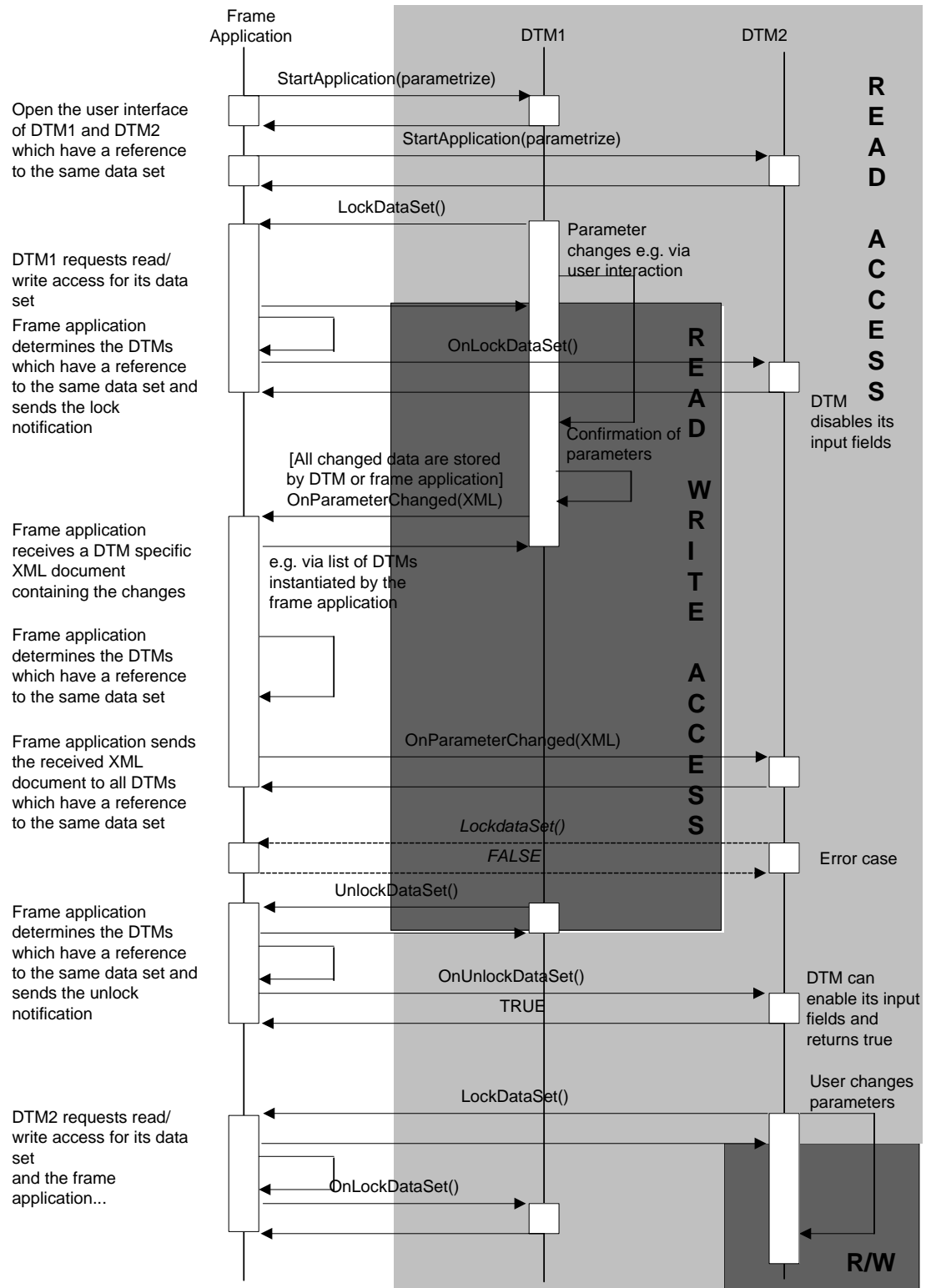
`IFdtContainer::UnlockDataSet()`

`IFdtEvents::OnLockDataSet()`

`IFdtEvents::OnUnlockDataSet()`

## 6.12.2 Locking for Synchronized DTMs

The synchronization of DTMs is an optional feature to provide a better handling for the user within a multi-user environment.



**Used methods:**

IDtmApplication::StartApplication()

IFdtContainer::LockDataSet()

IFdtContainer::UnlockDataSet()

IDtmEvents::OnParameterChanged()

IFdtEvents::OnParameterChanged()

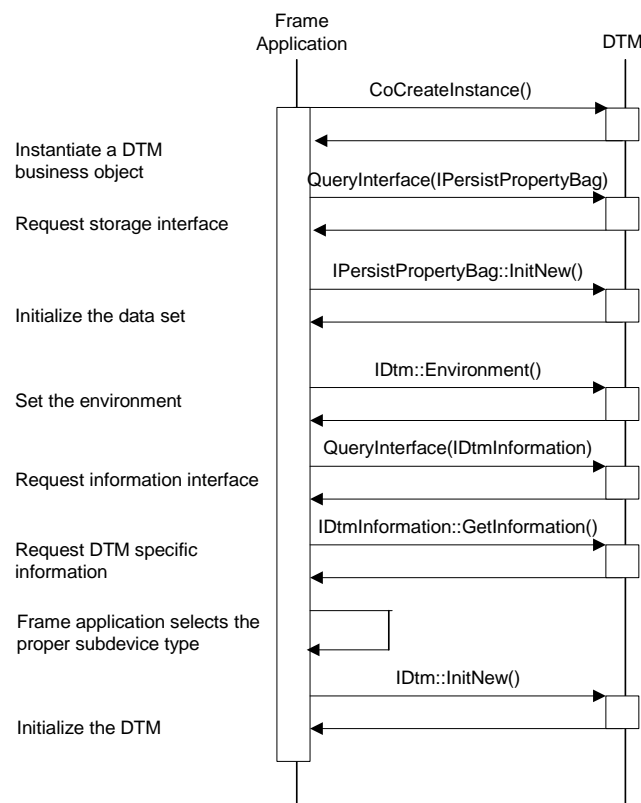
IFdtEvents::OnLockDataSet()

IFdtEvents::OnUnlockDataSet()

## 6.13 Instantiation and Release

### 6.13.1 Instantiation of a New DTM

After creation of a DTM business object the Frame Application must request the IPersistXXX interface pointer and call IPersistXXX::InitNew(). Within this method the DTM business object must initialize it's default device parameters.



#### Used methods:

Standard Microsoft

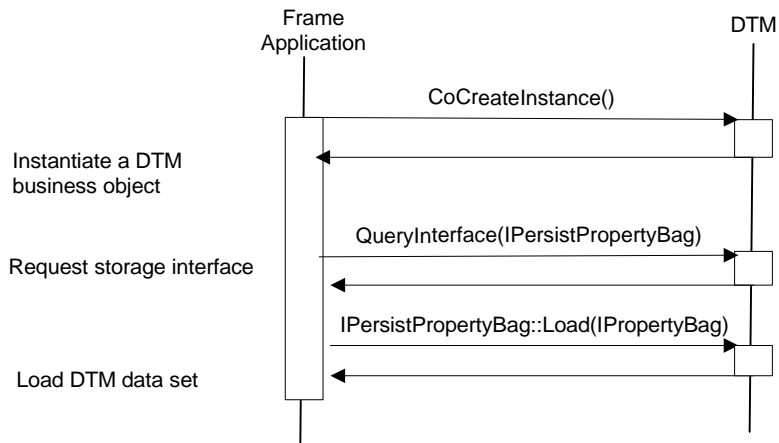
[IDtm::Environment\(\)](#)

[IDtm::InitNew\(\)](#)

[IDtmInformation::GetInformation\(\)](#)

### 6.13.2 Instantiation of an Existing DTM

After creation of a DTM business object for an existing instance the Frame Application must request the IPersistXXX interface pointer and call IPersistXXX::Load() with a reference to the stream object of the appropriate field device instance. Within this method the DTM business object must initialize it's device parameters based on the information of the stream object.

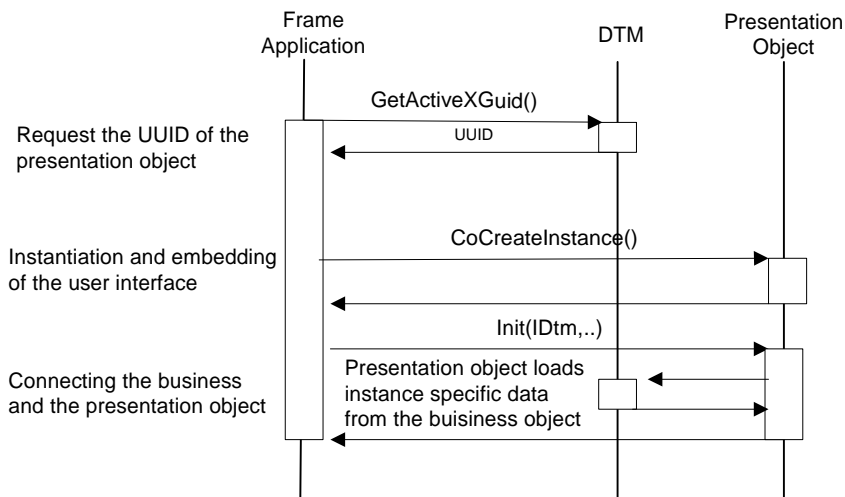


### Used methods:

Standard Microsoft

## 6.13.3 Instantiation of a DTM User Interface

After creation of a DTM business object the Frame Application can instantiate a presentation object as user interface for a special task related to the application context . Within the Init() method the presentation object must initialize it's device parameters based on the information of the business object.



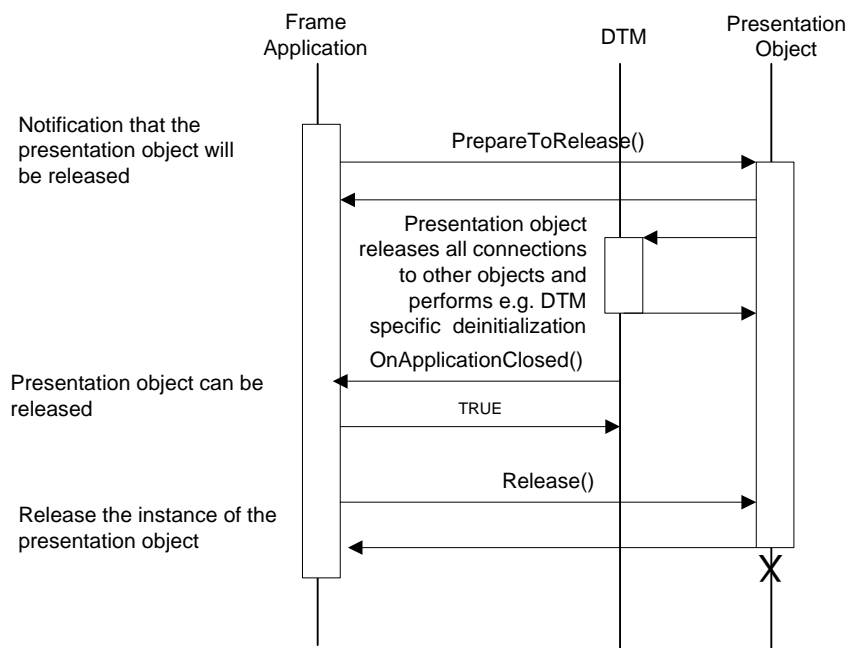
### Used methods:

Standard Microsoft

[IDtmActiveXControl::Init\(\)](#)  
[IDtmActiveXInformation::GetActiveXGuid\(\)](#)

### 6.13.4 Release of a DTM User Interface

If the Frame Application wants to release a presentation object of a DTM it first has to prepare the release by sending a notification to the presentation object. After the [IDtmActiveXControl::PrepareToRelease\(\)](#) method the presentation object must release all its connections to other components and can call DTM specific de-initialization methods.



#### Used methods:

Standard Microsoft

[IDtmActiveXControl::PrepareToRelease\(\)](#)

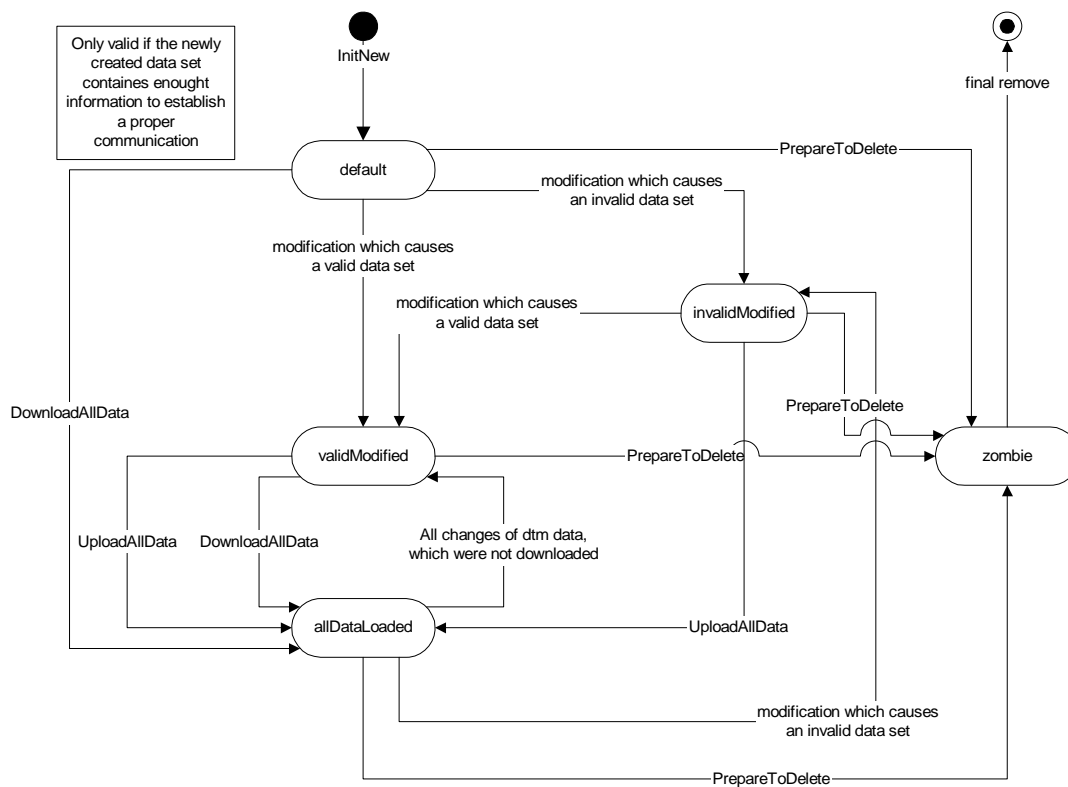
[IDtmEvents::OnApplicationClosed\(\)](#)

## 6.14 Persistent Storage of a DTM

### 6.14.1 State machine of instance data

#### 6.14.1.1 Modifications

This state machine reflects the possible states of an instance data set concerning modifications.

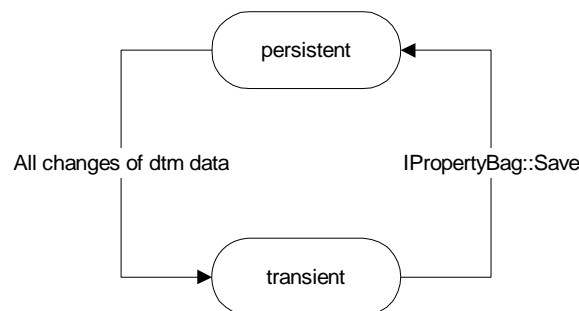


State	Meaning
default	The contents of the instance data set are the default data. This state will typically appear after creation of a new data set
validModified	The data set was modified in a consistent manner
invalidModified	The data set was modified. The data are not in a consistent state.
AllDataLoaded	The instance data set was loaded from or into the related device. NOTE: This state means not, that the data within the device are equal to the data found within the related instance data set. Due to the fact that a user can use tools out of the scope of FDT, the FDT-Specification can not guarantee such a state
zombie	The instance data set is prepared to delete, no access to this data is allowed



### 6.14.1.2 Persistence

This state machine reflects the possible states of an instance data set concerning the persistence of data.

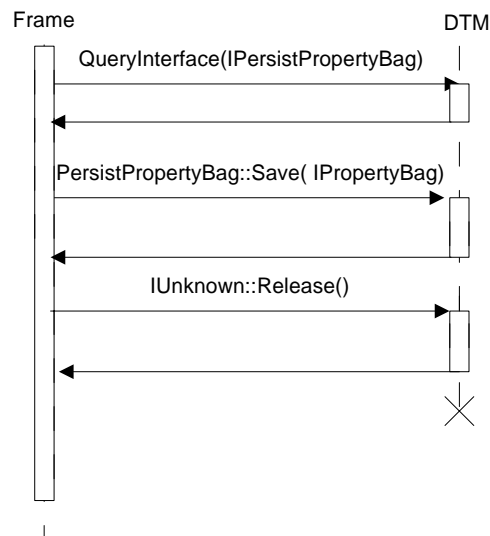


State	Meaning
persistent	The content of the instance data set is persistent. This state will typically appear after the data set was saved by the Frame Application using the persistence interfaces of the DTM.
transient	The data set was modified. These changes are not saved in a persistent way. All modified instance data within the DTM is temporary and not synchronized with other DTM instances or with the Frame Application. A <a href="#">IFdtContainer::SaveRequest()</a> call just informs the Frame Application that the data set should be stored.

### 6.14.2 Saving Instance Data of a DTM

To save the private data of a DTM object the Frame Application must request the IPersistXXX interface pointer and call IPersistXXX::Save() with a reference to the stream / property bag object of the appropriate field device instance. Within this method the DTM business object must save all it's device parameters necessary to re-establish its complete state based on the information of the stream / property bag object within a IPersistXXX::Load() call.

After saving private data of a DTM the Frame Application is able to release the DTM business object



**Used methods:**

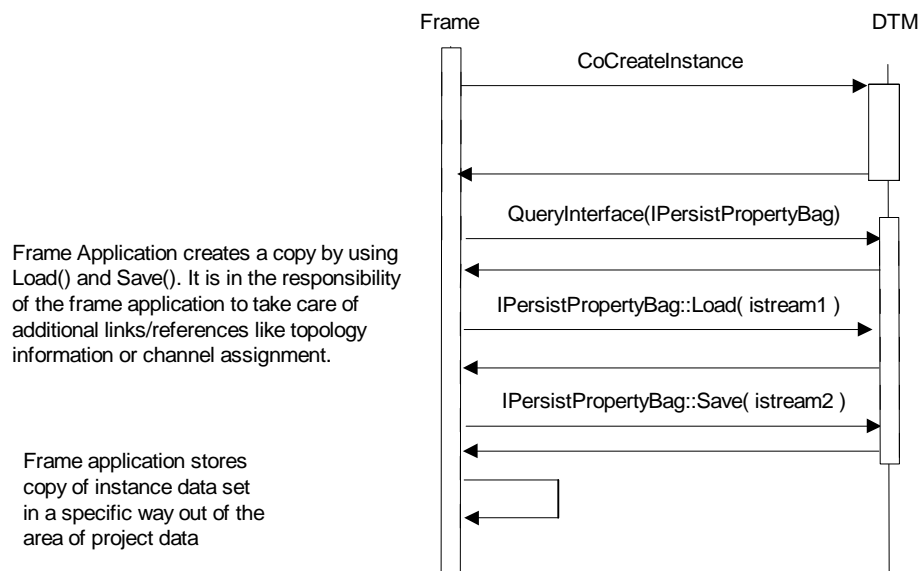
Standard Microsoft

### 6.14.3 Reload of a DTM Object for Another Instance

A Frame Application may reuse one DTM COM object for a number of different field device instances by calling IPersistXXX::Load() several times and with different Istream / IpropertyBag objects as argument. But, if an IPersistXXX::Load() call fails, the Frame Application must instantiate a new DTM COM object.

#### 6.14.4 Copy and Versioning of a DTM Instance

After creation of a DTM object the Frame Application must request the IPersistXXX interface pointer and call IPersistXXX::Load() with a reference to the stream / property bag object of the appropriate field device instance. Within this method the DTM business object must initialize it's device parameters based on the information of the stream object. To get a copy of the private data of a DTM business object the Frame Application must call IPersistXXX::Save() with a reference to a new stream / property bag object. It is up to the Frame Application to handle the Frame Application specific versioning aspects and to manage the different instance data sets for a device.



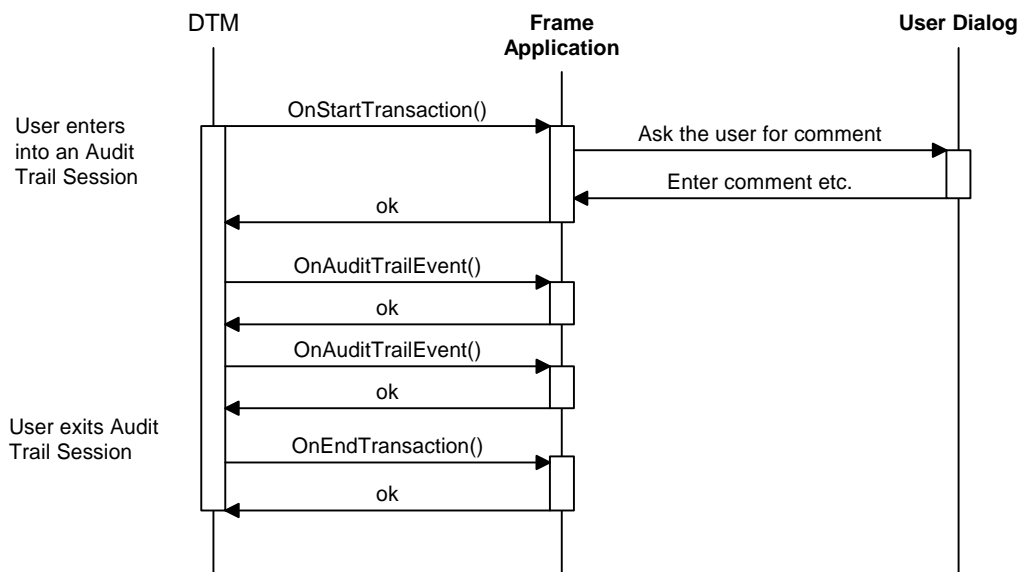
##### Used methods:

Standard Microsoft

## 6.15 Audit Trail

Audit Trail services are implemented in the Frame Application. It stores all information about audit trail session like username, date and time, description and comment of the session and description of all audit-trail events within the session. It provides saving, analyzing a documentation of the audit trail sessions. For the DTM it offers the interface IDtmAuditTrailEvents.

When Audit Trail is started the Frame Application may ask the user for additional comments.



### Used methods:

[IDtmAuditTrailEvents::OnStartTransaction\(\)](#)  
[IDtmAuditTrailEvents::OnAuditTrailEvent\(\)](#)  
[IDtmAuditTrailEvents::OnEndTransaction\(\)](#)

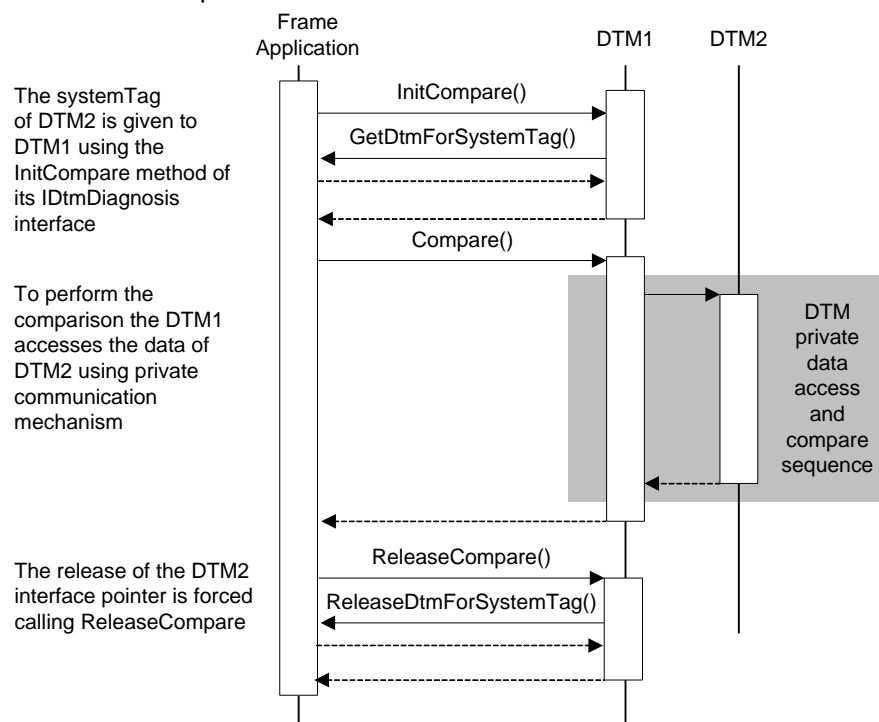
## 6.16 Comparison of Two Instance Data Sets

This section describes sequences to compare the data sets of two different instances using the IDtmDiagnosis interface.

The two DTM's invoked in the comparison have to be of the same type.

### 6.16.1 Comparison Without User Interface

The sequence chart for the comparison of two instance data sets is as follows:



#### Used methods:

IDtmDiagnosis::InitCompare()

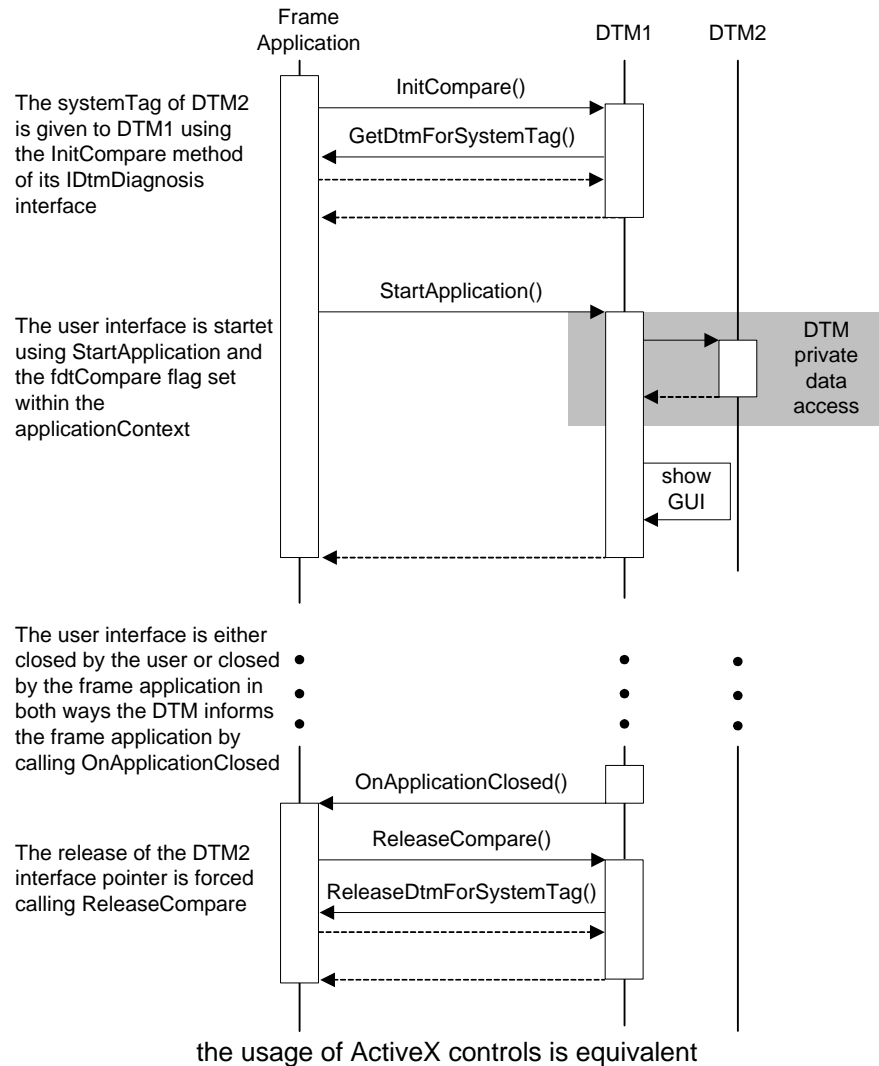
IDtmDiagnosis::Compare()

IFdtTopology::GetDtmForSystemTag()

IFdtTopology::ReleaseDtmForSystemTag()

## 6.16.2 Comparison With User Interface

To invoke a user interface for comparison, the following sequence is to be performed.



### Used methods:

IDtmDiagnosis::InitCompare()

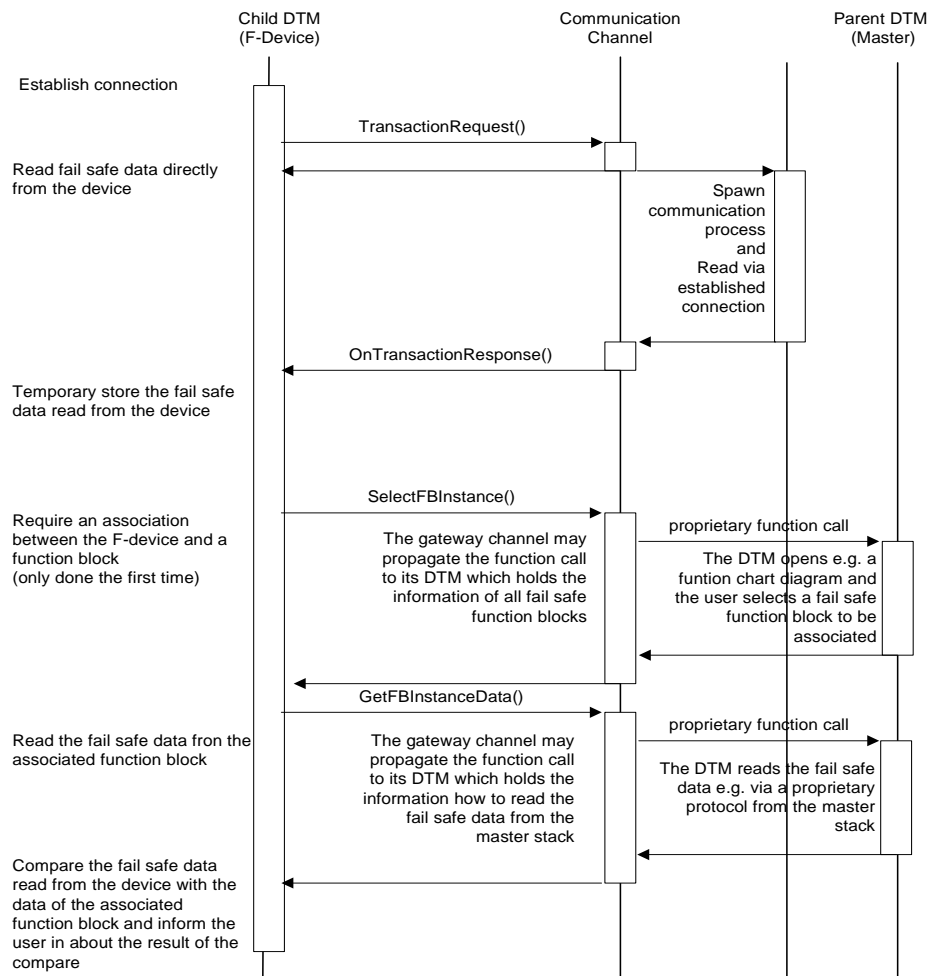
IDtmDiagnosis::ReleaseCompare()

IDtmApplication::StartApplication()

IDtmEvents::OnApplicationClosed()

## 6.17 Failsafe Data Access

The sequence chart shows how to access failsafe data from a device and its associated function block via two independent communication links.



### Used methods:

`IFdtCommunication::TransActionRequest()`

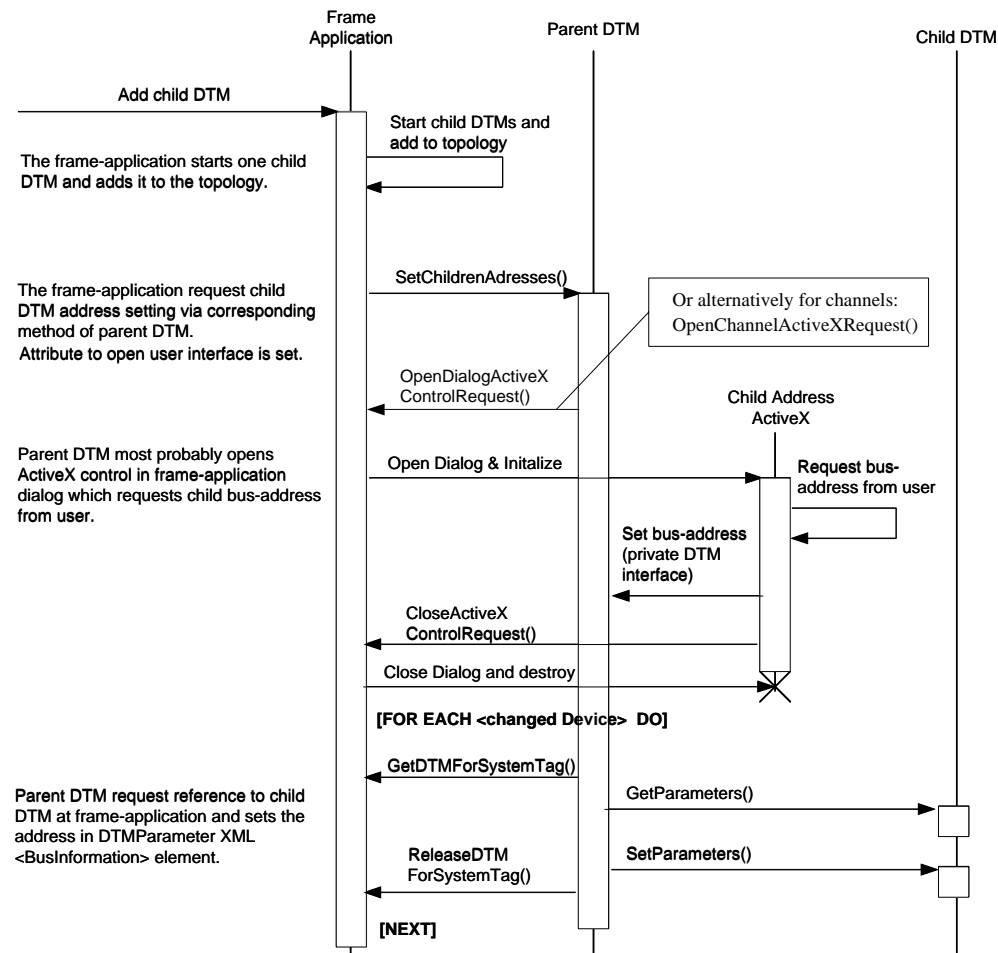
`IFdtCommunicationEvents::OnTransactionResponse()`

`IFdtFunctionBlockData::SelectFBInstance()`

`IFdtFunctionBlockData::GetFBInstanceData()`

## 6.18 Set or Modify Device Address With User Interface

In this scenario Frame Application requests a set of specific child device addresses at DTMs. This sequence is for example started if new DTM is added to the topology. Similar sequence can be used if a Frame Application offers manual change of address in context of a DTM.



### Used methods:

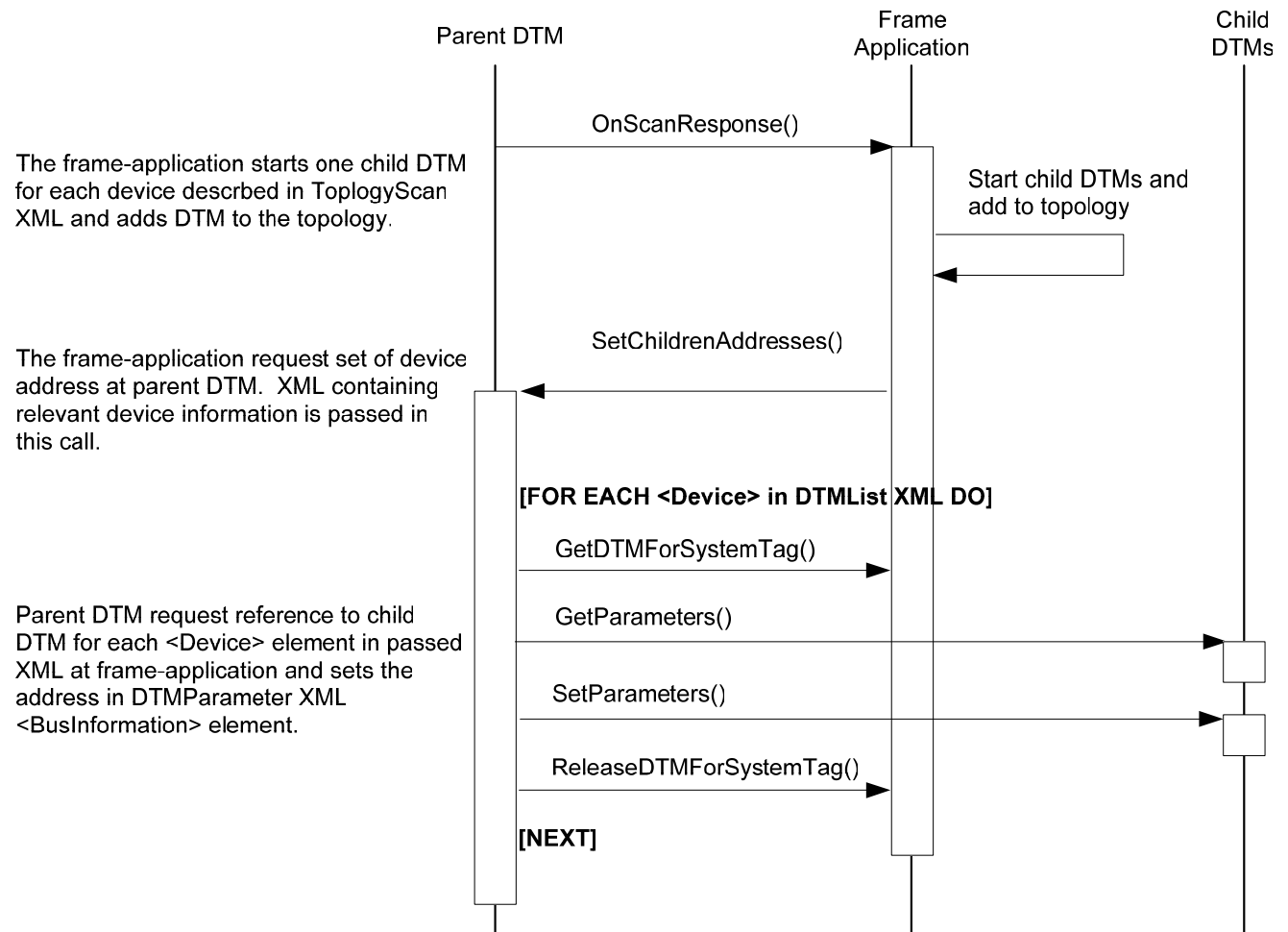
```

IFdtChannelSubTopology2::SetChildrenAddresses()
IFdtActiveX2::OpenDialogActiveXControlRequest()
IFdtActiveX::CloseActiveXControlRequest()
IFdtTopology::GetDtmForSystemTag()
IFdtTopology::ReleaseDtmForSystemTag()
IDtmParameter::SetParameters()
IDtmParameter::GetParameters()
  
```



## 6.19 Sets or Modifies Known Device Addresses Without User Interface

In this scenario, the Frame Application requests set of device addresses at DTM, for example after a scanning or import operation.

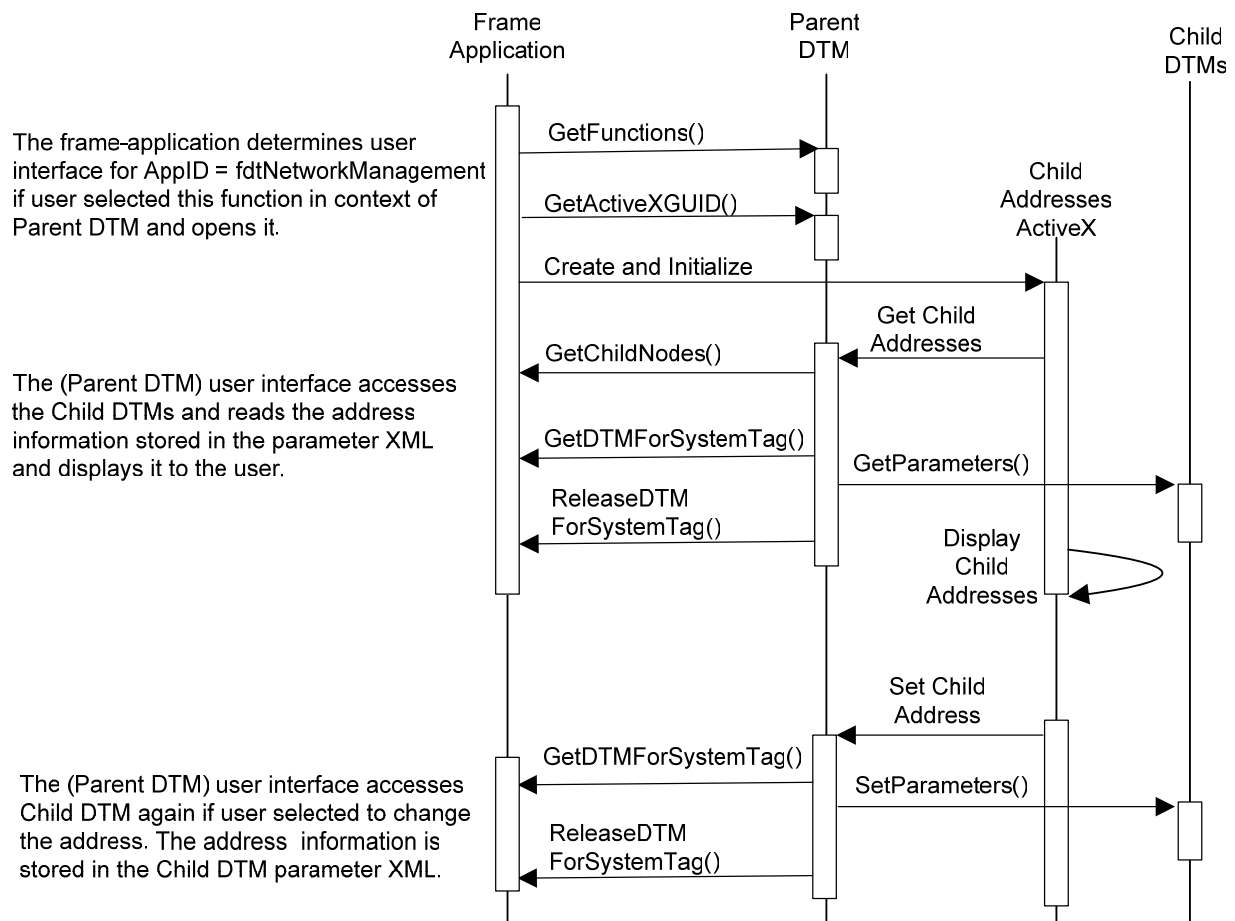


### Used methods:

[IFdtChannelSubTopology2::SetChildrenAddresses\(\)](#)  
[IDtmEvents::OnScanResponse\(\)](#)  
[IFdtTopology::GetDtmForSystemTag\(\)](#)  
[IFdtTopology::ReleaseDtmForSystemTag\(\)](#)  
[IDtmParameter::SetParameters\(\)](#)  
[IDtmParameter::GetParameters\(\)](#)

## 6.20 Display or Modify All Child Device Addresses With User Interface

In this scenario Frame Application requests display or modification of all child device addresses at a DTM. This sequence is for example started if user selects corresponding menu entry in context of a communication or Gateway-DTM..



### Used methods:

IDtm::GetFunctions()

IDtmActiveXInformation::GetActiveXGuid()

IFdtTopology::GetChildNodes()

IFdtTopology::GetDtmForSystemTag()

IFdtTopology::ReleaseDtmForSystemTag()

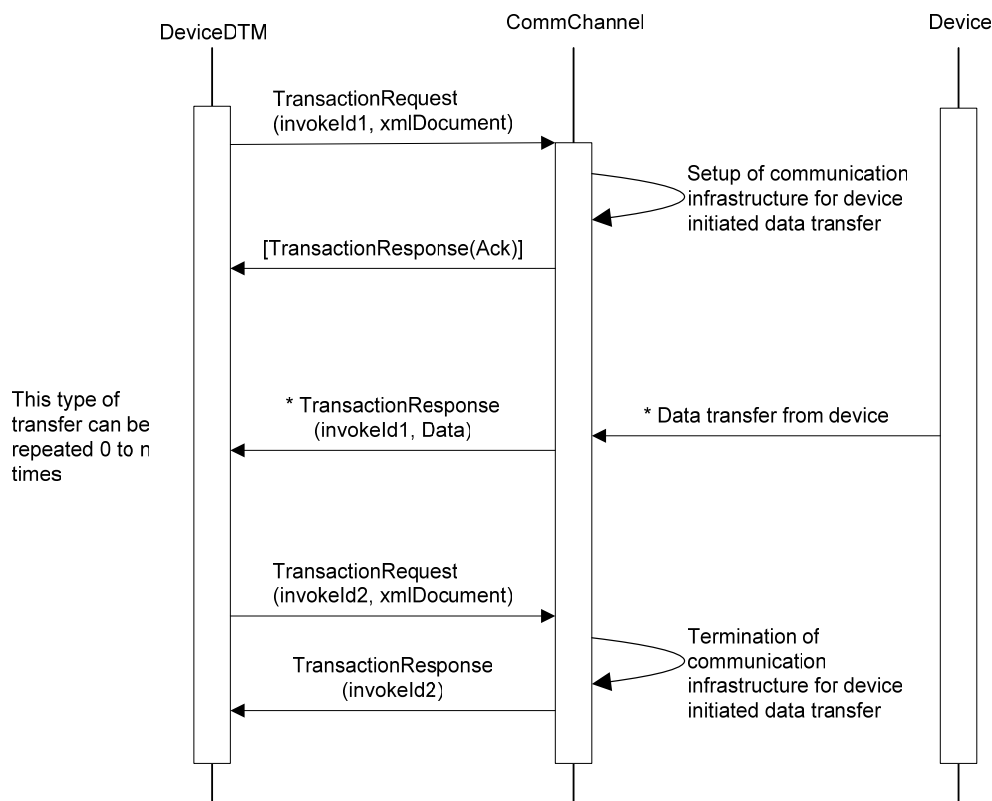
IDtmParameter::SetParameters()

IDtmParameter::GetParameters()

## 6.21 Device Initiated Data Transfer

Some protocols support data transfer services that are initiated by the device and not by the DTM. In order to support such services, following approach is recommended:

- the infrastructure (filter, service queue) for such services is initiated by a protocol-specific transaction request of the DTM, dependant on the protocol this service may be acknowledged or not
- the device initiated data transfer is transported by transaction responses to the initiating request (they carry the same invokeld)
- the infrastructure for these services is terminated by a protocol-specific transaction request of the DTM, which may carry the original invokeld as an attribute of the request xml-Document.



Device initiated data transfer needs management by the communication infrastructure. That is why it is necessary to unambiguously identify the request to initialize and terminate this type of behavior. The TransactionRequests to initiate or terminate device initiated data transfer will transport protocol specific XML-Documents.

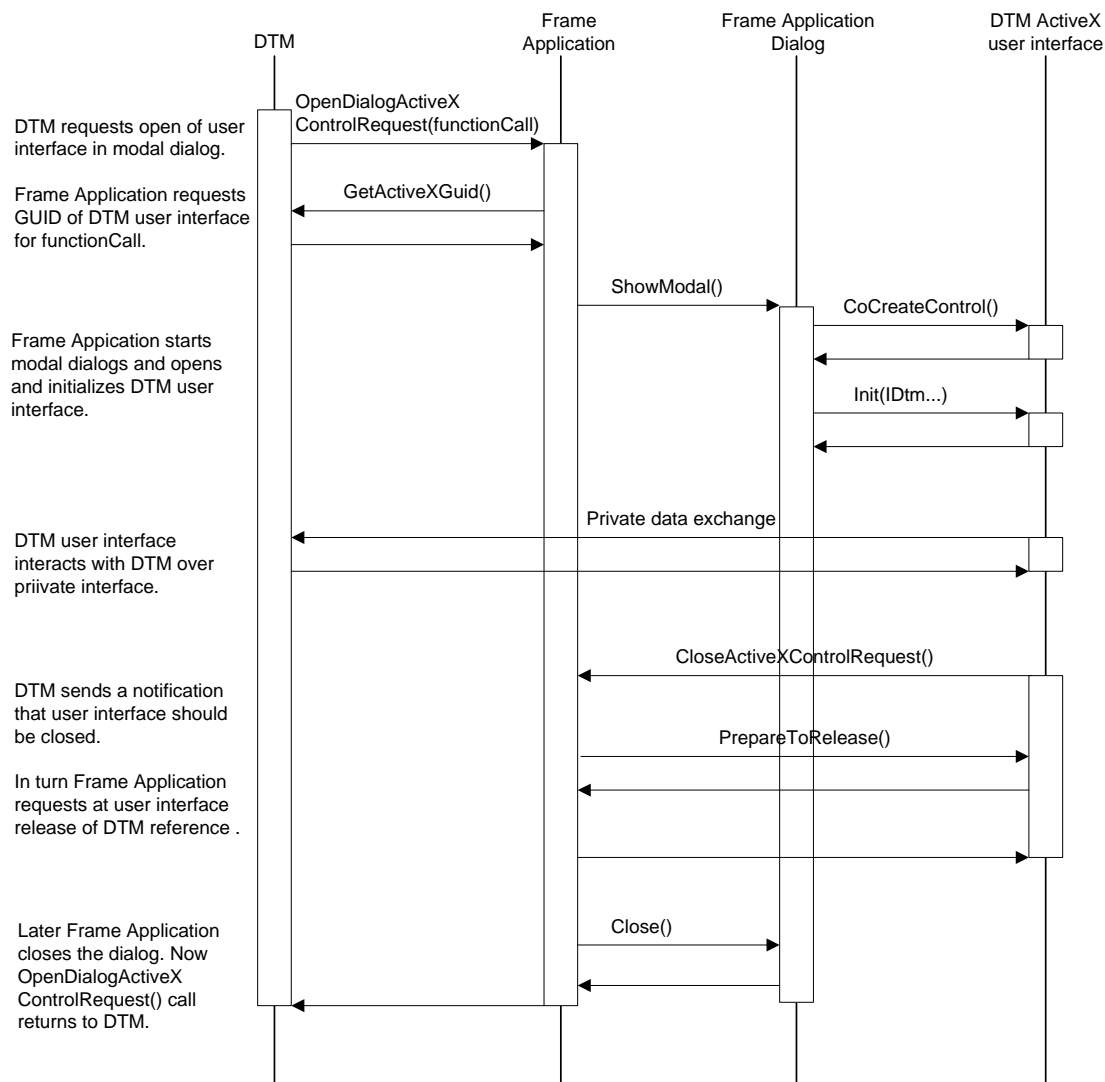
The terminating TransactionRequest will contain the information necessary to terminate the device initiated data transfer. This may include the invokeld of the initiating TransactionRequest.

If a DTM initializes such services, it is required to terminate these services. The termination has to occur, before the DTM can go offline (DisconnectRequest / Abort). If the connection is terminated by [IFdtCommunicationEvents::OnAbort\(\)](#), this service is terminated automatically.

The support of such services is protocol specific and device specific. If a communication channel is not able to support this type of service, it will return a protocol specific error document in the TransactionResponse, indicating that it is not able to support this feature.

## 6.22 Starting and Releasing DTM User Interface in Modal Dialog

DTM requests start of user interface over IFdtActiveX2 interface implemented by Frame Application. This starts modal dialog and opens DTM user interface in it. DTM user interface exchanges data with DTM over private interface and request closing of dialog after work is finished. Frame Application closes the dialog and DTM user interface. Now [IFdtActiveX2:OpenDialogActiveXControlRequest\(\)](#) call returns to the DTM.



### Used methods:

Standard Microsoft

[IDtmActiveXInformation::GetActiveXGuid\(\)](#)

[IFdtActiveX2:OpenDialogActiveXControlRequest\(\)](#)

[IFdtActiveX:CloseActiveXControlRequest\(\)](#)

[IDtmActiveXControl:Init\(\)](#)

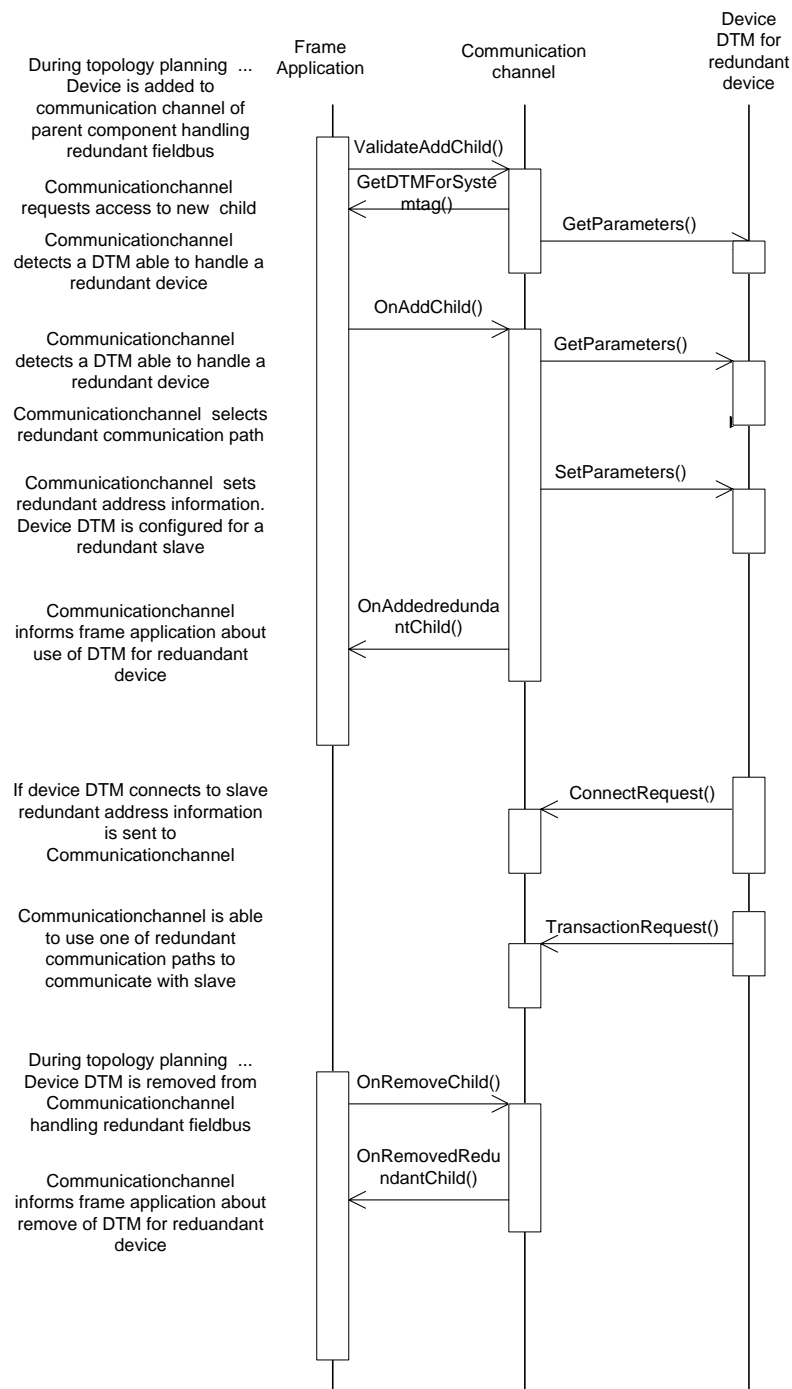
[IDtmActiveXControl:PerpareToRelease\(\)](#)

## **6.23 Parent Component Handling Redundant Slave**

A parent component, e.g. a COMM-DTM, handling a redundant fieldbus is able to detect a Device-DTM able to handle redundant slave within an [IFdtChannelSubTopology::OnAddChild\(\)](#) by examining the parameter document of this DTM. The COMM-DTM selects the redundant communication paths (either automatically or using a dialog) and sets redundancy information to the Device-DTM by calling `SetParameter()`.

Device-DTM provides this redundancy information within its connect request to the COMM-DTM. For each such opened connection the COMM-DTM is able to use one of the available communication paths without need for further interaction with the Device-DTM.

A Frame Application implementing the `IDtmRedundancyEvents` interface is able to get topology information about redundant DTMs. In such a Frame Application the topology view may show this redundancy information. On the other hand COMM-DTM and Device-DTM of a redundant slave can be used in a Frame Application without knowledge about redundancy, because all redundancy functionality is handled by the COMM-DTM and its child DTMs.



### Used methods:

[IDtmRedundancy::OnAddedRedundantChild\(\)](#)

[IDtmRedundancyEvents::OnRemovedRedundantChild\(\)](#)

## 6.24 Initialization of a Channel ActiveX Control

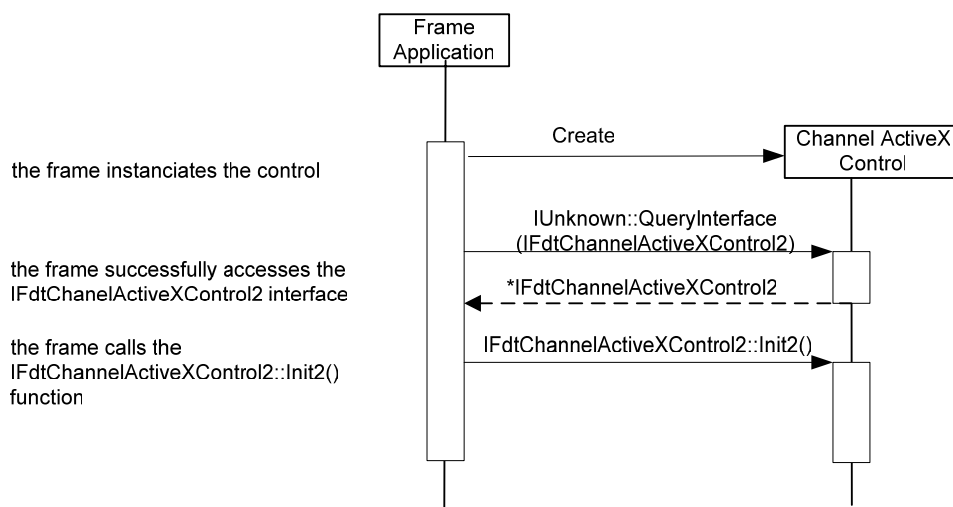
To initialize a channel ActiveX control conformant to FDT 1.2.1 specification, the Frame Application has first to check, if the interface IFdtChannelActiveXControl2 exists.

If it exists the Frame Application has to call the [IFdtChannelActiveXControl2::Init2\(\)](#) function at this interface.

If it does not exist, the Frame Application calls the [IFdtChannelActiveXControl::Init\(\)](#) function.

### 6.24.1 Supports IFdtChannelActiveXControl2

This diagram describes the sequence of the channel ActiveX control initialization for the case that the channel ActiveX control provides the IFdtChannelActiveXControl2 interface.



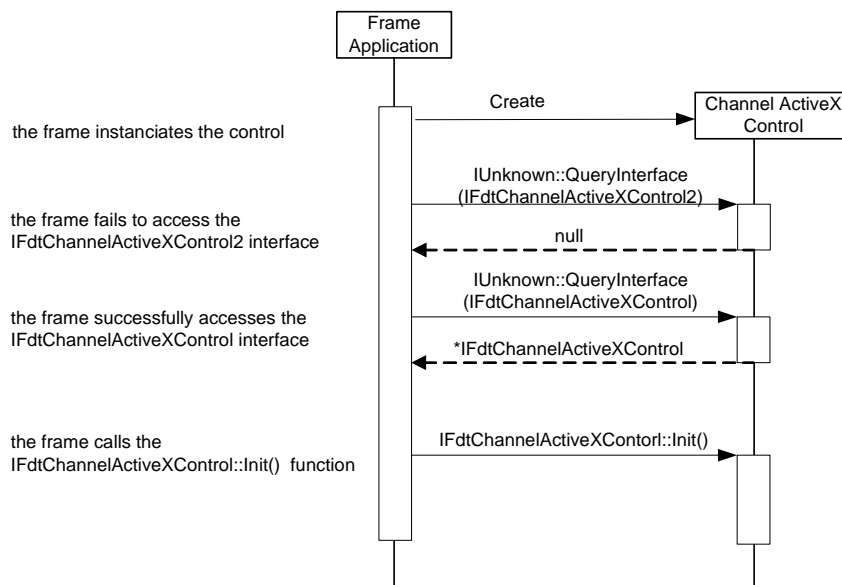
#### Used methods:

IUnknown::QueryInterface

[IFdtChannelActiveXControl2::Init2\(\)](#)

### 6.24.2 Does Not Support IFdtChannelActiveXControl2

This diagram describes the sequence of the channel ActiveX control initialization for the case that the channel ActiveX control does not provide the IFdtChannelActiveXControl2 interface.



### Used methods:

IUnknown::QueryInterface

[IFdtChannelActiveXControl::Init\(\)](#)

## 6.25 DTM Upgrade

The first step to be resolved during upgrade is to verify if the saved data set can be associated with the new DTM.

Each DTM should expose an unique identifier (UUID) specifying its data set format. A DTM can provide a list of compatible data set formats it can load. These UUIDs are returned as part of [IDtmInformation::GetInformation\(\)](#) method call.

If the ClassID and the ProgID of the DTM is not changed it is up to the DTM to handle version upgrade. The list of supported data set formats may not be considered by the Frame Application when data is loading in this case. Additional check for data compatibility must be done by the DTM when the data is loaded.

### 6.25.1 Saving Data from a DTM to be Upgraded

The first step is to store the information from the DTM that is for upgrade.

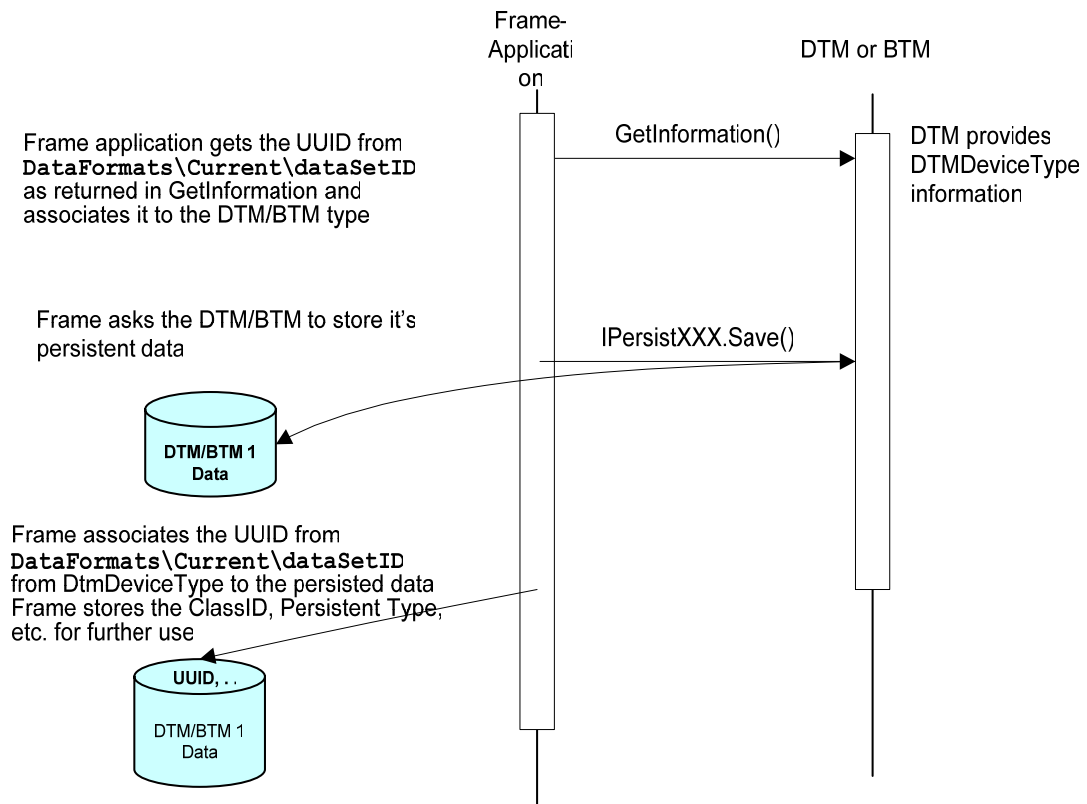
The Frame Application needs to store additional information about the DTM:

- ClassID of the DTM
- UUID of the stored data format
- Storage Type
- Additional information about the device.

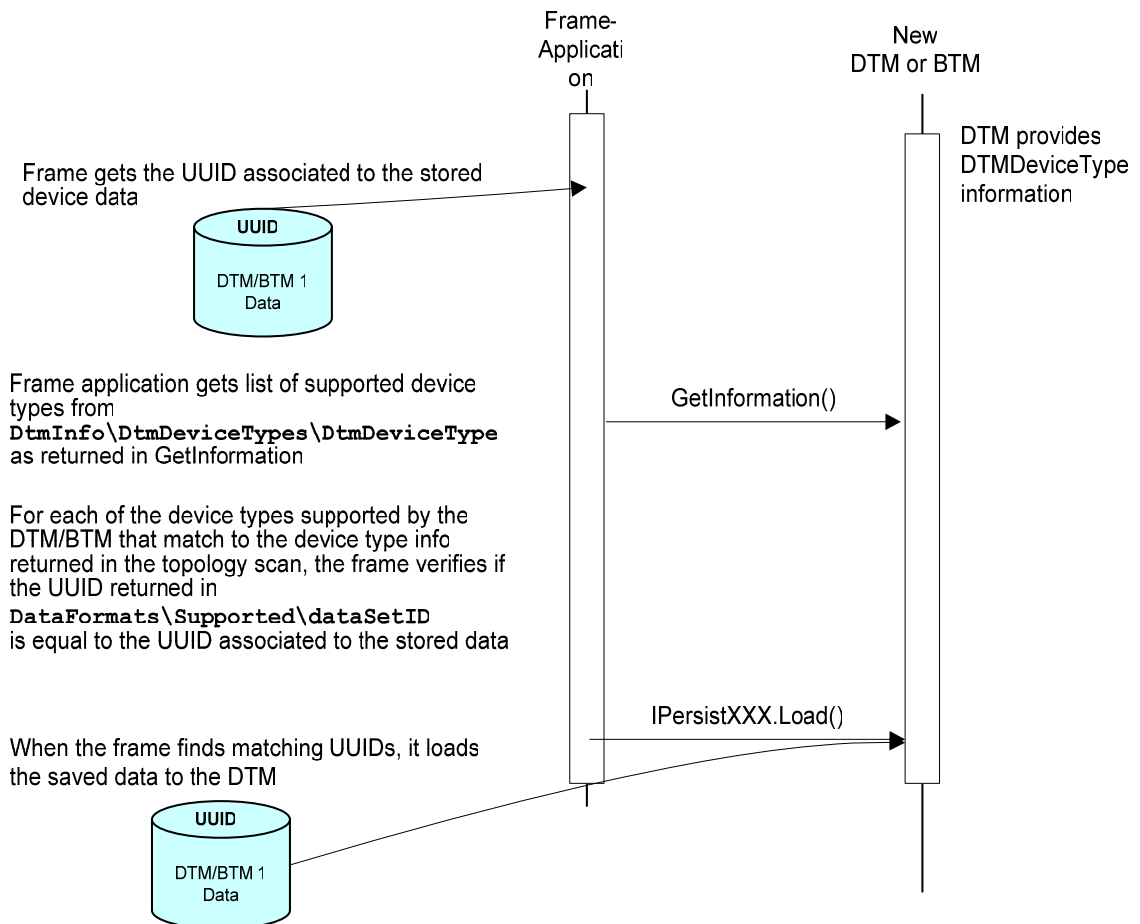


This information is associated to the stored data and can be used later by the Frame Application to identify and manage data set.

The following diagram provides an illustration of that sequence:

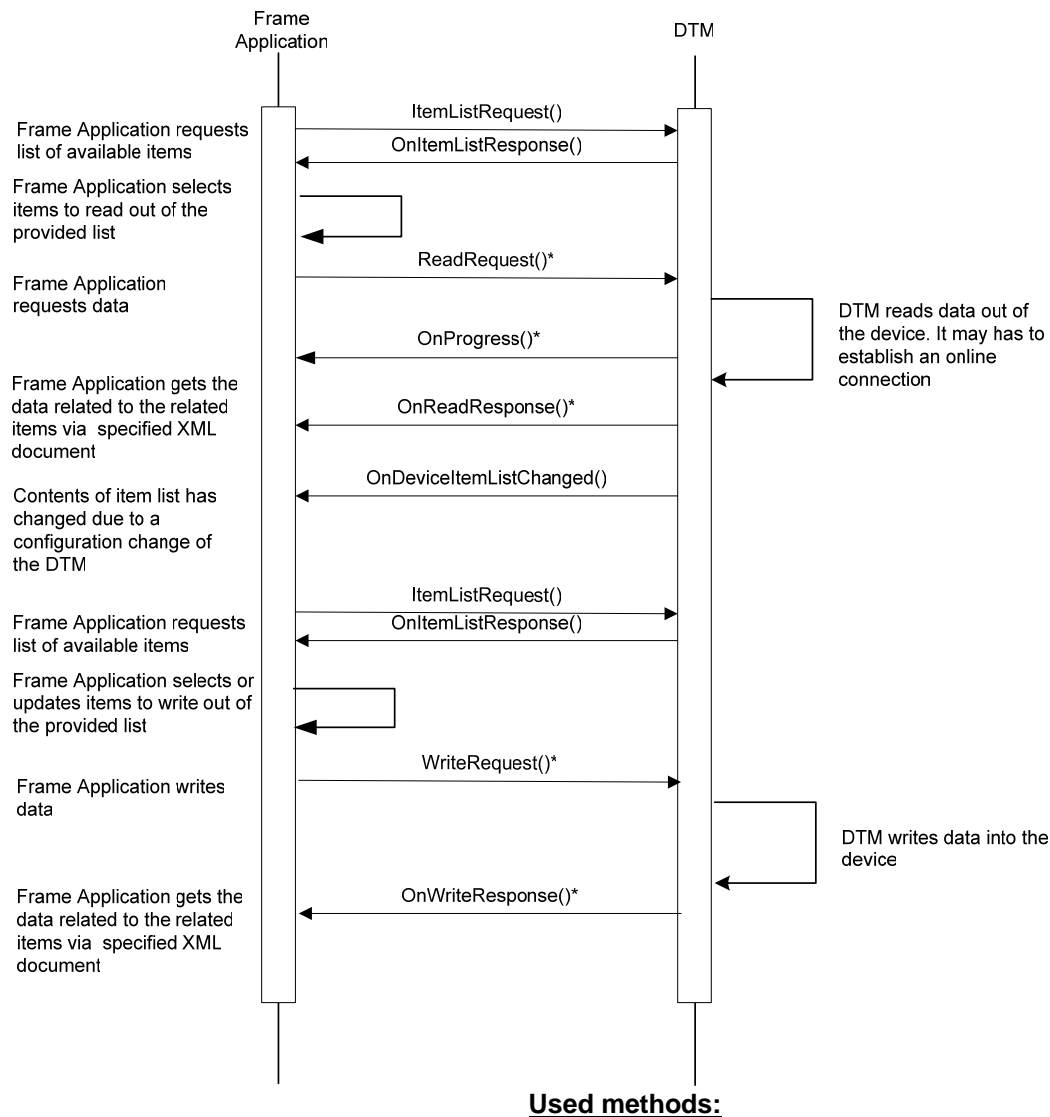


## 6.25.2 Loading Data in the Replacement DTM



## 6.26 Usage of *IDtmSingleDeviceDataAccess::ReadRequest / Write Request*

This sequence chart gives an example regarding handling of [ReadRequest\(\)](#) and [WriteRequest\(\)](#).



[IDtmSingleDeviceDataAccess::ItemListRequest\(\)](#)

[IDtmSingleDeviceDataAccessEvents::OnItemListResponse\(\)](#)

[IDtmSingleDeviceDataAccess::ReadRequest\(\)](#)

[IDtmSingleDeviceDataAccess::WriteRequest\(\)](#)

[IDtmSingleDeviceDataAccessEvents::OnReadResponse\(\)](#)

[IDtmSingleDeviceDataAccessEvents::OnWriteResponse\(\)](#)

[IDtmEvents::OnProgress\(\)](#)

## 6.27 Instantiation of DTM and BTM

A BTM is created by the same mechanism as a DTM, which means the Frame Application always creates a BTM. If a DTM must create a BTM, it has to use the interface IFdtTopology of the Frame Application. BTMs are instantiated by:

- Frame Application according to the defined sequence defined
- DTM triggers as described below.

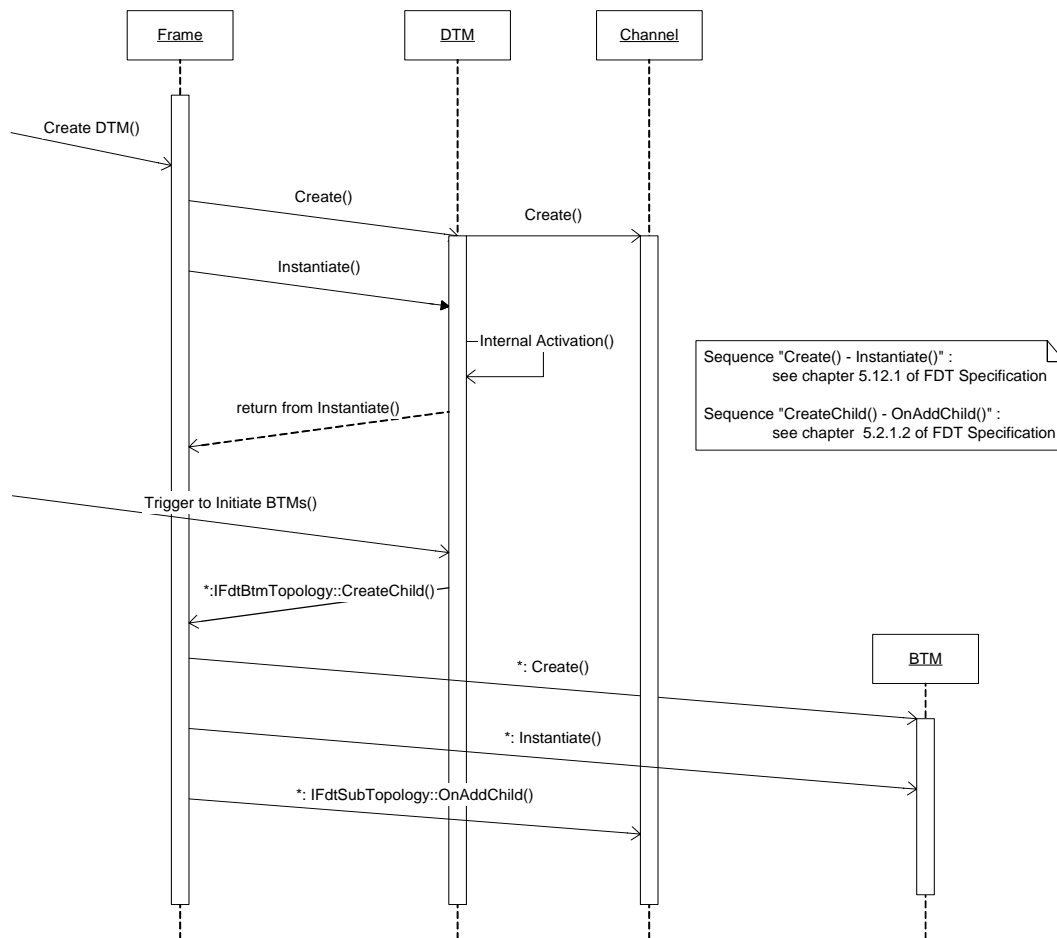
The verification of assigned Child-BTMs is done by using the [ValidateAddChild\(\)](#) method of the IFdtChannelSubTopology interface.

The general sequence is shown in the following chart. The creation of BTMs is possible in any of the following states:

- Running
- Configured
- Communication set
- Going Online
- Going Offline
- Online.

The trigger for creating BTMs (shown in the chart by the event “Trigger to Initiate BTMs”) can originate from the following sources:

- The DTM GUI (e.g. the GUI of the running DTM provides a method “Add block”)
- A function exposed by the DTM (that is, a function without GUI)
- An event on the DTM (e.g., transition from state “**Running**” to state “**Configured**”)



*General sequence of creation and instantiation of blocks*

The Frame Application can reject the CreateChild method. In this case, the procedure of creation of the BTM is aborted. It is up to the user to create the BTM.

It does not matter how to trigger BTMs initiation. The general sequence of events does not change. Thus, each DTM and each BTM is handled according to the DTM State Machine.

If the DTM is loaded (transition from **Up** to **Existing Created** to **Running** states), it shall not automatically trigger the BTM creation. The Frame Application should handle the instantiation of the BTMs.

## 7 Installation Issues

### 7.1 FDT Registry and Device Information

#### 7.1.1 Visibility of business objects of a DTM

Each business object class within a DTM which should be visible for integration within the Frame Application, or a separate DTM presentation object must be registered in the Windows Registry using FDT-specific COM-component category entries. These class objects of a DTM can then be detected by a Frame Application or a configuration tool using the Microsoft standard component category manager. The BTM to DTM assignment follows the same model as the assignment of module Device-DTM to DTM. Vendors are encouraged to define a unique CATID for the protocol between DTM and BTM in order to ensure correct block assignment. The same CATID can be used in different devices if the same BTMs are used.

#### 7.1.2 Component Categories

FDT defines the following Component Categories.

CATID description in the registry	SYMBOLIC NAME OF THE CATID	UUID of the CATID	Description
"FDT DTM"	CATID_FDT_DTM	{036D1490-387B-11D4-86E1-00E0987270B9}	Object compatible to FDT major version 1 providing class information via <a href="#">IDtmInformation</a> .
"FDT DTM Device"	CATID_FDT_DEVICE	{036D1491-387B-11D4-86E1-00E0987270B9}	Device object of a DTM for integration within a Frame Application
"FDT DTM Module"	CATID_FDT_MODULE	{036D1492-387B-11D4-86E1-00E0987270B9}	Module object of a DTM for integration within a Frame Application
"FDT BTM"	CATID_FDT_BTM	{036D1690-387B-11D4-86E1-00E0987270B9}	Object represents a block

A CATID consist of its symbolic name and the UUID within the registry. The FDT IDL defines a symbolic name, e.g., CATID\_FDT\_DTM.

The following table shows the valid combination of category ids:

SYMBOLIC NAME OF THE CATID	CATID_FDT_DTM	CATID_FDT_DEVICE	CATID_FDT_MODULE
CATID_FDT_DTM		√	√
CATID_FDT_DEVICE	√		
CATID_FDT_MODULE	√		
CATID_FDT_BT			

For example, class objects must register for categories according to the following list:

Description	Categories
DTM for a device	CATID_FDT_DTM CATID_FDT_DEVICE
DTM for a module of a modular device	CATID_FDT_DTM CATID_FDT_MODULE

It is expected that a DTM will first create any categories it uses and then registers for those categories during installation. Unregistering a server should cause it to be removed from that category but not to unregister the category by itself. See the `lcatRegister` documentation for additional information.

A DTM may register additional component categories according to COM rules.

### 7.1.3 Registry Entries

Each object (device, module, channel, class, presentation) within a DTM must provide all COM required registry entries within `HKEY_CLASSES_ROOT` and must support self-registration.

### 7.1.4 Installation Issues

It is assumed that the DTM vendor will provide a `SETUP.EXE` to install the needed components for his DTM. This will not be discussed further. Other than the actual components, the main issue affecting COM software is management of the Windows Registry and Component Categories.

All FDT components must have a version information resource containing at least a version number, so that an installation tool can decide whether it can overwrite an installed component or not.

Furthermore a Frame Application is responsible to install all FDT related XML schemas. A DTM has to use these documents provided by the Frame Application via [IFdtContainer::GetXMLSchemaPath\(\)](#). DTM specific schemas within FDT related XML documents must be declared as an inline definition. During the de-installation of FDT related components, the procedure has to take care about the availability of the FDT related interface description. To avoid problems the usage of Microsoft Installer Technology is mandatory. This means all Merge Modules provided by FDT-JIG must be used. E.g. it is not allowed to copy the FDT100.dll directly on the PC. Furthermore all common DLLs (e.g. for usage of ATL, VB runtime or third party controls) must be installed via a Merge Modules if provided by the vendor of the DLL.

Furthermore it is highly recommended not to include the FDT specific Interface description (IDL) within own components.

### 7.1.5 Microsoft's Standard Component Categories Manager

Using the Microsoft Standard Component Categories Manager a Frame Application is able to query a list of all available DTMs or a list of DTMs with a set of specific categories by using the interface method [IcatInformation::EnumClassesOfCategories](#).

### 7.1.6 Building a Frame Application-Database of Supported Devices

Using component categories a Frame Application is able to detect all installed DTMs. Additional information, e.g. vendor information, list of supported devices, can be determined by instantiating a DTM and using the [IDtmInformation](#) and [IDtmInformation2](#) interfaces.

By using this information, a Frame Application is able to build a database with

- All available DTMs
- All available BTMs
- DTMs supporting a specific fieldbus
- Supported field devices
- Etc.

Based on this information it is possible to generate a mapping between specific field devices and supporting DTMs. This functionality is described in chapter 4.10.8.

### 7.1.7 DTM Registration

**DTMs** have to write registry entries whenever the DTM is

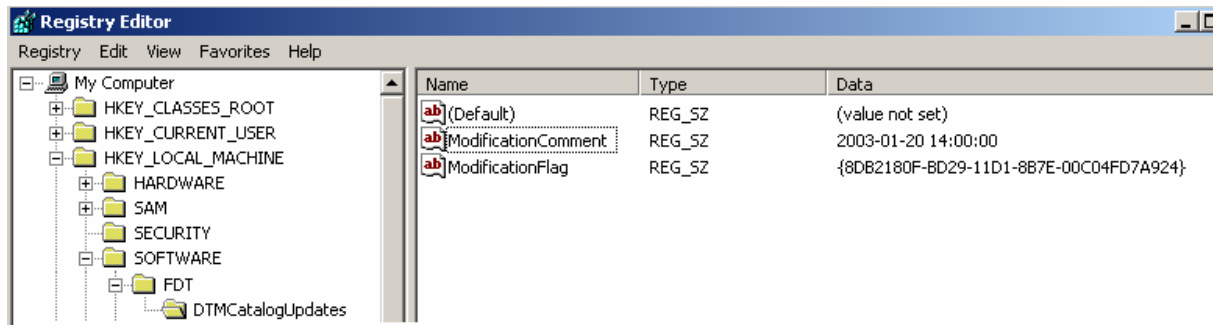
- Installed
- Uninstalled
- Modified, (e.g. new device types are supported or the DTM was updated (bug fix))

If the path doesn't exist, the first installed DTM has to create the path.

A FDT root path is defined for FDT in windows registry:



## DTM Registration – Path in registry

[HKEY\_LOCAL\_MACHINE\SOFTWARE\FDT\DTMCatalogUpdates]<sup>1</sup>

Key	Format	Description
ModificationComment	REG_SZ (String Value)	<b>Mandatory</b> Any string, e.g.: Timestamp of modification. Empty string allowed. <sup>2</sup>
ModificationFlag	REG_SZ (String – GUID)	<b>Mandatory</b> GUID created by DTM setup during setup runtime.

Frame Applications may update an DTM Library and save the last known ModificationFlag entry internally.

A Frame can easily check if there is a need to update the DTM Catalog by comparing last known entry with current entry.

**Comment:**

It was seen as acceptable to leave the entry in registry without a concept for a cleanup because there may be a number of Frame Applications installed on one PC.

Frame Applications are only allowed to read these keys. DTMs must update ModificationFlag and ModificationComment during setup.

<sup>1</sup> HKEY\_LOCAL\_MACHINE allows DTMs to write a registry key and string. During DTM setup user has to have Administrator rights anyway, so there is no missing right. While Frame session, a user can have normal rights. A Frame must only read the registry key so the missing right is not a problem.

<sup>2</sup> Instead of setting this attribute to optional, it's mandatory and allowed to set an empty string. This avoids having old comment together with new flag.

## 8 Description of Data Types, Parameters and Structures

### 8.1 Ids

Ids are unique identifiers in a specific context. They are used to identify components, notification about user roles and rights, and for the association of asynchronous function calls.

Name	Data type	Description
invokeld	<a href="#">FdtUUIDString</a>	An invokeld of a request method of a server object is a unique identifier to be generated by the client via CoCreateGuid() API and is only valid within the calling object. Within a callback method this identifier must be used by the client to identify the appropriate request send to the server object. Association of asynchronous function calls is used for methods like xxxRequest(), OnxxxResponse()
systemTag	BSTR	Unique identifier of a device instance within a project of the Frame Application. The system tag will be set during the initialization of a DTM and has to be used by the DTM for all instance specific function calls to the Frame Application. The DTM must not store the systemTag (e.g. in the instance data set). The DTM may not rely on the fact that it receives the same systemTag during each initialization (call of <a href="#">IDtm::Environment()</a> or <a href="#">IDtm2::Environment2()</a> )
CommunicationReference	<a href="#">FdtUUIDString</a>	Mandatory identifier for a communication link to a device. This identifier is allocated by the communication component during the connect via CoCreateGuid() API. The communication reference has to be used for all following communication calls.

## 8.2 Data Type Definitions

### Helper objects for documentation

Name	Data type	Description
FdtUUIDString	BSTR	<p>String containing a unique identifier according to the Microsoft standard UUID.</p> <p>The format must be e.g. "C2137DD1-7842-11d4-A3C9-005004DC410F" (without bracket).</p> <p>Due to the definition of the UUID format, the value must NOT be handled in a case sensitive way. That means comparing "C2137dd1-7842-11d4-A3C9-005004DC410F" with "C2137DD1-7842-11d4-A3C9-005004DC410f" will return TRUE</p>
FdtXmlDocument	BSTR	String containing an XML document
FdtXPath	BSTR	<p>String containing an Xpath to an element of an XML document</p> <p><b>For FDT 1.2 and FDT 1.2.1 the Xpath has to be the root tag "FDT"</b></p> <ul style="list-style-type: none"> <li>XML documents are accepted as complete document. Otherwise the DTM informs the Frame Application via the event interface about the occurred of errors</li> </ul> <p><b>Exceptions:</b></p> <ul style="list-style-type: none"> <li><a href="#">IFdtChannel::GetChannelPath()</a>, refer to method description</li> <li>Interface IFdtTopology, usage as channel path within the specified methods</li> </ul>
All boolean parameters	VARINAT_BOOL	TRUE and FALSE according to the definition of VARIANT_TRUE and VARIANT_FALSE

## 9 Glossary of Terms

### In the sense of this specification:

ActiveX	Component technology based on the Microsoft Component Object Model (COM/DCOM). Former standard was OLE controls (OCX)
Addressing	An address is a communication protocol specific identifier. Due to that the namespace, the format, ... is defined by the given protocol
Channel	Channels describe process values and their properties which are available via fieldbus communication. Usually the access to these data is not done via a DTM, but the channel description within the public data allows a Frame Application to configure its controller for a protocol specific access to these data.
Communication	Fieldbus protocol specific data transfer between a DTM and a device or between two devices. Communication can be caused by user action or by device internal processes
Configuration	System configuration created by configuring the plant components and the topology
Configure	To configure means setting parameters at the instance data set as well as the logical association of plant components to build up the plant topology (offline)
Connection	Established data path for communication with an selected device
Data	See transient- and persistent data
DCS	Distributed control system
DCS Manufacturer / System Manufacturer	In this context the manufacturer of the engineering system
Device	A piece of hardware which can be connected to a FIELDBUS system. It can be either a master, a slave, a bus coupler, or...
Device Manufacturer	In this context a manufacturer of fieldbus devices, usually slave devices.
Document	To document means to generate the documentation
Documentation	Documentation is the human readable information about a device instance. Meant is in the context of FDT the printed documentation and the documentation provided via XML as well. The documentation can consist of several documents.
DTM	Device Type Manager. A software component to handle a field device. This software component contains business rules to handle the field device's logic. Optionally, it contains the visual user interface.
Fielddevice	Sensor or actor for measuring or positioning as a plant component
Frame Application	Represents the components which build the DTM environment. This can be an engineering tool, a standalone tool or a web page
FDT Model	A model to describe the interaction between DTM and Frame Application
Communication Channel	A communication channel is a channel a device can be connected to. In general such a channel is the master of an underlying fieldbus system.
GSD	A text file containing a description of e.g. the PROFIBUS device with a predetermined syntax. It is delivered with e.g. the PROFIBUS device or can be obtained from the manufacturer.
MIDL	Microsoft Interface Definition Language.
Multi user environment	This term is used to specify an environment which allows that more than one DTM instance can get access to an identical instance data set
Net	A single bus system or a group of connected bus systems
Nested Communication	Communication path built up by a cascaded sequence of communication channels.

OPC	OLE for Process Control
Parameterization	Setting parameters at a device according to the task of the device (online)
Persistent Data	Permanent stored data
PLC	Programmable Logic Controller
Process	Industrial process accessed through field devices. Data acquisition and control.
Project Design	Configuration of devices and process units to build up a net
SCADA	Supervisory/System control and data acquisition
Session	A session encapsulates one or more data transactions. These transactions can be changes initiated by a DTM (e.g. during configuration) or by a Frame Application (e.g. changes within the topology). It is in the responsibility of the Frame Application to guarantee the data consistency within a session
Transient Data	Temporary data during configuration. Turn to persistent data after storage
UML	Unified Modeling Language
XML	Extensible Markup Language

## 10 Literature

Title	Version	Date
OPC Data Access Custom Interface Specification	2.03	July 27, 1999
Microsoft MSDN Library		March 2000
World Wide Web Consortium (W3C) Extensible Stylesheet Language (XSL) Document: WD-xsl-19981216	1.0	16-December-1998
XML kompakt Hanser Verlag ISBN 3-446-21302-3		1999
XML Handbuch Prentice Hall ISBN 3-8272-9575-0		1999
XML/XSL Examples Refer to MSDN Library and W3C < <a href="http://www.w3.org">http://www.w3.org</a> >		
DIN 19245 Teil3. Messen, Steuern, Regeln; PROFIBUS, Process Field Bus; Kommunikations-Modell, Anwendungsfunktionen, Protokoll, Codierung, Benutzerschnittstelle für die schnelle Kommunikation im Bereich der dezentralen Peripherie. Entwurf,		1994
PROFIBUS Nutzerorganisation e.V., PROFIBUS Guideline, Order-Nr. 2.0082, Technical Guideline, PROFIBUS – DP Extensions to EN 50170 (DPV1),	2.0	April 1998
HART SMART COMMUNICATION PROTOCOL SPECIFICATION	5.1	4 January 1991
PROFIBUS Profile for Safety Technology	1.20	23-Oct-2002
IEC 61158-5: Digital data communications for measurement and control Fieldbus for use in industrial control systems Part 5: Application layer service definition,		Third edition 200305

## 11 Appendix – FDT IDL

**NOTE: This IDL file should never be modified in any way. The standard marshaller can be used based on a type library generated from this IDL. If you add vendor specific interfaces to your application (which is allowed) you must generate a SEPARATE vendor specific IDL file to describe only those interfaces and a separate vendor specific ProxyStub DLL to marshal only those interfaces.**

```

/*****
 *
 * FDT 1.2.1 Interfaces
 *
 *****/

[
  uuid(036D1471-387B-11D4-86E1-00E0987270B9),
  version(1.20100)
]
library Fdt100
{
  importlib("STDOLE2.TLB");

  // FDT Datatypes
  typedef [uuid(036D1472-387B-11D4-86E1-00E0987270B9), version(1.0)] BSTR FdtUUIDString;
  typedef [uuid(036D1473-387B-11D4-86E1-00E0987270B9), version(1.0)] BSTR FdtXmlDocument;
  typedef [uuid(036D1474-387B-11D4-86E1-00E0987270B9), version(1.0)] BSTR FdtXPath;

  // Forward declaration of all required interfaces
  interface IFdtContainer;
  interface IFdtChannelCollection;
  interface IFdtCommunication;

  //
  // DTM Interfaces
  // =====

// IDtm
[
  odl,
  uuid(036D1481-387B-11D4-86E1-00E0987270B9),
  dual,
  oleautomation
]
interface IDtm : IDispatch {
  [id(0x1)]
  HRESULT Environment(
    [in] BSTR systemTag,
    [in] IFdtContainer* container,
    [out, retval] VARIANT_BOOL* result);

  [id(0x2)]
  HRESULT InitNew(
    [in] FdtXmlDocument deviceType,
    [out, retval] VARIANT_BOOL* result);

  [id(0x3)]
  HRESULT Config(
    [in] FdtXmlDocument userInfo,
    [out, retval] VARIANT_BOOL* result);

  [id(0x4)]
  HRESULT SetCommunication(
    [in] IFdtCommunication* communication,
    [out, retval] VARIANT_BOOL* result);

  [id(0x5)]
  HRESULT PrepareToRelease(
    [out, retval] VARIANT_BOOL* result);

  [id(0x6)]
  HRESULT PrepareToReleaseCommunication(
    [out, retval] VARIANT_BOOL* result);

  [id(0x7)]

```

```

        HRESULT ReleaseCommunication(
                                [out, retval] VARIANT_BOOL* result);
    [id(0x8)]
    HRESULT PrepareToDelete(
                                [out, retval] VARIANT_BOOL* result);
    [id(0x9)]
    HRESULT SetLanguage(
        [in] long languageId,
        [out, retval] VARIANT_BOOL* result);
    [id(0xa)]
    HRESULT GetFunctions(
        [in] FdtXmlDocument operationState,
        [out, retval] FdtXmlDocument* result);
    [id(0xb)]
    HRESULT InvokeFunctionRequest(
        [in] FdtUUIDString invokeld,
        [in] FdtXmlDocument functionCall,
        [out, retval] VARIANT_BOOL *result);
    [id(0xc)]
    HRESULT PrivateDialogEnabled(
        [in] VARIANT_BOOL enabled,
        [out, retval] VARIANT_BOOL *result);
};

// IDtmActiveXInformation
[
    odl,
    uuid(036D1480-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IDtmActiveXInformation : IDispatch {
    [id(0x1)]
    HRESULT GetActiveXGuid(
        [in] FdtXmlDocument functionCall,
        [out, retval] FdtUUIDString* result);

    [id(0x2)]
    HRESULT GetActiveXProgId(
        [in] FdtXmlDocument functionCall,
        [out, retval] BSTR* result);
};

// IDtmApplication
[
    odl,
    uuid(036D147E-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IDtmApplication : IDispatch {
    [id(0x1)]
    HRESULT StartApplication(
        [in] FdtUUIDString invokeld,
        [in] FdtXmlDocument functionCall,
        [in] BSTR windowTitle,
        [out, retval] VARIANT_BOOL* result);

    [id(0x2)]
    HRESULT ExitApplication(
        [in] FdtUUIDString invokeld,
        [out, retval] VARIANT_BOOL* result);
};

// IDtmChannel
[
    odl,
    uuid(036D1489-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IDtmChannel : IDispatch {
    [id(0x1)]
    HRESULT GetChannels(
                                [out, retval] IFdtChannelCollection** result);
};

```



```
// IDtmDocumentation
[
    odl,
    uuid(036D147C-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IDtmDocumentation : IDispatch {
    [id(0x1)]
    HRESULT GetDocumentation(
        [in] FdtXmlDocument functionCall,
        [out, retval] FdtXmlDocument* result);
};

// IDtmDiagnosis
[
    odl,
    uuid(036D1476-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IDtmDiagnosis : IDispatch {
    [id(0x1)]
    HRESULT Verify([out, retval] VARIANT_BOOL* result);
    [id(0x2)]
    HRESULT InitCompare(
        [in] BSTR systemTag,
        [out, retval] VARIANT_BOOL* result);
    [id(0x3)]
    HRESULT Compare(
        [out, retval] VARIANT_BOOL* result);
    [id(0x4)]
    HRESULT ReleaseCompare(
        [out, retval] VARIANT_BOOL* result);
};

// IDtmImportExport
[
    odl,
    uuid(036D148E-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IDtmImportExport : IDispatch {
    [id(0x1)]
    HRESULT Import(
        [in] IStream* stream,
        [out, retval] VARIANT_BOOL* result);
    [id(0x2)]
    HRESULT Export(
        [in] IStream* stream,
        [out, retval] VARIANT_BOOL* result);
};

// IDtmInformation
[
    odl,
    uuid(036D147F-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IDtmInformation : IDispatch {
    [id(0x1)]
    HRESULT GetInformation(
        [out, retval] FdtXmlDocument* result);
};

// IDtmOnlineDiagnosis
[
    odl,
    uuid(036D1477-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IDtmOnlineDiagnosis : IDispatch {
```

```

        [id(0x1)]
        HRESULT Compare(
                                [out, retval] FdtXmlDocument* result);

        [id(0x2)]
        HRESULT GetDeviceStatus(
                                [out, retval] FdtXmlDocument* result);
    };

// IDtmOnlineParameter
[
    odl,
    uuid(036D1483-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IDtmOnlineParameter : IDispatch {
    [id(0x1)]
    HRESULT CancelAction(
                                [in] FdtUUIDString invokeld,
                                [out,retval] VARIANT_BOOL* result);

    [id(0x2)]
    HRESULT DownloadRequest(
        [in] FdtUUIDString invokeld,
        [in] FdtXPath parameterPath,
        [out,retval] VARIANT_BOOL* result);

    [id(0x3)]
    HRESULT UploadRequest(
        [in] FdtUUIDString invokeld,
        [in] FdtXPath parameterPath,
        [out,retval] VARIANT_BOOL* result);
};

// IDtmParameter
[
    odl,
    uuid(036D147D-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IDtmParameter : IDispatch {
    [id(0x1)]
    HRESULT GetParameters(
        [in] FdtXPath parameterPath,
        [out, retval] FdtXmlDocument* result);

    [id(0x2)]
    HRESULT SetParameters(
        [in] FdtXPath parameterPath,
        [in] FdtXmlDocument FdtXmlDocument,
        [out, retval] VARIANT_BOOL* result);
};

//
// DTM event interfaces
// =====

// IFdtCommunicationEvents
[
    odl,
    uuid(036D1485-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IFdtCommunicationEvents: IDispatch {
    [id(0x1)]
    HRESULT OnAbort(
                                [in] FdtUUIDString communicationReference );

    [id(0x2)]
    HRESULT OnConnectResponse(
                                [in] FdtUUIDString invokeld,
                                [in] FdtXmlDocument response);

    [id(0x3)]
    HRESULT OnDisconnectResponse(
                                [in] FdtUUIDString invokeld,
                                [in] FdtXmlDocument response);

    [id(0x4)]

```

```

        HRESULT OnTransactionResponse(
            [in] FdtUUIDString invokeld,
            [in] FdtXmlDocument response);

};

// IFdtEvents
[
    odl,
    uuid(036D1478-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IFdtEvents : IDispatch {
    [id(0x1)]
        HRESULT OnChildParameterChanged(
            [in] BSTR systemTag;

            [id(0x2)]
            HRESULT OnParameterChanged(
                [in] BSTR systemTag,
                [in] FdtXmlDocument parameter);

            [id(0x3)]
            HRESULT OnLockDataSet(
                [in] BSTR systemTag,
                [in] BSTR userName);

            [id(0x4)]
            HRESULT OnUnlockDataSet(
                [in] BSTR systemTag,
                [in] BSTR userName,
                [out, retval] VARIANT_BOOL* result);

};

//
// DTM ActiveX Control interfaces
// =====
//

// IDtmActiveXControl
[
    odl,
    uuid(036D1486-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IDtmActiveXControl : IDispatch {
    [id(0x1)]
        HRESULT Init(
            [in] FdtUUIDString invokeld,
            [in] FdtXmlDocument functionCall,
            [in] IDtm* dtm,
            [out, retval] VARIANT_BOOL* result);

            [id(0x2)]
            HRESULT PrepareToRelease(
                [out, retval] VARIANT_BOOL* result);

};

//
// FDT Channel interfaces
// =====
//

// IFdtChannel
[
    odl,
    uuid(036D1488-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IFdtChannel : IDispatch {
    [id(0x1)]
        HRESULT GetChannelPath(
            [out, retval] FdtXPath* result);

            [id(0x2)]
            HRESULT GetChannelParameters(

```

```

        [in] FdtXPath parameterPath,
        [in] FdtUUIDString protocolId,
        [out, retval] FdtXmlDocument* result);

[id(0x3)]
HRESULT SetChannelParameters(
    [in] FdtXPath parameterPath,
    [in] FdtUUIDString protocolId,
    [in] FdtXmlDocument XmlDocument,
    [out, retval] VARIANT_BOOL* result);

};

// IFdtChannelActiveXInformation
[
    odl,
    uuid(F2BD2970-13FA-470c-A28C-6A5969A04037),
    dual,
    oleautomation
]
interface IFdtChannelActiveXInformation : IDispatch {
    [id(0x1)]
    HRESULT GetChannelActiveXGuid(
        [in] FdtXmlDocument functionCall,
        [out, retval] FdtUUIDString* result);

    [id(0x2)]
    HRESULT GetChannelActiveXProgId(
        [in] FdtXmlDocument functionCall,
        [out, retval] BSTR* result);

    [id(0x3)]
    HRESULT GetChannelFunctions(
        [in] FdtXmlDocument operationState,
        [out, retval] FdtXmlDocument* result);

};

// IFdtCommunication
[
    odl,
    uuid(039ECFC4-9CA8-44e6-944D-B37F288A34D8),
    dual,
    oleautomation
]
interface IFdtCommunication : IDispatch {
    [id(0x1)]
    HRESULT Abort(
        [in] FdtXmlDocument fieldbusFrame);

    [id(0x2)]
    HRESULT ConnectRequest(
        [in] IFdtCommunicationEvents* callBack,
        [in] FdtUUIDString invokeId,
        [in] FdtUUIDString protocolId,
        [in] FdtXmlDocument fieldbusFrame,
        [out, retval] VARIANT_BOOL* result);

    [id(0x3)]
    HRESULT DisconnectRequest(
        [in] FdtUUIDString invokeId,
        [in] FdtXmlDocument fieldbusFrame,
        [out, retval] VARIANT_BOOL* result);

    [id(0x4)]
    HRESULT TransactionRequest(
        [in] FdtUUIDString invokeId,
        [in] FdtXmlDocument fieldbusFrame,
        [out, retval] VARIANT_BOOL* result);

    [id(0x5)]
    HRESULT GetSupportedProtocols(
        [out, retval] FdtXmlDocument *result );

    [id(0x6)]
    HRESULT SequenceBegin(
        [in] FdtXmlDocument fieldbusFrame,
        [out, retval] VARIANT_BOOL* result);

    [id(0x7)]
    HRESULT SequenceStart(
        [in] FdtXmlDocument fieldbusFrame,
        [out, retval] VARIANT_BOOL* result);
};

```

```

[id(0x8)]
HRESULT SequenceEnd(
    [in] FdtXmlDocument fieldbusFrame,
    [out, retval] VARIANT_BOOL* result);

};

// IFdtChannelSubTopology
[
    odl,
    uuid(036D1484-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IFdtChannelSubTopology : IDispatch {
    [id(0x1)]
    HRESULT ScanRequest(
        [in] FdtUUIDString invokeId,
        [out, retval] VARIANT_BOOL* result);

    [id(0x2)]
    HRESULT ValidateAddChild(
        [in] BSTR childsysteTag,
        [out, retval] VARIANT_BOOL* result);

    [id(0x4)]
    HRESULT ValidateRemoveChild(
        [in] BSTR childsysteTag,
        [out, retval] VARIANT_BOOL* result);

    [id(0x5)]
    HRESULT OnAddChild(
        [in] BSTR childsysteTag);

    [id(0x6)]
    HRESULT OnRemoveChild(
        [in] BSTR childsysteTag);

};

// IFdtFunctionBlockData
[
    odl,
    uuid(036D1475-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IFdtFunctionBlockData : IDispatch {
    [id(0x1)]
    HRESULT SelectFBInstance(
        [in] BSTR systemTag,
        [out, retval] VARIANT_BOOL* result);

    [id(0x2)]
    HRESULT GetFBInstanceData(
        [in] BSTR systemTag,
        [out, retval] FdtXmlDocument* result);

};

//
// FDT Channel ActiveX Control interfaces
// =====
//

// IFdtChannelActiveXControl
[
    odl,
    uuid(036D148B-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IFdtChannelActiveXControl : IDispatch {
    [id(0x1)]
    HRESULT Init(
        [in] FdtUUIDString invokeId,
        [in] IFdtChannel* channel,
        [out, retval] VARIANT_BOOL* result);

    [id(0x2)]
    HRESULT PrepareToRelease(
        [out, retval] VARIANT_BOOL* result);

};

```

```

//
// Frame Application interfaces
// =====
//

// IDtmEvents
[
    odl,
    uuid(F15BA42E-BBF1-42ed-8009-7F664A002CFB),
    dual,
    oleautomation
]
interface IDtmEvents : IDispatch {
    [id(0x1)]
    HRESULT OnParameterChanged(
        [in] BSTR systemTag,
        [in] FdtXmlDocument parameter);

    [id(0x2)]
    HRESULT OnErrorMessage(
        [in] BSTR systemTag,
        [in] BSTR errorMessage);

    [id(0x3)]
    HRESULT OnProgress(
        [in] BSTR systemTag,
        [in] BSTR title,
        [in] short percent,
        [in] VARIANT_BOOL show);

    [id(0x4)]
    HRESULT OnUploadFinished(
        [in] FdtUUIDString invokeld,
        [in] VARIANT_BOOL success);

    [id(0x5)]
    HRESULT OnDownloadFinished(
        [in] FdtUUIDString invokeld,
        [in] VARIANT_BOOL success);

    [id(0x6)]
    HRESULT OnApplicationClosed(
        [in] FdtUUIDString invokeld);

    [id(0x8)]
    HRESULT OnFunctionChanged(
        [in] BSTR systemTag);

    [id(0x9)]
    HRESULT OnChannelFunctionChanged(
        [in] BSTR systemTag,
        [in] FdtXPath channelPath);

    [id(0xa)]
    HRESULT OnPrint(
        [in] BSTR systemTag,
        [in] FdtXmlDocument functionCall);

    [id(0xb)]
    HRESULT OnNavigation(
        [in] BSTR systemTag);

    [id(0xc)]
    HRESULT OnOnlineStateChanged(
        [in] BSTR systemTag,
        [in] VARIANT_BOOL onlineState);

    [id(0xd)]
    HRESULT OnPreparedToRelease(
        [in] BSTR systemTag);

    [id(0xe)]
    HRESULT OnPreparedToReleaseCommunication(
        [in] BSTR systemTag);

    [id(0xf)]
    HRESULT OnInvokedFunctionFinished(
        [in] FdtUUIDString invokeld,
        [in] VARIANT_BOOL success);

    [id(0x10)]
    HRESULT OnScanResponse(
        [in] FdtUUIDString invokeld,
        [in] FdtXmlDocument response);

};

```

```
// IDtmAuditTrailEvents
[
    odl,
    uuid(036D1479-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IDtmAuditTrailEvents : IDispatch {
    [id(0x1)]
    HRESULT OnStartTransaction(
        [in] BSTR systemTag,
        [out, retval] VARIANT_BOOL* result);

    [id(0x2)]
    HRESULT OnAuditTrailEvent(
        [in] BSTR systemTag,
        [in] FdtXmlDocument logEntry);

    [id(0x3)]
    HRESULT OnEndTransaction(
        [in] BSTR systemTag,
        [out, retval] VARIANT_BOOL* result);
};

// IFdtActiveX
[
    odl,
    uuid(5959f485-2c51-4a55-80a7-dd3c45d8baf2),
    dual,
    oleautomation
]
interface IFdtActiveX : IDispatch {
    [id(0x1)]
    HRESULT OpenActiveXControlRequest(
        [in] BSTR systemTag,
        [in] FdtXmlDocument functionCall,
        [out, retval] VARIANT_BOOL* result);

    [id(0x2)]
    HRESULT CloseActiveXControlRequest(
        [in] FdtUUIDString invokeld,
        [out, retval] VARIANT_BOOL* result);
};

// IFdtBulkData
[
    odl,
    uuid(036D148D-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IFdtBulkData : IDispatch {
    [id(0x60030000)]
    HRESULT GetProjectRelatedPath(
        [in] BSTR systemTag,
        [out, retval] BSTR* result);

    [id(0x60030001)]
    HRESULT GetInstanceRelatedPath(
        [in] BSTR systemTag,
        [out, retval] BSTR* result);
};

// IFdtContainer
[
    odl,
    uuid(036D1487-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IFdtContainer : IDispatch {
    [id(0x1)]
    HRESULT SaveRequest(
        [in] BSTR systemTag,
        [out, retval] VARIANT_BOOL* result);

    [id(0x2)]
    HRESULT LockDataSet(
        [in] BSTR systemTag,
        [out, retval] VARIANT_BOOL* result);
};
```

```

[id(0x3)]
HRESULT UnlockDataSet(
    [in] BSTR systemTag,
    [out, retval] VARIANT_BOOL* result);

[id(0x4)]
HRESULT GetXmlSchemaPath(
    [out, retval] BSTR* result);
};

// IFdtDialog
[
    odl,
    uuid(15C19931-6161-11d4-A0A9-005004011A04),
    dual,
    oleautomation
]
interface IFdtDialog : IDispatch {
    [id(0x1)]
    HRESULT UserDialog(
        [in] BSTR systemTag,
        [in] FdtXmlDocument userMessage,
        [out, retval] FdtXmlDocument* result);
};

// IFdtTopology
[
    odl,
    uuid(036D147A-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
]
interface IFdtTopology : IDispatch {
    [id(0x1)]
    HRESULT GetParentNodes(
        [in] BSTR systemTag,
        [out, retval] FdtXmlDocument* result);

    [id(0x2)]
    HRESULT GetChildNodes(
        [in] BSTR systemTag,
        [in] FdtXPath channelPath,
        [out, retval] FdtXmlDocument* result);

    [id(0x3)]
    HRESULT CreateChild(
        [in] FdtXmlDocument deviceType,
        [in] FdtXPath channelPath,
        [out, retval] BSTR* result);

    [id(0x4)]
    HRESULT DeleteChild(
        [in] BSTR systemTag,
        [in] FdtXPath channelPath,
        [out, retval] VARIANT_BOOL* result);

    [id(0x5)]
    HRESULT GetDtmForSystemTag(
        [in] BSTR systemTag,
        [out,retval] IDtm **result);

    [id(0x6)]
    HRESULT GetDtmInfoList(
        [out, retval] FdtXmlDocument* result);

    [id(0x7)]
    HRESULT MoveChild(
        [in] BSTR systemTag,
        [in] FdtXPath destinationchannelPath,
        [out, retval] VARIANT_BOOL* result);

    [id(0x8)]
    HRESULT ReleaseDtmForSystemTag(
        [in] BSTR systemTag,
        [out, retval] VARIANT_BOOL* result);
};

//
// Collections
// =====

// IFdtChannelCollection
[

```



```

odl,
uuid(E4F31A10-45BF-11d4-BBB3-0060080993FF),
dual,
oleautomation
]
    interface IFdtChannelCollection : IDispatch {
        [propget, id(0x1)]
        HRESULT Item(
                                [in] VARIANT *pvarIndex,
                                [out,retval] IFdtChannel **ppRes);

        [propget, id(0x2)]
        HRESULT Count(
                                [out,retval] long* plCount);
        //support VB FOR EACH syntax via an IEnumVariant
        [propget, id(DISPID_NEWENUM)]
        HRESULT NewEnum(
                                [out,retval] IUnknown** ppEnumVariant);
    };

    //
    // BTM Interfaces
    // =====

// IBtm
[
    odl,
    uuid(96341E37-9611-46ba-80ED-A85BD73BF518),
    dual,
    oleautomation
]
interface IBtm : IDtm {
};

// IBtmInformation
[
    odl,
    uuid(87DCC81C-9F97-46c2-A483-5A89B155204C),
    dual,
    oleautomation
]
interface IBtmInformation : IDispatch {
    [id(0x1)]
    HRESULT GetInformation(
                                [out, retval] FdtXmlDocument* result);
};

// IBtmParameter
[
    odl,
    uuid(43D592CC-5F8E-4eca-9365-8D4749390C55),
    dual,
    oleautomation
]
interface IBtmParameter : IDispatch {
    [id(0x1)]
    HRESULT GetParameters(
        [in] FdtXPath parameterPath,
        [out, retval] FdtXmlDocument* result);

    [id(0x2)]
    HRESULT SetParameters(
        [in] FdtXPath parameterPath,
        [in] FdtXmlDocument FdtXmlDocument,
        [out, retval] VARIANT_BOOL* result);
};

// IBtmActiveXControl
[
    odl,
    uuid(0E0418B4-9BC6-4a28-B980-5D3E7F457E4F),
    dual,
    oleautomation
]
interface IBtmActiveXControl : IDispatch {
    [id(0x1)]

```

```

        HRESULT Init(
            [in] FdtUUIDString invokeId,
            [in] FdtXmlDocument functionCall,
            [in] IBtm* btm,
            [out, retval] VARIANT_BOOL* result);
    [id(0x2)]
    HRESULT PrepareToRelease(
        [out, retval] VARIANT_BOOL* result);
};

// IFdtBtmTopology
[
    odl,
    uuid(18250F40-73FB-4c22-A0F1-DB2A11B3FE8D),
    dual,
    oleautomation
]
interface IFdtBtmTopology : IDispatch {
    [id(0x1)]
    HRESULT GetParentNodes(
        [in] BSTR systemTag,
        [out, retval] FdtXmlDocument* result);

    [id(0x2)]
    HRESULT GetChildNodes(
        [in] BSTR systemTag,
        [in] FdtXPath channelPath,
        [out, retval] FdtXmlDocument* result);

    [id(0x3)]
    HRESULT CreateChild(
        [in] FdtXmlDocument deviceType,
        [in] FdtXPath channelPath,
        [out, retval] BSTR* result);

    [id(0x4)]
    HRESULT DeleteChild(
        [in] BSTR systemTag,
        [in] FdtXPath channelPath,
        [out, retval] VARIANT_BOOL* result);

    [id(0x5)]
    HRESULT GetBtmForSystemTag(
        [in] BSTR systemTag,
        [out, retval] IBtm **result);

    [id(0x6)]
    HRESULT GetBtmInfoList(
        [out, retval] FdtXmlDocument* result);

    [id(0x7)]
    HRESULT MoveChild(
        [in] BSTR systemTag,
        [in] FdtXPath destinationchannelPath,
        [out, retval] VARIANT_BOOL* result);

    [id(0x8)]
    HRESULT ReleaseBtmForSystemTag(
        [in] BSTR systemTag,
        [out, retval] VARIANT_BOOL* result);
};

//
// FDT 1.2.1 Interfaces
// =====

// IFdtActiveX2
[
    odl,
    uuid(1922C2DE-4EE7-4085-878A-80AC6C6728AD),
    dual,
    oleautomation
]
interface IFdtActiveX2 : IDispatch
{
    [id(0x1)]
    HRESULT OpenDialogActiveXControlRequest(
        [in] BSTR systemTag,
        [in] FdtXmlDocument functionCall,
        [out, retval] VARIANT_BOOL* result);

    [id(0x2)]

```

```

        HRESULT OpenChannelActiveXControlRequest(
            [in] BSTR channelPath,
            [in] FdtXmlDocument functionCall,
            [out, retval] VARIANT_BOOL* result);

        [id(0x3)]
        HRESULT CloseChannelActiveXControlRequest(
            [in] FdtUUIDString invokeld,
            [out, retval] VARIANT_BOOL* result);

        [id(0x4)]
        HRESULT OpenDialogChannelActiveXControlRequest(
            [in] BSTR channelPath,
            [in] FdtXmlDocument functionCall,
            [out, retval] VARIANT_BOOL* result);
    };

// IDtm2
[
    odl,
    uuid(51E1F44B-D6A1-423d-B11F-AD38EDE78047),
    dual,
    oleautomation
]
interface IDtm2: IDispatch
{
    [id(0x1)]
    HRESULT Environment2(
        [in] BSTR systemTag,
        [in] IFdtContainer* container,
        [out, retval] VARIANT_BOOL* result);
        [in] FdtXmlDocument frameInfo,

    [id(0x2)]
    HRESULT SetSystemGuiLabel(
        [in] FdtXmlDocument systemGuiLabel,
        [out, retval] VARIANT_BOOL* result);
};

// IDtmRedundancyEvents
[
    odl,
    uuid(04808A4C-90C3-45a7-B69E-034A2FA8314D),
    dual,
    oleautomation
]
interface IDtmRedundancyEvents: IDispatch
{
    [id(0x1)]
    HRESULT OnAddedRedundantChild(
        [in] BSTR systemTag,
        [in] FdtXPath channelPath,
        [out, retval] VARIANT_BOOL* result);

    [id(0x2)]
    HRESULT OnRemovedRedundantChild(
        [in] BSTR systemTag,
        [in] FdtXPath channelPath,
        [out, retval] VARIANT_BOOL* result);
};

// IDtmEvents2
[
    odl,
    uuid(494FFD2B-6E58-4e42-80DB-85EBDF6E2CF5),
    dual,
    oleautomation
]
    interface IDtmEvents2 : IDispatch
    {
        [id(0x1)]
        HRESULT OnStateChanged(
            [in] BSTR systemTag,
            [in] FdtXmlDocument xmldoc);
    };

```

```

// IFdtChannelActiveXControl2
[
    odl,
    uuid(73757F49-F3A6-41f7-BEA0-1A3E59D69D5B),
    dual,
    oleautomation
]
interface IFdtChannelActiveXControl2 : IDispatch
{
    [id(0x1)]
    HRESULT Init2(
        [in] FdtUUIDString invokeld,
        [in] FdtXMLDocument functionCall,
        [in] IFdtChannel* channel,
        [out, retval] VARIANT_BOOL* result);

};

// IDtmScanEvents
[
    odl,
    uuid(515590B9-5177-474a-9310-708A3E785B2B),
    dual,
    oleautomation
]
interface IDtmScanEvents : IDispatch
{
    [id(0x1)]
    HRESULT OnScanResponse(
        [in] FdtUUIDString invokeld,
        [in] FdtXMLDocument response);

    [id(0x2)]
    HRESULT OnScanHardwareResponse (
        [in] FdtUUIDString invokeld,
        [in] FdtXMLDocument response);

};

// IFdtChannelScan
[
    odl,
    uuid(64A4310D-F9EE-411d-B4F9-51D3360DF359),
    dual,
    oleautomation
]
interface IFdtChannelScan : IDispatch
{
    [id(0x1)]
    HRESULT ScanRequest(
        [in] FdtUUIDString invokeld,
        [in] FdtXMLDocument request,
        [out, retval] VARIANT_BOOL* result);

    [id(0x2)]
    HRESULT CancelAction (
        [in] FdtUUIDString invokeld,
        [out, retval] VARIANT_BOOL* result);

};

// IFdtChannelSubTopology2
[
    odl,
    uuid(6359FBF1-D373-4202-90E8-E696C37739D4),
    dual,
    oleautomation
]
interface IFdtChannelSubTopology2 : IDispatch
{
    [id(0x1)]
    HRESULT SetChildrenAdresses(
        [in] FdtXMLDocument dtmDeviceList,
        [out, retval] FdtXMLDocument* result);

};

```

```
// IDtmInformation2
[
    odl,
    uuid(C2934CC6-B72D-4611-890F-1C6531D1F8EB),
    dual,
    oleautomation
]
interface IDtmInformation2 : IDispatch
{
    [id(0x1)]
    HRESULT GetDeviceIdentificationInformation(
        [in] FdtXMLDocument request,
        [in] FdtUUIDString protocolId,
        [out, retval] FdtXMLDocument* response);
};

// IDtmHardwareIdentification
[
    odl,
    uuid(9A1DD233-987C-43d1-9424-DA5C1FC6F292),
    dual,
    oleautomation
]
interface IDtmHardwareIdentification : IDispatch
{
    [id(0x1)]
    HRESULT ScanHardwareRequest (
        [in] FdtUUIDString invokeId,
        [in] FdtXMLDocument request,
        [out, retval] VARIANT_BOOL* result);

    [id(0x2)]
    HRESULT CancelAction (
        [in] FdtUUIDString invokeId,
        [out, retval] VARIANT_BOOL* result);
};

// IFdtCommunicationEvents2
[
    odl,
    uuid(7A0AEF6A-A9E7-4673-8F45-01B8AC28F55D),
    dual,
    oleautomation
]
interface IFdtCommunicationEvents2 : IDispatch
{
    [id(0x1)]
    HRESULT OnConnectResponse2(
        [in] FdtUUIDString invokeId,
        [in] FdtXMLDocument parentInformation,
        [in] FdtXMLDocument response);
};

// IDtmSingleDeviceDataAccess
[
    odl,
    uuid(D67240E4-664B-44b0-B692-A1D1ED3FB8F8),
    dual,
    oleautomation
]
interface IDtmSingleDeviceDataAccess : IDispatch
{
    [id(0x1)]
    HRESULT CancelRequest (
        [in] FdtUUIDString invokeId,
        [out, retval] VARIANT_BOOL* result);

    [id(0x2)]
    HRESULT ItemListRequest(
        [in] FdtUUIDString invokeId);

    [id(0x3)]
    HRESULT ReadRequest(
        [in] FdtUUIDString invokeId,
        [in] FdtXMLDocument itemSelectionList);

    [id(0x4)]
};
```

```

        HRESULT WriteRequest(
                                [in] FdtUUIDString invokeld,
                                [in] FdtXmlDocument itemList);

    };

// IDtmSingleInstanceDataAccess
[
    odl,
    uuid(84F9A19A-7E38-40b5-A311-60B14F30C258),
    dual,
    oleautomation
]
    interface IDtmSingleInstanceDataAccess : IDispatch
    {
        [id(0x1)]
        HRESULT GetItemList(
                                [out, retval] FdtXmlDocument* result);

        [id(0x2)]
        HRESULT Read(
                                [in] FdtXmlDocument itemSelectionList,
                                [out, retval] FdtXmlDocument* result);

        [id(0x3)]
        HRESULT Write(
                                [in] FdtXmlDocument itemList,
                                [out, retval] FdtXmlDocument* result);

    };

// IDtmSingleDeviceDataAccessEvents
[
    odl,
    uuid(2357691C-E69A-4e0a-A8AA-EB7F1D080CEF),
    dual,
    oleautomation
]
    interface IDtmSingleDeviceDataAccessEvents : IDispatch
    {
        [id(0x1)]
        HRESULT OnItemListResponse(
                                [in] FdtUUIDString invokeld,
                                [in] FdtXMLDocument response);

        [id(0x2)]
        HRESULT OnDeviceItemListChanged(
                                [in] BSTR systemTag);

        [id(0x3)]
        HRESULT OnReadResponse(
                                [in] FdtUUIDString invokeld,
                                [in] FdtXMLDocument response);

        [id(0x4)]
        HRESULT OnWriteResponse(
                                [in] FdtUUIDString invokeld,
                                [in] FdtXMLDocument response);

    };

// IDtmSingleInstanceDataAccessEvents
[
    odl,
    uuid(EBC093F1-4F7A-4208-A209-C172E54AB185),
    dual,
    oleautomation
]
    interface IDtmSingleInstanceDataAccessEvents : IDispatch
    {
        [id(0x1)]
        HRESULT OnInstanceItemListChanged(
                                [in] BSTR systemTag);

    };

};

```

## 12 Appendix – FDT XML Schemas

### 12.1 FDTDataTypesSchema

The data type schema is used as a global FDT definition. Data types of this schema are reference via the prefix fdt: within the other schemas. Several data defined as attribute and as element to support the XML features for element and group definitions.

Attribute	Description
alarmType	Identifier for the alarm type to show the association between high- and low alarm and high-high- and low-low-alarm
applicationDomain	Attribute defining the application domain, that applies to provided semanticId. This may be a protocol-specific ID or an other FDT-defined application domain.
address	Parameter Addressing information. The format of the value for this parameter is described for each communication protocol in the corresponding section of the specification. For interoperability reasons this parameter is defined as an optional in the XML schema document. The protocol portion of the specification provides the guidelines for address attribute. Most of the protocols specify that the DTM must expose the addressing information to the Frame Application.
binData	Variable containing binary data
bitLength	Length of a binary variable as bit count
bitPosition	Position of a bit within a enumeration (0 based position)
boolValue	Variable for configured static boolean data like alarm value
busCategory	Unique identifier for a supported bus type like Profibus or HART according to the FDT specific <a href="#">unique</a> identifier
busRedundancy	Number of redundant fieldbus interfaces
byteArray	Variable used to transfer binary data. Binary data can be transferred if the attribute is defined as 'bin.hex'. The value has to be set as string at the DOM. This string has to be generated by the DTM developer, because the 'bin.hex' data type of XML shows the problem, that the last byte gets lost at the non typed contents variable, if the value is set to the nodeTypedValue of the DOM
byteLength	Number of bytes to describe data types like string
channelMode	Type information enumeration for a channel
classificationDomainId	Device classification domain group according to IEC62390 AnnexG. See tables below.
classificationId	Unique identifier for classification of a channel or device according to its primary function. Primarily, attribute 'classificationId' should be selected from the table 'Device classification according to IEC Annex G' and use corresponding 'classificationDomainId' attribute for it. If suitable classification ID does not exist in that table, it should be selected from 'FDT classification ID table' and use it without 'classificationDomainId' attribute. See tables below.
communicationError	Fieldbus protocol independent error occurred during communication. This kind of error is used especially if an error occurred during nested communication. The fieldbus specific communication error is part of the fieldbus specific XML schema Communication errors detected by a communication component, e.g.

Attribute	Description
	<p>a COMM-DTM, must be handled within the XML document returned to the child component</p> <ul style="list-style-type: none"> <li>All errors returned by a device as result of a fieldbus transaction should be returned by protocol specific response read or write response elements, typically by using fieldbus specific status and error attributes.</li> <li>All errors detected by the COMM-DTM must be mapped to one of the fdt:communicationError enumeration values (e.g., abort busy invalidCommunicationReference noConnection noParallelServices noPendingRequest unknownError timeout dtmSpecific notSupportedFeature) and returned as a fdt:CommunciationError element to the child DTM.</li> </ul> <p>A DTM must be able to handle both types of error responses for a transaction request sent to the parent component.</p>
dataSetID	Unique identifier of the data set version
dataSetState	State of an instance data set concerning modifications (refer to chapter 6.14.1.1 Modifications)
dataType	Identifier for the data type of a transferred variable
dataTypeDescriptor	Description of data type
date	Date variable within an element
descriptor	Human readable description within the context of a element
description	A human readable description of the supported and current data set format. Can be used to provide additional information to the user in case of partial level of support
deviceGraphicState	<p>List of possible device states used in the DeviceIcon and DeviceBitmap element. Possible states are:</p> <ul style="list-style-type: none"> <li>device: symbolic representation of the device</li> <li>diagnostic: symbolic representation of device if there is online diagnosis available</li> </ul> <p>oem: symbol representation of device in special operating modes (e.g. OEM service)</p> <p>The Frame Application should display the correct picture according to protocol specific rules.</p>
deviceTypeId	Unique identifier for a device type within the name space of the fieldbus specification
deviceTypeInfo	<p>Additional device type information supplied with a device. E.g. a PROFIBUS device has to provide its GSD information as human readable string at this attribute</p> <p>The GSD information must provided based on the current locale according to the usage of <a href="#">IDtm::SetLanguage()</a>.</p> <p>Remark: the GSD information is accessible via</p> <ul style="list-style-type: none"> <li><a href="#">IDtmParameter::GetParameters()</a></li> <li><a href="#">IDtmInformation::GetInformation()</a></li> </ul>
deviceTypeInfoPath	<p>Path to the file containing the information which is provided via the attribute 'deviceTypeInfo'.</p> <p>It is recommended to use this attribute for providing additional protocol specific descriptions of the device type. The use of this attribute is specified in the protocol specific annex.</p>



Attribute	Description
deviceTypeVariant	Manufacturer specific device type variant definition string.
deviceTypeVariantInfo	Contains additional information for manufacturer specific device type variant definition.
display	Carries an human readable display string for tasks like documentation
displayFormat	Describes the display format for a display attribute (e.g. “%3.2f “ for a float value)
documentClassification	Specifies the subject of the document
documentLanguageId	Identifier for the language of the document. The language-id is defined as a Windows locale identifier (LCID)
dtmDevTypeID	<p>dtmDevTypeID is a DTM specific unique identifier of the software related to the device type.</p> <p>For the same device type it's value remains unchanged although some identification information in DtmDeviceType element is updated. This can be a result of DTM update.</p> <p>The same dtmDevTypeID value must always be related to the same device as in the previous DTM version (e.g. to provide backward compatibility).</p> <p><b>It is strongly recommended to provide this attribute!</b></p>
dtmDevTypeIDVersion	<p>Version number of the dtmDevTypeID. It is also used to indicate that the information related to DtmDeviceType is changed. Frame Application can use this information e.g. to update DTM related information in DTM library. <b>It is strongly recommended to provide this attribute!</b></p> <p>An example: When a DTM is DD based, an upgrade of DD will cause the change of dtmDevTypeIDVersion without changing the dtmDevTypeID.</p>
dtmStateMachine	current state of DTM state machine
errorCode	Status information according to the Profibus specification
file	Contains a document or executable file with a absolute path.
help	Human readable help string for a document
Id	Unique identifier for an element. This identifier is used within collections for the direct access of elements. This id must be unique within the namespace of a device instance. The according reference within the XML schemas is the attribute 'idref'.
Idref	Reference to an element specified by the attribute 'id'. The identifier is used within collections for the direct access of elements.
Index	Index within an enumerator
label	Human readable label for a document
levelOfSupport	<p>Provides an indication about the level of support of the supported data set version.</p> <ul style="list-style-type: none"> <li>full – the data set is fully supported</li> </ul> <p>partial – some of the information in the data set can not be used in the upgrade procedure. Additional information should be provided in description attribute. Can be used to warn the user that some values can be lost.</p>
languageId	<p>Identifier for a supported language or the language a DTM should interact in according to the Microsoft standard.</p> <p>See also <a href="#">IDtm::SetLanguage()</a></p>
manufacturerId	Unique identifier for a manufacturer within the name space of the fieldbus specification
modifiedInDevice	<p>Flag to provide more information about current state of the instance data set. Although this flag is an optional attribute, usage of the flag is strongly recommended.</p> <p>TRUE, indicates that parameters have been changed in the device but</p>

Attribute	Description
	<p>not in instance data set (E.g.: see use case Online Parameterization, IDtmSingleDeviceDataAccess interface definition)</p> <p>'modifiedInDevice' flag must be set only once in case the data in the device has been changed.</p> <p>In case of successful Upload or Download of complete data set, 'modifiedInDevice' flag must be reset (set to FALSE).</p> <p>Data in the device may also be modified directly by a tool out of the scope of the FDT. In this case, the flag 'modifiedInDevice' should not be set.</p> <p>Data set must be locked before an application is started, which may need to change the flag "modifiedInDevice" (the flag 'modifiedInDevice' is part of the instance data set and could not be changed if the data set is not locked)</p>
name	Human readable name within the context of a element
nodeId	DTM specific node identifier. Can be used to speed up the access to a node
number	Number variable like float, integer or other numeric data types
orderCode	Order code of the device variant.
parameters	Contains the parameters to be passed to the application, if the file attribute specifies an executable file.
path	Path to the icon for a device
physicalLayer	CATID for description of a physical layer of a fieldbus
physicalLayerName	human readable description for the physical layer
readAccess	Specifies whether the value can be read from a device
reference	Reference to a variable of a structure
semanticId	<p>Semantic identifier for an element. This identifier must be unique at least within the context of an element. By using this attribute e.g. a Frame Application is able to get the information regarding the meaning and usage of a single data structure. The definition regarding the identifier is protocol specific and described in protocol specific annexes</p> <p>Furthermore the following protocol independent semanticIds are defined to cover the network information:</p> <ul style="list-style-type: none"> <li>FDT.slaveAddress</li> <li>FDT.busAddress</li> <li>FDT.busMasterConfigurationPart</li> </ul> <p>A Parameter with one of these semanticId must be the same parameter as the corresponding XML attribute (slaveAddress, busAddress or busMasterConfigurationPart) within in the DTMPParameterSchema represents.</p>
signalType	Specifies a signal as input or output
staticValue	Variable for configured static Data like an alarm value.
statusFlag	Identifier for the current status of a device or module
storageState	State of an instance data set concerning the persistent state (refer to chapter 6.14.1.2 Persistence)
string	String variable within an element
subDeviceType	Manufacturer specific unique identifier for a device type within the name space of the device type id. This parameter must be passed to the DTM during initialization via <a href="#">IDtm::Init()</a> to advise a pre configuration for the requested sub type. For example, the same transmitter can be pre configured for Level or flow measuring.
substituteType	Type of an substitute value

Attribute	Description
systemGuiLabel	Unique human readable identifier of the DTM instance in the context of the Frame Application
tag	Unique identifier for a device, module, or channel
time	Time variable within an element
url	Contains a URL to a document in the Web
vendor	Human readable description of the vendor of component
version	Human readable description of the version of component like "1.0"
writeAccess	Specifies whether the value can be written into a device

Tag	Description
Alarm	Description of an alarm
AlarmEnumerationEntry	Alarm enumeration entry.
AlarmEnumerationEntries	Collection of alarm enumeration entries.
Alarms	Collection of alarms
BinaryVariable	Element containing binary data
BitEnumeratorEntries	Collection of EnumerationEntry
BitEnumeratorVariable	Current EnumeratorEntry and the collection of possible EnumeratorEntries
BitVariable	Selected element of an enumeration
BoolValue	Variable for configured static boolean data
BusCategory	Description of busCategory
BusCategories	Collection of BusCategory
BusRedundancy	Number of redundant fieldbus interfaces
ChannelMode	Type information element for a channel
ChannelModes	List of ChannelMode elements
ChannelInformation	Type information for a FDT channel. This element is recommended within a document returned by a <a href="#">IDtmParameter::GetParameters()</a> call. The BusCategories element should contain the list of supported fieldbus protocols of this channel.
ChannelReference	Reference to an object identified by its id
ChannelReferences	Collection of references
ClassificationId	Classification Id. See tables below
ClassificationIds	Collection of ClassificationId elements. See tables below
CommunicationData	Variable used to transfer binary communication data
CommunicationError	Description of a fieldbus protocol independent error occurred during nested communication with error code, the tag of the communication channel's device and optional error description
Current	A current version of the data set used for saving the data.
DataSetFormats	Data formats of the persisted data used and supported by the DTM
Deadband	Deadband is the amount of value changes that triggers for example new trend values.
DeviceIcon	Icon for a device
DeviceTypeVariant	<p>Contains manufacturer specific device type variant information.</p> <p>A device type variant is a property of the physical device that has no influence to the software (i.e. flange material, Ex certificate ...). However some Frame Applications will use this information for documentation purpose.</p> <p>The DeviceVariant element can occur under DeviceVariants element within context of DTMInformationSchema XMLs or directly under DtmDeviceType elements in context of</p>

Tag	Description
	DTMInitInstanceSchema or DTMPParameterSchema XMLs.
DeviceTypeVariants	Collection of DeviceVariant elements.  The DeviceVariants element should only be used within context of DTMInformationSchema XMLs.
Display	Display variable
DtmDeviceType	Description of a device type
DtmVariable	Variable description with name, value, range, etc
DtmVariables	Collection of DTM variables
DtmVariableReference	Reference to a DTM variable
EnumeratorEntries	Enumeration element
EnumeratorEntry	Element of an enumeration
EnumeratorVariable	Current EnumeratorEntry and the collection of possible EnumeratorEntries
LanguageId	Contains the languageId (refer to languageId)
LowerRange	Definition of a lower range element
LowerRawValue	A numeric entry which reflects the real via e.g. PROFIBUS transferred value. The value is mapped to the related range (LowerRangeValue).  E.g. if a PROFIBUS device maps a range from 10 to 100 mbar to an integer of value 1024-4096 the 'LowerRawValue' contains 1024, the 'UpperRawValue' contains 4096.
NumberData	Number variable like float, integer or other numeric data types
PhysicalLayer	Unique identifier for a physical layer of a fieldbus like Profibus PA
Range	Describes the valid range of a process variable
Ranges	Collection of ranges
SemanticInformation	The element provides semantic information for a data object. For each object at least one <SemanticInformation> element with an FDT-defined protocol-specific semanticId must be provided. The DTM must provide a <SemanticInformation> element for all supported fieldbus protocol of the DTM instance.
StaticValue	Variable for configured static data like an alarm value.
StatusInformation	Current status information of a device or module
StringData	String variable
StructuredElement	Variable as display value or as reference to a variable with data length information
StructuredElements	Collection of structured elements
StructuredVariable	Describes a binary value and its structure information
SubstituteValue	Describes a substitute value which is used in combination of the behavior of disturbed channel values
Supported	A list of the supported data set that can be upgraded by the DTM
SupportedLanguages	Collection of language ids
TimeData	Element of a time date value
Unit	Current unit and the collection of possible units of a process variable
UpperRange	Definition of an upper range element
UpperRawValue	A numeric entry which reflects the real via e.g. PROFIBUS transferred value. The value is mapped to the related range (UpperRangeValue).  E.g. if a PROFIBUS device maps a range from 10 to 100 mbar to an integer of value 1024-4096 the 'UpperRawValue' contains 4096, the 'UpperRange' contains 4096.
Value	Contains the display string for a variable or the variable itself
Variable	Selected element of an enumeration

Tag	Description
Variant	Variable with data type and display format
VersionInformation	<p>Description of the version of a component where used.</p> <p>For example: This element provides the version information of the DTM when used in DtmInfo.</p> <p>It is recommended to use this element to describe the physical device information in the DtmDeviceType. This information should correlate to information provided by SW revision or HW revision in IDtmInformation2::GetDeviceIdentificationInformation().</p>
DtmDocument	Definition of a document, which is provided by a DTM and displayed by the Frame Application.
DocumentFile	Defines a file document
DocumentExe	Defines a executable, which is used to show DTM documents
DocumentUrl	<p>Defines a URL to a document in the Web</p> <p>Implementation hint: The file, url and executable documents can be implemented by using the ShellExecute() function.</p>
DtmDocuments	List of DTM documents. This tag allows a DTM to publish his documents.
DeviceBitmap	Bitmap for a device in BMP format (70*40 pixels (width*height), 16 colors)

FDT classification ID tables

classificationID
flow
level
pressure
temperature
valve
positioner
actuator
nc_rc
encoder
speedDrive
hmi
analogInput
analogOutput
digitalInput
digitalOutput
electrochemicalAnalyser
dtmSpecific
universal
analyser
remoteIO
analogCombinedIO
digitalCombinedIO
recorder
controller
angle
limitSwitch
converter
motor

Device classification according to IEC Annex G.

Domain ()	Subdomain
classificationDomainId	IEC domain group name classificationId

powerDistribution	Power distribution	switchboard
		circuitBreaker
		powerMonitoring
		distributionPanel
motionControl	Motion control	contactor
		protectionStarter
		softStarter
		drive
		axisControl
		motorControlCenter
		motorMonitoring
		positioner
		controlValve
measurement	Detection, measurement	electrical
		density
		flow
		level
		quality
		pressure
		speedOrRotaryFrequency
		radiation
		temperature
		weightMass
		distanceOrPositionOrPresence
operatorInterface	Dialogue / operator interfaces	pushButton
		joystick
		keypad
		pilotLight
		stackLight
		display
		combinedButtonsAndLights
		operatorStation
modulesAndControllers	Logic / universal I/O modules and controllers	generalInput
		generalOutput
		combinedInputOutput
		relay
		timer
		scanner
		programmableController

<Schema name="FDTDataTypesSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes">

<!--Definition of Attributes-->

<AttributeType name="alarmType" dt:type="enumeration" dt:values="highHighAlarm highAlarm lowLowAlarm lowAlarm"/>

<AttributeType name="binData" dt:type="bin.hex"/>

<AttributeType name="bitLength" dt:type="ui4"/>

<AttributeType name="byteArray" dt:type="bin.hex"/>

<AttributeType name="byteLength" dt:type="ui4"/>

<AttributeType name="classificationId" dt:type="enumeration" dt:values="flow level pressure temperature valve positioner

actuator nc\_rc encoder speedDrive hmi analogInput analogOutput digitalInput digitalOutput electrochemicalAnalyser dtmSpecific universal analyser remotelO analogCombinedIO digitalCombinedIO recorder controller angle limitSwitch converter motor switchboard circuitBreaker powerMonitoring distributionPanel contactor protectionStarter softStarter drive axisControl motorControlCenter controlValve electrical density quality speedOrRotaryFrequency radiation weightMass distanceOrPositionPresence pushButton joystick keypad pilotLight stackLight display combinedButtonsAndLights operatorStation generalInput generalOutput combinedInputOutput relay timer scanner programmableController"/>

<AttributeType name="communicationError" dt:type="enumeration" dt:values="abort busy invalidCommunicationReference noConnection noParallelServices noPendingRequest unknownError timeout dtmSpecific notSupportedFeature sequenceTimeExpired"/>

<AttributeType name="dataSetState" dt:type="enumeration" dt:values="default validModified invalidModified allDataLoaded"/>

<AttributeType name="dataType" dt:type="enumeration" dt:values="byte float double int unsigned enumerator bitEnumerator index ascii packedAscii password bitString hexString date time dateAndTime duration binary structured dtmSpecific"/>

<AttributeType name="dataTypeDescriptor" dt:type="string"/>

<AttributeType name="date" dt:type="date"/>

<AttributeType name="descriptor" dt:type="string"/>

<AttributeType name="display" dt:type="string"/>

<AttributeType name="displayFormat" dt:type="string"/>

<AttributeType name="errorCode" dt:type="bin.hex"/>

<AttributeType name="id" dt:type="string"/>

<AttributeType name="idref" dt:type="string"/>

<AttributeType name="index" dt:type="ui4"/>

<AttributeType name="name" dt:type="string"/>

```

<AttributeType name="number" dt:type="number"/>
<AttributeType name="reference" dt:type="string"/>
<AttributeType name="signalType" dt:type="enumeration" dt:values="input output"/>
<AttributeType name="staticValue" dt:type="number"/>
<AttributeType name="statusFlag" dt:type="enumeration" dt:values="ok warning error invalid"/>
<AttributeType name="storageState" dt:type="enumeration" dt:values="persistent transient"/>
<AttributeType name="string" dt:type="string"/>
<AttributeType name="tag" dt:type="string"/>
<AttributeType name="time" dt:type="dateTime"/>
<AttributeType name="vendor" dt:type="string"/>
<AttributeType name="version" dt:type="string"/>
<AttributeType name="nodeId" dt:type="id"/>
<AttributeType name="readAccess" dt:type="boolean" default="1"/>
<AttributeType name="writeAccess" dt:type="boolean" default="0"/>
<AttributeType name="deviceTypeId" dt:type="ui4"/>
<AttributeType name="subDeviceType" dt:type="string"/>
<AttributeType name="deviceTypeInfo" dt:type="string"/>
<AttributeType name="languageId" dt:type="ui4"/>
<AttributeType name="manufacturerId" dt:type="ui4"/>
<AttributeType name="busCategory" dt:type="uiid"/>
<AttributeType name="substituteType" dt:type="enumeration" dt:values="lastValue lastValidValue upperRange lowerRange"/>
<AttributeType name="path" dt:type="uri"/>
<AttributeType name="communicationType" dt:type="enumeration" dt:values="supported required"/>
<AttributeType name="busCategoryName" dt:type="string"/>
<AttributeType name="help" dt:type="string"/>
<AttributeType name="label" dt:type="string"/>
<AttributeType name="file" dt:type="uri"/>
<AttributeType name="url" dt:type="uri"/>
<AttributeType name="parameters" dt:type="string"/>
<AttributeType name="documentLanguageId" dt:type="ui4"/>
<AttributeType name="documentClassification" dt:type="enumeration" dt:values="help technicalDocumentation
orderingInformation miscellaneous"/>
<AttributeType name="deviceGraphicState" dt:type="enumeration" dt:values="device diagnosis oem"/>
<AttributeType name="deviceTypeInfoPath" dt:type="uri"/>
<AttributeType name="systemGuiLabel" dt:type="string"/>
<AttributeType name="busAddress" dt:type="string"/>
<AttributeType name="systemTag" dt:type="string"/>
<AttributeType name="channelMode" dt:type="enumeration" dt:values="communication moduleSlot processValue"/>
<AttributeType name="physicalLayerName" dt:type="string"/>
<AttributeType name="physicalLayer" dt:type="uiid"/>
<AttributeType name="busRedundancy" dt:type="ui4"/>
<AttributeType name="modifiedInDevice" dt:type="boolean" default="0"/>
<AttributeType name="dtmStateMachine" dt:type="enumeration" dt:values="communicationSet goingOnline goingOffline
online"/>
<AttributeType name="orderCode" dt:type="string"/>
<AttributeType name="deviceTypeVariant" dt:type="string"/>
<AttributeType name="deviceTypeVariantInfo" dt:type="string"/>
<AttributeType name="bitPosition" dt:type="ui4"/>
<AttributeType name="boolValue" dt:type="boolean"/>
<AttributeType name="classificationDomainId" dt:type="enumeration" dt:values="powerDistribution motionControl
measurement operatorInterface modulesAndControllers"/>
<AttributeType name="dataSetId" dt:type="uiid"/>
<AttributeType name="description" dt:type="string"/>
<AttributeType name="levelOfSupport" dt:type="enumeration" dt:values="full partial"/>
<AttributeType name="semanticId" dt:type="string"/>
<AttributeType name="address" dt:type="string"/>
<AttributeType name="applicationDomain" dt:type="string"/>
<AttributeType name="dtmDevTypeId" dt:type="uiid"/>
<AttributeType name="dtmDevTypeIdVersion" dt:type="ui4"/>

<!--Definition of Elements-->
<ElementType name="SemanticInformation" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="applicationDomain" required="yes"/>
  <attribute type="semanticId" required="yes"/>
  <attribute type="address" required="no"/>
</ElementType>
<ElementType name="LowerRawValue" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="number" required="yes"/>
</ElementType>
<ElementType name="UpperRawValue" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="number" required="yes"/>
</ElementType>

```



```

<ElementType name="Current" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="dataSetID" required="yes"/>
  <attribute type="description" required="no"/>
</ElementType>
<ElementType name="Supported" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="dataSetID" required="yes"/>
  <attribute type="levelOfSupport" required="no" default="partial"/>
  <attribute type="description" required="no"/>
</ElementType>
<ElementType name="DataSetFormats" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="Current" minOccurs="1" maxOccurs="1"/>
  <element type="Supported" minOccurs="0" maxOccurs="*/>
</ElementType>
<ElementType name="ClassificationId" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="classificationId" required="yes"/>
</ElementType>
<ElementType name="ClassificationIds" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="classificationDomainId" required="no"/>
  <element type="ClassificationId" minOccurs="1" maxOccurs="*/>
</ElementType>
<ElementType name="BoolValue" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="boolValue" required="no"/>
</ElementType>
<ElementType name="Deadband" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="number" required="yes"/>
</ElementType>
<ElementType name="AlarmEnumerationEntry" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="bitPosition" required="yes"/>
  <attribute type="name" required="yes"/>
  <attribute type="boolValue" required="yes"/>
  <attribute type="descriptor" required="no"/>
</ElementType>
<ElementType name="AlarmEnumerationEntries" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="AlarmEnumerationEntry" minOccurs="1" maxOccurs="*/>
</ElementType>
<ElementType name="DeviceTypeVariant" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="deviceTypeVariant" required="yes"/>
  <attribute type="deviceTypeVariantInfo" required="no"/>
  <attribute type="orderCode" required="no"/>
</ElementType>
<ElementType name="DeviceTypeVariants" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="DeviceTypeVariant" minOccurs="1" maxOccurs="*/>
</ElementType>
<ElementType name="BusRedundancy" content="empty" model="closed">
  <attribute type="busRedundancy" required="yes"/>
</ElementType>
<ElementType name="ChannelMode" content="empty" model="closed">
  <attribute required="no" type="nodeId"/>
  <attribute required="yes" type="channelMode"/>
</ElementType>
<ElementType name="ChannelModes" content="eltOnly" model="closed">
  <attribute required="no" type="nodeId"/>
  <element type="ChannelMode" minOccurs="1" maxOccurs="*/>
</ElementType>
<ElementType name="ChannelInformation" content="eltOnly" model="closed">
  <attribute required="no" type="nodeId"/>
  <element type="BusCategories" minOccurs="1" maxOccurs="1"/>
  <element maxOccurs="1" minOccurs="1" type="ChannelModes"/>
</ElementType>
<ElementType name="CommunicationTypeEntry" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="communicationType" required="yes"/>
</ElementType>
<ElementType name="DeviceIcon" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>

```



```

        <attribute type="deviceGraphicState" required="no"/>
        <attribute type="path" required="yes"/>
    </ElementType>
    <ElementType name="DeviceBitmap" content="empty" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="deviceGraphicState" required="yes"/>
        <attribute type="path" required="yes"/>
    </ElementType>
    <ElementType name="SubstituteType" content="empty" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="substituteType" required="yes"/>
    </ElementType>
    <ElementType name="LanguageId" content="empty" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="languageId" required="yes"/>
    </ElementType>
    <ElementType name="SupportedLanguages" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <element type="LanguageId" minOccurs="1" maxOccurs="**"/>
    </ElementType>
    <ElementType name="PhysicalLayer" content="empty" model="closed">
        <attribute type="physicalLayer" required="yes"/>
        <attribute type="physicalLayerName" required="no"/>
    </ElementType>
    <ElementType name="BusCategory" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="busCategory" required="yes"/>
        <attribute type="busCategoryName" required="no"/>
        <element type="CommunicationTypeEntry" minOccurs="0" maxOccurs="**"/>
        <element type="PhysicalLayer" minOccurs="0" maxOccurs="**"/>
    </ElementType>
    <ElementType name="BusCategories" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <element type="BusCategory" minOccurs="1" maxOccurs="**"/>
    </ElementType>
    <!--Definition of a document entry, which specifies a path to a DTM provided document-->
    <ElementType name="DocumentFile" content="empty" model="closed">
        <attribute type="file" required="yes"/>
    </ElementType>
    <ElementType name="DocumentUrl" content="empty" model="closed">
        <attribute type="url" required="yes"/>
    </ElementType>
    <ElementType name="DocumentExe" content="empty" model="closed">
        <attribute type="file" required="yes"/>
        <attribute type="parameters" required="no"/>
    </ElementType>
    <ElementType name="DtmDocument" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="label" required="yes"/>
        <attribute type="help" required="no"/>
        <attribute type="documentClassification" required="yes"/>
        <attribute type="documentLanguageId" required="no"/>
        <group order="one" minOccurs="1" maxOccurs="1">
            <element type="DocumentFile" minOccurs="0" maxOccurs="1"/>
            <element type="DocumentUrl" minOccurs="0" maxOccurs="1"/>
            <element type="DocumentExe" minOccurs="0" maxOccurs="1"/>
        </group>
    </ElementType>
    <ElementType name="DtmDocuments" content="mixed" model="closed">
        <attribute type="nodeId" required="no"/>
        <group order="many">
            <element type="DtmDocument" minOccurs="1" maxOccurs="**"/>
        </group>
    </ElementType>
    <ElementType name="DtmDeviceType" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="readAccess" required="no"/>
        <attribute type="writeAccess" required="no"/>
        <attribute type="manufacturerId" required="no"/>
        <attribute type="deviceTypeId" required="no"/>
        <attribute type="subDeviceType" required="no"/>
        <attribute type="deviceTypeInfo" required="no"/>
        <attribute type="deviceTypeInfoPath" required="no"/>
        <attribute type="classificationId" required="no"/>

```

```

    <attribute type="dtmDevTypeID" required="no"/>
    <attribute type="dtmDevTypeIDVersion" required="no"/>
    <element type="VersionInformation" minOccurs="1" maxOccurs="1"/>
    <element type="SupportedLanguages" minOccurs="1" maxOccurs="1"/>
    <element type="BusCategories" minOccurs="0" maxOccurs="1"/>
    <element type="DeviceIcon" minOccurs="0" maxOccurs="1"/>
    <element type="DtmDocuments" minOccurs="0" maxOccurs="1"/>
    <element type="DeviceBitmap" minOccurs="0" maxOccurs="1"/>
    <element type="ClassificationIds" minOccurs="0" maxOccurs="1"/>
    <element type="DataSetFormats" minOccurs="0" maxOccurs="1"/>
    <group order="one" minOccurs="0" maxOccurs="1">
      <element type="DeviceTypeVariants"/>
      <element type="DeviceTypeVariant"/>
    </group>
  </ElementType>
<!-- Definition of Version Information -->
<ElementType name="VersionInformation" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="readAccess" required="no"/>
  <attribute type="writeAccess" required="no"/>
  <attribute type="name" required="yes"/>
  <attribute type="vendor" required="no"/>
  <attribute type="version" required="no"/>
  <attribute type="date" required="no"/>
  <attribute type="descriptor" required="no"/>
</ElementType>
<ElementType name="NumberData" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="number" required="yes"/>
</ElementType>
<ElementType name="StringData" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="string" required="yes"/>
</ElementType>
<ElementType name="TimeData" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="time" required="yes"/>
</ElementType>
<!-- Definition of Binary Variable -->
<ElementType name="BinaryVariable" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="binData" required="yes"/>
</ElementType>
<!-- Definition of Enumerator Variable -->
<ElementType name="EnumeratorEntry" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="index" required="yes"/>
  <attribute type="name" required="yes"/>
  <attribute type="descriptor" required="no"/>
</ElementType>
<ElementType name="Variable" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="EnumeratorEntry" minOccurs="1" maxOccurs="1"/>
</ElementType>
<ElementType name="EnumeratorEntries" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="EnumeratorEntry" minOccurs="1" maxOccurs="1"/>
</ElementType>
<ElementType name="BitEnumeratorEntries" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="EnumeratorEntry" minOccurs="1" maxOccurs="1"/>
</ElementType>
<ElementType name="EnumeratorVariable" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="Variable" minOccurs="1" maxOccurs="1"/>
  <element type="EnumeratorEntries" minOccurs="0" maxOccurs="1"/>
</ElementType>
<!-- Definition of Bit Enumerator Variable -->
<ElementType name="BitVariable" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="EnumeratorEntry" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="BitEnumeratorVariable" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="BitVariable" minOccurs="1" maxOccurs="1"/>
  <element type="BitEnumeratorEntries" minOccurs="0" maxOccurs="1"/>

```

```

</ElementType>
<!-- Definition of Unit -->
<ElementType name="Unit" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="readAccess" required="no"/>
  <attribute type="writeAccess" required="no"/>
  <group order="one" minOccurs="0" maxOccurs="1">
    <element type="EnumeratorVariable"/>
    <element type="ChannelReference"/>
  </group>
</ElementType>
<!-- Definition of LowerRange -->
<ElementType name="LowerRange" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <group order="one" minOccurs="0" maxOccurs="1">
    <element type="NumberData"/>
    <element type="StringData"/>
    <element type="TimeData"/>
    <element type="ChannelReference"/>
  </group>
</ElementType>
<!-- Definition of UpperRange -->
<ElementType name="UpperRange" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <group order="one" minOccurs="0" maxOccurs="1">
    <element type="NumberData"/>
    <element type="StringData"/>
    <element type="TimeData"/>
    <element type="ChannelReference"/>
  </group>
</ElementType>
<!-- Definition of Ranges -->
<ElementType name="Range" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="LowerRange" minOccurs="1" maxOccurs="1"/>
  <element type="UpperRange" minOccurs="1" maxOccurs="1"/>
  <element type="Unit" minOccurs="0" maxOccurs="1"/>
  <element type="LowerRawValue" minOccurs="0" maxOccurs="1"/>
  <element type="UpperRawValue" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="Ranges" content="eltOnly" model="closed">
  <attribute type="readAccess" required="no"/>
  <attribute type="writeAccess" required="no"/>
  <attribute type="nodeId" required="no"/>
  <element type="Range" minOccurs="1" maxOccurs="*/>
</ElementType>
<!-- Definition of Channel References -->
<ElementType name="ChannelReference" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="idref" required="yes"/>
  <element type="ChannelInformation" maxOccurs="1" minOccurs="0" />
</ElementType>
<ElementType name="ChannelReferences" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="ChannelReference" minOccurs="1" maxOccurs="*/>
</ElementType>
<!-- Definition of Alarms -->
<ElementType name="StaticValue" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="staticValue" required="yes"/>
</ElementType>
<ElementType name="Alarm" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="alarmType" required="yes"/>
  <element type="Unit" minOccurs="0" maxOccurs="1"/>
  <group order="one" minOccurs="0" maxOccurs="1">
    <element type="StaticValue"/>
  </group>
  <element type="NumberData"/>
  <element type="AlarmEnumerationEntries"/>
  <element type="ChannelReferences"/>
</group>
</ElementType>
<ElementType name="Alarms" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="Alarm" minOccurs="1" maxOccurs="*/>
</ElementType>

```

```

<!-- DtmVariable -->
<ElementType name="Display" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="string" required="yes"/>
</ElementType>
<ElementType name="DtmVariableReference" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="reference" required="yes"/>
</ElementType>
<!-- StructuredVariable -->
<ElementType name="StructuredElement" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="bitLength" required="yes"/>
  <group order="one" minOccurs="1" maxOccurs="1">
    <element type="Display"/>
    <element type="DtmVariableReference"/>
  </group>
</ElementType>
<ElementType name="StructuredElements" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="StructuredElement" minOccurs="1" maxOccurs="*" />
</ElementType>
<ElementType name="StructuredVariable" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="BinaryVariable" minOccurs="1" maxOccurs="1" />
  <element type="StructuredElements" minOccurs="1" maxOccurs="1" />
  <attribute type="dataTypeDescriptor" required="no"/>
</ElementType>
<ElementType name="Variant" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="dataType" required="yes"/>
  <attribute type="byteLength" required="no"/>
  <attribute type="displayFormat" required="no"/>
  <group order="one" minOccurs="1" maxOccurs="1">
    <element type="StringData"/>
    <!-- for types: ascii, packedAscii, password, bitString, hexString -->
    <element type="NumberData"/>
    <!-- for types: float, double, int, unsigned, index -->
    <element type="TimeData"/>
    <!-- for types: date, time, dateAndTime, duration -->
    <element type="EnumeratorVariable"/>
    <!-- for type: enumerator -->
    <element type="BitEnumeratorVariable"/>
    <!-- for type: bitEnumerator -->
    <element type="BinaryVariable"/>
    <!-- for types: binary, dtmSpecific -->
    <element type="StructuredVariable"/>
    <!-- for type: structured -->
  </group>
</ElementType>
<!-- SubstituteValue -->
<ElementType name="SubstituteValue" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <group order="one" minOccurs="1" maxOccurs="1">
    <element type="SubstituteType"/>
    <element type="Variant"/>
  </group>
</ElementType>
<!-- Value -->
<ElementType name="Value" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="readAccess" required="no"/>
  <attribute type="writeAccess" required="no"/>
  <group order="one" minOccurs="1" maxOccurs="1">
    <element type="Display"/>
    <element type="Variant"/>
  </group>
</ElementType>
<!-- DTMVariableStatus -->
<ElementType name="StatusInformation" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="readAccess" required="no"/>
  <attribute type="writeAccess" required="no"/>
  <element type="EnumeratorEntry" minOccurs="1" maxOccurs="*" />
</ElementType>
<!-- DtmVariable -->

```

```

<ElementType name="DtmVariable" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="SemanticInformation" minOccurs="0" maxOccurs="*" />
  <attribute type="name" required="yes"/>
  <attribute type="descriptor" required="no"/>
  <element type="Value" minOccurs="1" maxOccurs="1"/>
  <element type="Unit" minOccurs="0" maxOccurs="1"/>
  <element type="Ranges" minOccurs="0" maxOccurs="1"/>
  <attribute type="statusFlag" required="no"/>
  <element type="StatusInformation" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="DtmVariables" content="mixed" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="SemanticInformation" minOccurs="0" maxOccurs="*" />
  <attribute type="name" required="yes"/>
  <attribute type="descriptor" required="no"/>
  <group order="many">
    <element type="DtmVariables" minOccurs="0" maxOccurs="*" />
    <element type="DtmVariable" minOccurs="0" maxOccurs="*" />
  </group>
</ElementType>
<!-- Communication Data -->
<ElementType name="CommunicationData" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="byteArray" required="yes"/>
</ElementType>
<!-- Definition of FDT specic communication errors for nested communication-->
<ElementType name="CommunicationError" content="mixed" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="communicationError" required="yes"/>
  <attribute type="tag" required="yes"/>
  <attribute type="errorCode" required="no"/>
  <attribute type="descriptor" required="no"/>
</ElementType>
</Schema>

```

## 12.2 FDTApplicationIdSchema

The application id schema is used as a global FDT definition. The application id of this schema is reference via the prefix fdtappid: within the other schemas. The appearance and the functionality of a DTM user interface is controlled by the entry of the element applicationId, functionId, and operationPhase.

Attribute	Description
applicationId	Identification of the current application context (The enumeration refers to the DTM Realization Elements)

Tag	Description
ApplicationId	FDT global application id coded as an enumeration
FDTApplicationIds	Collection of application id
FDT	Root tag

```
<Schema name="FDTApplicationIdSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="applicationId" dt:type="enumeration" dt:values="fdtAdjustSetValue fdtAssetManagement fdtAuditTrail
fdtConfiguration fdtDiagnosis fdtForce fdtManagement fdtObserve fdtOfflineCompare fdtOfflineParameterize fdtOnlineCompare
fdtOnlineParameterize fdtIdentify fdtCalibration fdtMainOperation fdtNetworkManagement"/>
  <!-- ApplicationId specifies the standard user interface called -->
  <ElementType name="ApplicationId" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="applicationId" required="yes"/>
  </ElementType>
  <!-- FDTApplicationIds: a list of application -->
  <ElementType name="FDTApplicationIds" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="ApplicationId" minOccurs="0" maxOccurs="*" />
  </ElementType>
  <!-- main FDT element -->
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="FDTApplicationIds" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>
```

Example:

See usage within [DTMFunctionsSchema](#)

## 12.3 FDTUserInformationSchema

Used at: [IDtm::Config\(\)](#)

The user information schema is used as a global FDT definition. It informs the DTM during initialization about the role and the rights of the current user.

Attribute	Description
administrator	Flag describing if the user has administrative right (default to "FALSE")
loginLocation	Description of the location the user logged in
loginTime	Time of last login
oemService	Flag describing if the user has OEM service rights (default to "FALSE")
projectName	Unique identifier for the project within the name space of the Frame Application
sessionDescription	Description of the user session (may be given by the user)
userLevel	User level specifying users rights
userName	Name of the current user

Tag	Description
FDTUserInformation	Description of the user role
FDT	Root tag

```
<Schema name="FDTUserInformationSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!-- Definition of Attributes -->
  <AttributeType name="administrator" dt:type="boolean" default="0"/>
  <AttributeType name="loginLocation" dt:type="string"/>
  <AttributeType name="loginTime" dt:type="dateTime"/>
  <AttributeType name="oemService" dt:type="boolean" default="0"/>
  <AttributeType name="projectName" dt:type="string"/>
  <AttributeType name="sessionDescription" dt:type="string"/>
  <AttributeType name="userLevel" dt:type="enumeration" dt:values="observer operator maintenance
planningEngineer"/>
  <AttributeType name="userName" dt:type="string"/>
  <!-- FDTUserInformation -->
  <!-- the contents of an FDTUserInformation instance lies in the responsibility of the frame-application only -->
  <ElementType name="FDTUserInformation" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="projectName" required="yes"/>
    <!-- project name -->
    <attribute type="userName" required="yes"/>
    <!-- user name -->
    <attribute type="userLevel" required="yes"/>
    <!-- user level as defined within the FDT sepcification -->
    <attribute type="oemService" required="no"/>
    <!-- oemService set to True enables OEM access -->
    <attribute type="administrator" required="no"/>
    <!-- administrator set to True enables administrative access -->
    <attribute type="loginTime" required="no"/>
    <!-- time and date of login for this session -->
    <attribute type="loginLocation" required="no"/>
    <!-- station description at which the user logged in -->
    <attribute type="sessionDescription" required="no"/>
    <!-- descriptive string for the actual session -->
  </ElementType>
  <!-- FDTUserInformation -->
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
  </ElementType>
</Schema>
```

```
<element type="FDTUserInformation" minOccurs="1" maxOccurs="1"/>
</ElementType>
</Schema>
```

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<FDT xmlns="x-schema:FDTUserInformationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <FDTUserInformation userName="ThisUser" userLevel="maintenance" loginTime="2000-05-07T18:39:09"
loginLocation="WorkStation10" sessionDescription="The actor may specify his session here!"/>
</FDT>
```



## 12.4 DTMInformationSchema

Used at: [IDtmInformation::GetInformation\(\)](#)

The XML document provides DTM specific information.

Attribute	Description
major	Major part of the FDT-Specification version on which the implementation of a DTM is based on. For FDT-Specification 1.2.1.0 it must be set to '1'
minor	Minor part of the FDT-Specification version on which the implementation of a DTM is based on. For FDT-Specification 1.2.1.0 it must be set to '2'
release	release part of the FDT-Specification version on which the implementation of a DTM is based on. For FDT-Specification 1.2.1.0 it must be set to '1'. It is highly recommended to use this attribute.
build	build part of the FDT-Specification version on which the implementation of a DTM is based on. For FDT-Specification 1.2.1.0 it must be set to '0'. It is highly recommended to use this attribute.

Tag	Description
DtmDeviceTypes	Collection of device types
DtmInfo	Describes the DTM itself
DtmSchemaPath	File system path with protocol specific schemas and XSLT files (Path including a trailing backslash). This path should not contain any other files. In order to avoid validation errors, developers should be careful with the usage of XML-specific characters (like '&') in this path.
DtmSchemaPaths	Collection of DTM schema paths
FDTVersion	Definition of the element which specifies the version information on which the implementation of a DTM is based on
FDT	Root tag

```
<Schema name="DTMInformationSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="major" dt:type="number"/>
  <AttributeType name="minor" dt:type="number"/>
  <AttributeType name="deviceTypeInfo" dt:type="string"/>
  <AttributeType name="release" dt:type="number"/>
  <AttributeType name="build" dt:type="number"/>
  <!--Definition of Elements-->
  <ElementType name="FDTVersion" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="major" required="yes"/>
    <attribute type="minor" required="yes"/>
    <attribute type="release" required="no"/>
    <attribute type="build" required="no"/>
  </ElementType>
  <ElementType name="DtmDeviceTypes" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
  </ElementType>
</Schema>
```

```

    <element type="fdt:DtmDeviceType" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="DtmSchemaPath" content="empty" model="closed">
    <attribute type="fdt:nodeld" required="no" />
    <attribute type="fdt:busCategory" required="yes" />
    <attribute type="fdt:path" required="yes" />
  </ElementType>
  <ElementType name="DtmSchemaPaths" content="eltOnly" model="closed">
    <attribute type="fdt:nodeld" required="no" />
    <element type="DtmSchemaPath" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="DtmInfo" content="eltOnly" model="closed" order="seq">
    <attribute type="fdt:nodeld" required="no" />
    <element type="FDTVersion" minOccurs="1" maxOccurs="1" />
    <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1" />
    <element type="DtmSchemaPaths" minOccurs="0" maxOccurs="1" />
    <element type="DtmDeviceTypes" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeld" required="no" />
    <element type="DtmInfo" minOccurs="1" maxOccurs="1" />
  </ElementType>
</Schema>

```

### Examples:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMInformationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" >
  <DtmInfo>
    <FDTVersion major="1" minor="2" release="1" build="1" />
    <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05" />
    <DtmSchemaPaths>
      <DtmSchemaPath fdt:busCategory="12345678-ABCD-7890-86E1-00E0987270B9" fdt:path="file://c:/mySchemaPath/"
    />
    </DtmSchemaPaths>
    <DtmDeviceTypes>
      <fdt:DtmDeviceType>
        <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05" />
        <fdt:SupportedLanguages>
          <fdt:LanguageId languageId="1131" />
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
          <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9" busCategoryName="Profibus
DP/V1">
            <fdt:CommunicationTypeEntry communicationType="required" />
          </fdt:BusCategory>
          <fdt:BusCategory busCategory="12345678-ABCD-7890-86E1-00E0987270B9" busCategoryName="Private
Bus">
            <fdt:CommunicationTypeEntry communicationType="supported" />
          </fdt:BusCategory>
          <fdt:BusCategory busCategory="036D1498-387B-11D4-86E1-00E0987270B9" busCategoryName="HART">
            <fdt:CommunicationTypeEntry communicationType="supported" />
            <fdt:PhysicalLayer physicalLayer="036D1590-387B-11D4-86E1-00E0987270B9"
physicalLayerName="IEC61158-2" />
          </fdt:BusCategory>
        </fdt:BusCategories>
        <fdt:DeviceIcon path="file://c:/programs/manufacturer/icons/device.ico" />
        <fdt:DeviceIcon deviceGraphicState="device" path="file://c:/dev.ico" />
        <fdt:DeviceIcon deviceGraphicState="diagnosis" path="file://c:/devdiag.ico" />
        <fdt:DtmDocuments>
          <fdt:DtmDocument label="Online Help" help="Help for Dtm" documentClassification="help"
documentLanguageId="1033">
            <fdt:DocumentFile file="c:\mydtm\helpEN.chm" />
          </fdt:DtmDocument>
          <fdt:DtmDocument label="Online Hilfe" help="Hilfe für DTM" documentClassification="help"
documentLanguageId="1031">
            <fdt:DocumentFile file="c:\mydtm\helpDE.chm" />
          </fdt:DtmDocument>
          <fdt:DtmDocument label="Ordering Information" help="Order our Products"
documentClassification="orderingInformation">

```

```

        <fdt:DocumentUrl url="http://www.manufacturer.com"/>
    </fdt:DtmDocument>

    <fdt:DtmDocument label="Product Database" help="Find a Product" documentClassification="miscellaneous"
documentLanguageId="1033">
        <fdt:DocumentExe file="c:\programs\manufacturer\productFinder.exe" parameters="-L1033" />
    </fdt:DtmDocument>

</fdt:DtmDocuments>
<fdt:DeviceBitmap deviceGraphicState="device" path="file://c:/dev.bmp" />
<fdt:DeviceBitmap deviceGraphicState="diagnosis" path="file://c:/devdiag.bmp" />
<fdt:DataSetFormats>
    <fdt:Current dataSetID="036D1497-387B-11D4-86E1-00E0987270B9" description="The data set in encrypted
according to the client recommendation"/>
    <fdt:Supported dataSetID="036D1497-387B-11D4-86E1-00E0987270B9" levelOfSupport="full"
description="The data set is fully supported"/>
    <fdt:Supported dataSetID="036D1497-387B-11D4-86E1-00E0987270BA" levelOfSupport="full"
description="The default value for fault action is not supported in this version"/>
</fdt:DataSetFormats>
<fdt:DeviceTypeVariants>
    <fdt:DeviceTypeVariant deviceTypeVariant="PM255-EX" deviceTypeVariantInfo="PM255-EX Info"
orderCode="ExampleOrderCode1"/>
    <fdt:DeviceTypeVariant deviceTypeVariant="PM255-TX" deviceTypeVariantInfo="PM255-TX Info"
orderCode="ExampleOrderCode2"/>
    <fdt:DeviceTypeVariant deviceTypeVariant="PM255-ZX" deviceTypeVariantInfo="PM255-ZX Info"
orderCode="ExampleOrderCode3"/>
    <fdt:DeviceTypeVariant deviceTypeVariant="PM255-XX" deviceTypeVariantInfo="PM255-XX Info"
orderCode="ExampleOrderCode4"/>
</fdt:DeviceTypeVariants>
</fdt:DtmDeviceType>
</DtmDeviceTypes>
</DtmInfo>
</FDT>

```

### Example for PROFIBUS :

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DtmInformationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" >
    <DtmInfo>
        <FDTVersion major="1" minor="2" release="1" build="1"/>
        <fdt:VersionInformation name="myName" vendor="myVendor" version="1.0" date="2000-08-05"/>
    </DtmInfo>
    <DtmDeviceTypes>
        <fdt:DtmDeviceType manufacturerId="12" deviceTypeId="3456" deviceTypeInformation="
#Profibus_DP
GSD_Revision          = 2
Vendor_Name           = &myVendor&;
Model_Name            = &myName&;
Revision              = &1.0&;
Ident_Number          = 0x0D80
Protocol_Ident        = 0
Station_Type          = 0
Bitmap_Device = &Src0D80n&;
FMS_supp              = 0
Hardware_Release      = &1.0&;
Software_Release      = &1.0&;
31.25_supp            = 1
45.45_supp            = 1
93.75_supp            = 1
187.5_supp            = 1
MaxTsdr_31.25         = 100
MaxTsdr_45.45         = 250
MaxTsdr_93.75         = 1000
MaxTsdr_187.5         = 1000
Redundancy            = 0
Repeater_Ctrl_Sig     = 0
24V_Pins              = 0
Freeze_Mode_supp      = 0
Sync_Mode_supp        = 0
Auto_Baud_supp        = 0
Set_Slave_Add_supp    = 1
Min_Slave_Intervall   = 250
Modular_Station       = 1
Max_Module            = 6
Max_Input_Len         = 30
Max_Output_Len        = 6

```

```

Max_Data_Len          = 36
Max_Diag_Data_Len     = 14
Slave_Family          = 12
User_Prm_Data_Len     = 0
;Ext_User_Prm_Data_Const(0) = 0x00,0x00,0x00

;Empty module
Module = &EMPTY_MODULE& 0x00
EndModule
"
    deviceTypeInformationPath="file://c:/myDtm/myGsdFile.gsd"
    >
        <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
        <fdt:SupportedLanguages>
            <fdt:LanguageId languageId="1131"/>
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
            <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9" busCategoryName="Profibus
DP/V1">
            <fdt:CommunicationTypeEntry communicationType="required"/>
        </fdt:BusCategory>
        </fdt:BusCategories>
        <fdt:DeviceIcon path="file://c:/programs/manufacturer/icons/device.ico" />
        <fdt:DeviceIcon deviceGraphicState="device" path="file://c:/myDtm/Src0D80n.ico" />
        <fdt:DeviceIcon deviceGraphicState="diagnosis" path="file://c:/myDtm/Src0D80ndiag.ico" />
        <fdt:DtmDocuments>
            <fdt:DtmDocument label="Online Help" help="Help for Dtm" documentClassification="help"
documentLanguageId="1033">
                <fdt:DocumentFile file="c:\mydtm\helpEN.chm"/>
            </fdt:DtmDocument>
            <fdt:DtmDocument label="Online Hilfe" help="Hilfe für DTM" documentClassification="help"
documentLanguageId="1031">
                <fdt:DocumentFile file="c:\mydtm\helpDE.chm"/>
            </fdt:DtmDocument>

            <fdt:DtmDocument label="Ordering Information" help="Order our Products"
documentClassification="orderingInformation">
                <fdt:DocumentUrl url="http://www.manufacturer.com"/>
            </fdt:DtmDocument>

            <fdt:DtmDocument label="Product Database" help="Find a Product" documentClassification="miscellaneous"
documentLanguageId="1033">
                <fdt:DocumentExe file="c:\programs\manufacturer\productFinder.exe" parameters="-L1033" />
            </fdt:DtmDocument>

        </fdt:DtmDocuments>
        <fdt:DeviceBitmap deviceGraphicState="device" path="file://c:/myDtm/Src0D80n.bmp" />
        <fdt:DeviceBitmap deviceGraphicState="diagnosis" path="file://c:/myDtm/Src0D80ndiag.bmp" />
        <fdt:DataSetFormats>
            <fdt:Current dataSetId="036D1497-387B-11D4-86E1-00E0987270B9" description="The data set in encrypted
according to the client recommendation"/>
            <fdt:Supported dataSetId="036D1497-387B-11D4-86E1-00E0987270B9" levelOfSupport="full"
description="The data set is fully supported"/>
            <fdt:Supported dataSetId="036D1497-387B-11D4-86E1-00E0987270BA" levelOfSupport="full"
description="The default value for fault action is not supported in this version"/>
        </fdt:DataSetFormats>
        <fdt:DeviceTypeVariants>
            <fdt:DeviceTypeVariant deviceTypeVariant="PM255-EX" deviceTypeVariantInfo="PM255-EX Info"
orderCode="ExampleOrderCode1"/>
            <fdt:DeviceTypeVariant deviceTypeVariant="PM255-TX" deviceTypeVariantInfo="PM255-TX Info"
orderCode="ExampleOrderCode2"/>
            <fdt:DeviceTypeVariant deviceTypeVariant="PM255-ZX" deviceTypeVariantInfo="PM255-ZX Info"
orderCode="ExampleOrderCode3"/>
            <fdt:DeviceTypeVariant deviceTypeVariant="PM255-XX" deviceTypeVariantInfo="PM255-XX Info"
orderCode="ExampleOrderCode4"/>
        </fdt:DeviceTypeVariants>
        </fdt:DtmDeviceType>
    </DtmDeviceTypes>
</DtmInfo>
</FDT>

```

## 12.5 DTMFunctionCallSchema

Used at: [IDtmActiveXInformation::GetActiveXGuid\(\)](#)  
[IDtmActiveXInformation::GetActiveXProgId\(\)](#)  
[IDtmApplication::StartApplication\(\)](#)  
[IDtmDocumentation::GetDocumentation\(\)](#)  
[IFdtChannelActiveXInformation::GetChannelActiveXGuid\(\)](#)  
[IFdtChannelActiveXInformation::GetChannelActiveXProgId\(\)](#)

The XML document provides information to specify a specific function call.

Attribute	Description
-----------	-------------

Tag	Description
FDTFUNCTIONCALL	Definition of the element which specifies a specific function call
FDTRoot tag	

```
<Schema name="DTMFunctionCallSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:appld="x-schema:FDTApplicationIdSchema.xml" xmlns:func="x-schema:DTMFunctionsSchema.xml" xmlns:ops="x-schema:FDTOperationPhaseSchema.xml">
  <!--Definition of Elements-->
  <ElementType name="FDTFUNCTIONCALL" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="func:functionId" required="yes"/>
    <element type="appld:ApplicationId" minOccurs="0" maxOccurs="1"/>
    <attribute type="ops:operationPhase" required="no"/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="FDTFUNCTIONCALL" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>
```

Example :

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMFunctionCallSchema.xml" xmlns:func="x-schema:DTMFunctionsSchema.xml"
xmlns:appld="x-schema:FDTApplicationIdSchema.xml" xmlns:ops="x-schema:FDTOperationPhaseSchema.xml">
  <FDTFUNCTIONCALL func:functionId="32" ops:operationPhase="runtime">
    <appld:ApplicationId applicationId="fdtObserve"/>
  </FDTFUNCTIONCALL>
</FDT>
```

## 12.6 DTMPParameterSchema

Used at: [IDtmParameter::GetParameters\(\)](#)

[IDtmParameter::SetParameters\(\)](#)

The XML document provides all instance specific information about a device.

Attribute	Description
busAddress	Network participant number of a device according to fieldbus protocols like Profibus
busMasterConfigurationPart	<p>Bus parameters are needed to allow an interaction between DTMs and a master configuration tool. To provide a standard access to this bus specific data, it is stored as a binary stream which contains the device specific bus information according to the fieldbus-specification.</p> <p>Each DTM must at least fill in the device specific parameters and all parameters which can be changed by its application.</p> <p>All other entries may be filled up with substitute values like zeros. The substituted values of the structure will be set by the environment's master configuration tool according to the requirements of the complete bus.</p> <p>Independent of the values filled in, it is very important that the structure generated by the DTM adheres to the definitions of the fieldbus-specification.</p> <p>After all network participants have written their instance data, the master configuration tool can commission the fieldbus. For that purpose it collects the configuration part of each network participant and calculates the bus parameters of the corresponding master device.</p>
configurationData	Additional configuration data as binary stream according to the fieldbus-specification
moduleId	Unique identifier for a module within the name space of the device instance
moduleTypeId	Unique identifier for a module type within the name space of the device type
redundant	Specifies whether a device or module is redundant
slaveAddress	Universal slave address e.g. polling address for HART communication
slot	Unique identifier for the slot of a module within the name space of the device instance

Tag	Description
BusInformation	<p>Device-DTMs with more than one required protocol should include busCategory in BusInformation. .</p> <ul style="list-style-type: none"> <li>If a 1.2.1. DTM is connected to a 1.2 communication channel,</li> </ul>

Tag	Description
	<ul style="list-style-type: none"> <li>An 1.2.1 Frame Application has to consider: <ul style="list-style-type: none"> <li>the first BusInformation element in the DtmParameter document of the DTM is regarded as primary protocol</li> <li>the Frame Application needs to check if the primary protocol of the DTM is supported by the channel</li> </ul> </li> <li>An 1.2 Frame Application: <ul style="list-style-type: none"> <li>The DTM can support only one protocol, because the old schemas supports only one BusInformation element</li> </ul> </li> </ul>
DtmDevice	Description of a device instance Device-DTMs with more than one required protocol should include BusInformation for each Protocol.
ExportedVariables	Collection of DTM variables for common access. This means that the Frame Application or other DTMs are allowed to get access to the data within this section
FDT	Root tag
InternalChannel	An internal channel is the connection point for an internal module within the internal topology
InternalTopology	Description of the internal topology of a modular device build as a none software modular DTM
MasterSlaveBus	Description of bus parameters for communication and configuration
Module	Description of a hardware or software module of a device
Modules	Collection of modules
SlaveAddress	Universal slave address
UserDefinedBus	Description bus parameters for the integration of proprietary bussystems

If a DTM can be used by a redundant aware parent component as DTM for a redundant slave the parameter document must provide additional redundancy information:

- the BusInformation element must contain a BusRedundancy element with the number of supported redundant fieldbus interfaces of the slave

Within address setting by calling SetParameter() the parent component should provide complete redundant address information. The DTM is then able to detect if it's appropriate slave is used as redundant slave. Only in this case specific redundancy handling is to be activated within the DTM, e.g., during communication requests or configuration dialogs. Therefore all fieldbus addresses should be set using SlaveAddress elements by the parent component.

```
<Schema name="DTMParameterSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="busAddress" dt:type="ui4"/>
  <AttributeType name="busMasterConfigurationPart" dt:type="bin.hex"/>
  <AttributeType name="configurationData" dt:type="bin.hex"/>
  <AttributeType name="moduleId" dt:type="ui4"/>
  <AttributeType name="moduleTypeId" dt:type="ui4"/>
  <AttributeType name="redundant" dt:type="boolean"/>
  <AttributeType name="slaveAddress" dt:type="ui4"/>
  <AttributeType name="slot" dt:type="ui4"/>
  <!--Definition of Elements-->
  <ElementType name="SlaveAddress" content="empty" model="closed">
```



```

    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:readAccess" required="no"/>
    <attribute type="fdt:writeAccess" required="no"/>
    <attribute type="slaveAddress" required="yes"/>
  </ElementType>
  <ElementType name="MasterSlaveBus" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:readAccess" required="no"/>
    <attribute type="fdt:writeAccess" required="no"/>
    <attribute type="busAddress" required="yes"/>
    <attribute type="busMasterConfigurationPart" required="yes"/>
  </ElementType>
  <ElementType name="UserDefinedBus" content="mixed" model="open"/>
  <ElementType name="BusInformation" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:busCategory" required="no"/>
    <element type="fdt:BusRedundancy" maxOccurs="1" minOccurs="0"/>
    <group order="one" minOccurs="0" maxOccurs="*">
      <element type="SlaveAddress"/>
      <element type="MasterSlaveBus"/>
      <element type="UserDefinedBus"/>
    </group>
  </ElementType>
  <ElementType name="ExportedVariables" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="fdt:DtmVariables" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="Module" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:readAccess" required="no"/>
    <attribute type="fdt:writeAccess" required="no"/>
    <attribute type="moduleId" required="yes"/>
    <attribute type="moduleTypeId" required="no"/>
    <attribute type="slot" required="no"/>
    <attribute type="redundant" required="no"/>
    <attribute type="configurationData" required="no"/>
    <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1"/>
    <element type="fdt:ChannelReferences" minOccurs="0" maxOccurs="1"/>
    <element type="ExportedVariables" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="InternalChannel" content="eltOnly" model="closed">
    <attribute type="fdt:readAccess" required="no"/>
    <attribute type="fdt:writeAccess" required="no"/>
    <attribute type="fdt:nodeId" required="no"/>
    <element type="Module" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="InternalTopology" content="eltOnly" model="closed">
    <attribute type="fdt:readAccess" required="no"/>
    <attribute type="fdt:writeAccess" required="no"/>
    <attribute type="fdt:nodeId" required="no"/>
    <element type="BusInformation" minOccurs="0" maxOccurs="1"/>
    <element type="InternalChannel" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="DtmDevice" content="eltOnly" model="closed" order="seq">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:readAccess" required="no"/>
    <attribute type="fdt:writeAccess" required="no"/>
    <attribute type="fdt:tag" required="yes"/>
    <attribute type="redundant" required="no"/>
    <element type="fdt:ChannelReferences" minOccurs="0" maxOccurs="1"/>
    <element type="BusInformation" minOccurs="0" maxOccurs="1"/>
    <element type="InternalTopology" minOccurs="0" maxOccurs="1"/>
    <element type="ExportedVariables" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:storageState" required="yes"/>
    <attribute type="fdt:dataSetState" required="yes"/>
    <attribute type="fdt:modifiedInDevice" required="no"/>
    <element type="fdt:DtmDeviceType" minOccurs="1" maxOccurs="1"/>
    <element type="DtmDevice" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>

```



Example for a simple device:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default ">
  <fdt:DtmDeviceType>
    <fdt:VersionInformation name="Transmitter" vendor="Vendor name" version="1.0" date="2000-08-05"/>
    <fdt:SupportedLanguages>
      <fdt:LanguageId languageId="1"/>
    </fdt:SupportedLanguages>
    <fdt:BusCategories>
      <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
      <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
    </fdt:BusCategories>
  </fdt:DtmDeviceType>
  <DtmDevice fdt:tag="00PGH10EC001">
    <fdt:ChannelReferences>
      <fdt:ChannelReference idref="temperature"/>
    </fdt:ChannelReferences>
    <ExportedVariables>
      <fdt:DtmVariables name="Parameter" descriptor="root of parameters">
        <fdt:DtmVariable name="Cell" descriptor="Measuring point data">
          <fdt:Value>
            <fdt:Display string="PT100"/>
          </fdt:Value>
          <fdt:Unit>
            <fdt:EnumeratorVariable>
              <fdt:Variable>
                <fdt:EnumeratorEntry index="1" name="C"/>
              </fdt:Variable>
              <fdt:EnumeratorEntries>
                <fdt:EnumeratorEntry index="1" name="C"/>
                <fdt:EnumeratorEntry index="2" name="K"/>
                <fdt:EnumeratorEntry index="3" name="F"/>
              </fdt:EnumeratorEntries>
            </fdt:EnumeratorVariable>
          </fdt:Unit>
        </fdt:DtmVariable>
        <fdt:DtmVariables name="Curve" descriptor="characteristic curve interpolation points">
          <fdt:DtmVariable name="point1">
            <fdt:Value>
              <fdt:Display string="1.1"/>
            </fdt:Value>
          </fdt:DtmVariable>
          <fdt:DtmVariable name="point2">
            <fdt:Value>
              <fdt:Display string="1.2"/>
            </fdt:Value>
          </fdt:DtmVariable>
          <fdt:DtmVariable name="point3">
            <fdt:Value>
              <fdt:Display string="1.3"/>
            </fdt:Value>
          </fdt:DtmVariable>
        </fdt:DtmVariables>
      </fdt:DtmVariables>
    </ExportedVariables>
  </DtmDevice>
</FDT>
```

Example for a redundant slave:

```
<?xml version="1.0"?>
<FDT fdt:dataSetState="default" fdt:storageState="persistent" xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
```

```

<fdt:DtmDeviceType readAccess="1" writeAccess="0">
  <fdt:VersionInformation date="2000-08-05" name="myname" readAccess="1" vendor="myVendor" version="1.0"
writeAccess="0"/>
  <fdt:SupportedLanguages>
    <fdt:LanguageId languageId="1"/>
  </fdt:SupportedLanguages>
  <fdt:BusCategories>
    <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
  </fdt:BusCategory>
    <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
  </fdt:BusCategory>
  </fdt:BusCategories>
</fdt:DtmDeviceType>
<DtmDevice fdt:tag="myTag">
  <fdt:ChannelReferences>
    <fdt:ChannelReference idref="n1"/>
    <fdt:ChannelReference idref="n25"/>
  </fdt:ChannelReferences>
  <BusInformation>
    <fdt:BusRedundancy busRedundancy="2"/>
    <SlaveAddress fdt:readAccess="1" fdt:writeAccess="0" slaveAddress="123"/>
    <SlaveAddress fdt:readAccess="1" fdt:writeAccess="0" slaveAddress="1234"/>
  </BusInformation>
  <ExportedVariables>
    <fdt:DtmVariables descriptor="root of parameters" name="ParaRoot">
      <fdt:DtmVariable descriptor="display only parameter" name="Simple">
        <fdt:Value readAccess="1" writeAccess="0">
          <fdt:Display string="12.345"/>
        </fdt:Value>
      </fdt:DtmVariable>
    </fdt:DtmVariables>
  </ExportedVariables>
</DtmDevice>
</FDT>

```

Example for the usage of channel mode:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default">
  <fdt:DtmDeviceType readAccess="1" writeAccess="0">
    <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
    <fdt:SupportedLanguages>
      <fdt:LanguageId languageId="1"/>
    </fdt:SupportedLanguages>
    <fdt:BusCategories>
      <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
      <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
    </fdt:BusCategories>
  </fdt:DtmDeviceType>
  <DtmDevice fdt:tag="myTag">
    <fdt:ChannelReferences>
      <fdt:ChannelReference idref="n1"/>
      <fdt:ChannelReference idref="n25"/>
    </fdt:ChannelReferences>
    <InternalTopology>
      <InternalChannel>
        <Module moduleId="1">
          <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
          <fdt:ChannelReferences>
            <fdt:ChannelReference idref="m25.n1">
              <fdt:ChannelInformation>
                <fdt:BusCategories>
                  <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
                </fdt:BusCategories>
                <fdt:ChannelModes>
                  <fdt:ChannelMode channelMode="processValue" />
                </fdt:ChannelModes>
              </fdt:ChannelInformation>
            </fdt:ChannelReference>
            <fdt:ChannelReference idref="m25.n25">
              <fdt:ChannelInformation>
                <fdt:BusCategories>
                  <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
                  <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
                </fdt:BusCategories>
              </fdt:ChannelInformation>
            </fdt:ChannelReference>
          </fdt:ChannelReferences>
        </Module>
      </InternalChannel>
    </InternalTopology>
  </DtmDevice>
</FDT>

```

```

        </fdt:BusCategories>
        <fdt:ChannelModes>
            <fdt:ChannelMode channelMode="communication" />
            <fdt:ChannelMode channelMode="processValue" />
        </fdt:ChannelModes>
    </fdt:ChannelInformation>
    <fdt:ChannelReference>
    </fdt:ChannelReferences>
</Module>
</InternalChannel>
</InternalTopology>
<ExportedVariables>
    <fdt:DtmVariables name="ParaRoot" descriptor="root of parameters">
        <fdt:DtmVariable name="Simple" descriptor="display only parameter">
            <fdt:Value>
                <fdt:Display string="12.345"/>
            </fdt:Value>
        </fdt:DtmVariable>
        <fdt:DtmVariables name="SubStructur" descriptor="sub structure of parameters">
            <fdt:DtmVariable name="Integer" descriptor="editable integer format">
                <fdt:Value>
                    <fdt:Variant dataType="int" byteLength="2">
                        <fdt:NumberData number="4711"/>
                    </fdt:Variant>
                </fdt:Value>
            </fdt:DtmVariable>
        </fdt:DtmVariables>
    </fdt:DtmVariables>
</ExportedVariables>
</DtmDevice>
</FDT>

```

Example for a modular device:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default">
    <fdt:DtmDeviceType>
        <fdt:VersionInformation name="Remotel/O" vendor="Vendor name" version="1.0" date="2000-08-05"
descriptor="Bus coupler"/>
        <fdt:SupportedLanguages>
            <fdt:LanguageId languageId="1"/>
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
            <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
            <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
        </fdt:BusCategories>
    </fdt:DtmDeviceType>
    <DtmDevice fdt:tag="00PGH10EC001">
        <fdt:ChannelReferences>
            <fdt:ChannelReference idref="Binary input"/>
        </fdt:ChannelReferences>
        <InternalTopology>
            <InternalChannel>
                <Module moduleId="1">
                    <fdt:VersionInformation name="AI4" vendor="Vendor name" version="1.0" date="2000-08-05"
descriptor="Analog Input 4 Channels"/>
                    <fdt:ChannelReferences>
                        <fdt:ChannelReference idref="AI4.C1"/>
                        <fdt:ChannelReference idref="AI4.C2"/>
                        <fdt:ChannelReference idref="AI4.C3"/>
                        <fdt:ChannelReference idref="AI4.C4"/>
                    </fdt:ChannelReferences>
                </Module>
            </InternalChannel>
            <InternalChannel>
                <Module moduleId="8">

```

```

        <fdt:VersionInformation name="AO4" vendor="Vendor name" version="1.0" date="2000-08-05"
descriptor="Analog Output 4 Channels"/>
        <fdt:ChannelReferences>
            <fdt:ChannelReference idref="AO4.C1"/>
            <fdt:ChannelReference idref="AO4.C2"/>
            <fdt:ChannelReference idref="AO4.C3"/>
            <fdt:ChannelReference idref="AO4.C4"/>
        </fdt:ChannelReferences>
        <ExportedVariables>
            <fdt:DtmVariables name="Parameter" descriptor="root of parameters">
                <fdt:DtmVariable name="Substitute Value" descriptor="Module type specific parameter">
                    <fdt:Value>
                        <fdt:Display string="false"/>
                    </fdt:Value>
                </fdt:DtmVariable>
            </fdt:DtmVariables>
        </ExportedVariables>
    </Module>
</InternalChannel>
</InternalTopology>
</DtmDevice>
</FDT>

```

Example for a modular device with one DTM for the bus coupler and a DTM for each module type:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default">
    <fdt:DtmDeviceType>
        <fdt:VersionInformation name="Remotel/O" vendor="Vendor name" version="1.0" date="2000-08-05"
descriptor="Bus coupler"/>
        <fdt:SupportedLanguages>
            <fdt:LanguageId languageId="1"/>
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
            <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
            <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
        </fdt:BusCategories>
    </fdt:DtmDeviceType>
    <DtmDevice fdt:tag="00PGH10EC001">
        <fdt:ChannelReferences>
            <fdt:ChannelReference idref="Binary input"/>
        </fdt:ChannelReferences>
    </DtmDevice>
</FDT>

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default">
    <fdt:DtmDeviceType>
        <fdt:VersionInformation name="AI4" vendor="Vendor name" version="1.0" date="2000-08-05" descriptor="Analog
Input 4 Channels"/>
        <fdt:SupportedLanguages>
            <fdt:LanguageId languageId="1"/>
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
            <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
        </fdt:BusCategories>
    </fdt:DtmDeviceType>
    <DtmDevice fdt:tag="00PGH10EC001">
        <fdt:ChannelReferences>
            <fdt:ChannelReference idref="AI4.C1"/>
            <fdt:ChannelReference idref="AI4.C2"/>
        </fdt:ChannelReferences>
    </DtmDevice>
</FDT>

```

```

        <fdt:ChannelReference idref="A14.C3"/>
        <fdt:ChannelReference idref="A14.C4"/>
    </fdt:ChannelReferences>
</DtmDevice>
</FDT>

```

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default">
  <fdt:DtmDeviceType>
    <fdt:VersionInformation name="AO4" vendor="Vendor name" version="1.0" date="2000-08-05" descriptor="Analog
Output 4 Channels"/>
    <fdt:SupportedLanguages>
      <fdt:LanguageId languageId="1"/>
    </fdt:SupportedLanguages>
    <fdt:BusCategories>
      <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
      <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
    </fdt:BusCategories>
  </fdt:DtmDeviceType>
  <DtmDevice fdt:tag="00PGH10EC001">
    <fdt:ChannelReferences>
      <fdt:ChannelReference idref="AO4.C1"/>
      <fdt:ChannelReference idref="AO4.C2"/>
      <fdt:ChannelReference idref="AO4.C3"/>
      <fdt:ChannelReference idref="AO4.C4"/>
    </fdt:ChannelReferences>
    <ExportedVariables>
      <fdt:DtmVariables name="Parameter" descriptor="root of parameters">
        <fdt:DtmVariable name="Substitute Value" descriptor="Module type specific parameter">
          <fdt:Value>
            <fdt:Display string="false"/>
          </fdt:Value>
        </fdt:DtmVariable>
      </fdt:DtmVariables>
    </ExportedVariables>
  </DtmDevice>
</FDT>

```

Example for a compiled datatype structure :

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default">
  <fdt:DtmDeviceType>
    <fdt:VersionInformation name="Transmitter" vendor="Vendor name" version="1.0" date="2000-08-05"/>
    <fdt:SupportedLanguages>
      <fdt:LanguageId languageId="1"/>
    </fdt:SupportedLanguages>
    <fdt:BusCategories>
      <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
    </fdt:BusCategories>
  </fdt:DtmDeviceType>
  <DtmDevice fdt:tag="00PGH10EC001">
    <ExportedVariables>
      <fdt:DtmVariables name="Parameter" descriptor="root of parameters">
        <fdt:DtmVariable name="Float">
          <fdt:Value>
            <fdt:Display string="2.34"/>
          </fdt:Value>
        </fdt:DtmVariable>
        <fdt:DtmVariable name="Integer">
          <fdt:Value>
            <fdt:Variant dataType="int">
              <fdt:NumberData number="2"/>
            </fdt:Variant>
          </fdt:Value>
        </fdt:DtmVariable>
      </fdt:DtmVariables>
    </ExportedVariables>
  </DtmDevice>
</FDT>

```

```

        </fdt:Variant>
      </fdt:Value>
    </fdt:DtmVariable>
    <fdt:DtmVariable name="Integer2">
      <fdt:Value>
        <fdt:Variant dataType="int">
          <fdt:NumberData number="3"/>
        </fdt:Variant>
      </fdt:Value>
    </fdt:DtmVariable>
    <fdt:DtmVariable name="Structured1">
      <fdt:Value>
        <fdt:Variant dataType="structured">
          <fdt:StructuredVariable>
            <fdt:BinaryVariable binData="28FC215403"/>
            <fdt:StructuredElements>
              <fdt:StructuredElement bitLength="4">
                <fdt:DtmVariableReference reference="Integer"/>
              </fdt:StructuredElement>
              <fdt:StructuredElement bitLength="32">
                <fdt:DtmVariableReference reference="Float"/>
              </fdt:StructuredElement>
              <fdt:StructuredElement bitLength="4">
                <fdt:DtmVariableReference reference="Integer2"/>
              </fdt:StructuredElement>
            </fdt:StructuredElements>
          </fdt:StructuredVariable>
        </fdt:Variant>
      </fdt:Value>
    </fdt:DtmVariable>
  </fdt:DtmVariables>
</ExportedVariables>
</DtmDevice>
</FDT>

```

### Example for Device type variant

```

<FDT xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
  fdt:storageState="persistent" fdt:dataSetState="default">
  <fdt:DtmDeviceType readAccess="1" writeAccess="0">
    <fdt:VersionInformation name="Name" vendor="Vendor" version="1.0" date="2000-08-
05"/>
    <fdt:SupportedLanguages>
      <fdt:LanguageId languageId="1311"/>
    </fdt:SupportedLanguages>
    <fdt:BusCategories>
      <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
    </fdt:BusCategories>
    <fdt:DeviceTypeVariant deviceTypeVariant="PM255-EX" deviceTypeVariantInfo="PM255-EX Info"
orderCode="ExampleOrderCode"/>
  </fdt:DtmDeviceType>
  <DtmDevice fdt:tag="myTag">
    ...
  </DtmDevice>
</FDT>

```

## 12.7 DTMDocumentationSchema

Used at: [IDtmDocumentation::GetDocumentation\(\)](#)

The XML document contains the instance specific information according to the request. The context (function id, application id and operation phase) is described via the passed XML document of type DTMFunctionCallSchema.

Attribute	Description
path	Path to an object that has to be embedded within the document like bitmaps or other graphical elements
title	Human readable title for the documentation

Tag	Description
DocumentVariable	Human readable variable description with name, value, range, etc
DocumentVariables	Collection of document variables
DTMSpecificXMLData	Optional additional information which must be described by a private style
DTMStyleForCompleteDocument	Optional style information which has to be provided by a DTM if it returns documents which cannot be described by the FDT standard style
FDT	Root tag
GraphicReference	Reference to an object that has to be embedded within the document like bitmaps or other graphical elements

```
<Schema name="DTMDocumentationSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:dtmi="x-schema:DTMInformationSchema.xml" >
  <!--Definition of Attributes-->
  <AttributeType name="path" dt:type="uri"/>
  <AttributeType name="title" dt:type="string"/>
  <!--Definition of Elements-->
  <ElementType name="GraphicReference" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="title" required="yes"/>
    <attribute type="fdt:descriptor" required="no"/>
    <attribute type="path" required="yes"/>
  </ElementType>
  <ElementType name="DocumentVariable" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:name" required="yes"/>
    <attribute type="fdt:descriptor" required="no"/>
    <attribute type="fdt:display" required="yes"/>
    <element type="fdt:Unit" minOccurs="0" maxOccurs="1"/>
    <element type="fdt:Ranges" minOccurs="0" maxOccurs="1"/>
    <attribute type="fdt:statusFlag" required="no"/>
    <element type="fdt:StatusInformation" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="DocumentVariables" content="mixed" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:name" required="yes"/>
    <attribute type="fdt:descriptor" required="no"/>
    <group order="many">
      <element type="DocumentVariables" minOccurs="0" maxOccurs="*/"/>
      <element type="DocumentVariable" minOccurs="0" maxOccurs="*/"/>
    </group>
  </ElementType>
</Schema>
```



```

        <element type="GraphicReference" minOccurs="0" maxOccurs="*" />
    </group>
</ElementType>
<ElementType name="DTMStyleForCompleteDocument" content="mixed" model="open">
    <!-- can be filled with DTM specific information -->
</ElementType>
<ElementType name="DTMSpecificXMLData" content="mixed" model="open">
    <!-- can be filled with DTM specific information -->
</ElementType>
<ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodetd" required="no" />
    <attribute type="title" required="yes" />
    <attribute type="fdt:classificationId" required="no" />
    <attribute type="fdt:manufacturerId" required="no" />
    <attribute type="fdt:deviceTypeId" required="no" />
    <attribute type="dtmi:deviceTypeInfo" required="no" />
    <attribute type="fdt:descriptor" required="no" />
    <attribute type="fdt:date" required="no" />
    <attribute type="fdt:systemGuiLabel" required="no" />
    <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1" />
    <element type="fdt:ClassificationIds" minOccurs="0" maxOccurs="1" />
    <element type="DocumentVariables" minOccurs="1" maxOccurs="1" />
    <group order="seq" minOccurs="0" maxOccurs="1">
        <element type="DTMStyleForCompleteDocument" minOccurs="1" maxOccurs="1" />
        <element type="DTMSpecificXMLData" minOccurs="1" maxOccurs="1" />
    </group>
</ElementType>
</Schema>

```

#### Example:

```

<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="DTMDocumentationStyle.xsl"?>
<FDT xmlns="x-schema:DTMDocumentationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:dtmi="x-
schema:DTMInformationSchema.xml" title="Example of FDT-Documentation" fdt:classificationId="dtmSpecific"
fdt:manufacturerId="0" fdt:deviceTypeId="0" dtmi:deviceTypeInfo="Special Information concerning the type"
fdt:descriptor="Description of: Example of FDT-Documentation" fdt:date="2000-02-06">
    <fdt:VersionInformation name="Name concerning this information" vendor="Name of Vendor" version="Current Version"
date="2000-02-06" descriptor="Additional Description"/>
    <DocumentVariables fdt:name="myName">
        <DocumentVariable fdt:name="1st Value" fdt:descriptor="Description of 1st Value" fdt:display="11.111"
fdt:statusFlag="ok">
            <fdt:Unit>
                <fdt:EnumeratorVariable>
                    <fdt:Variable>
                        <fdt:EnumeratorEntry index="1" name="°C" descriptor="Additional Description" />
                    </fdt:Variable>
                </fdt:EnumeratorVariable>
            </fdt:Unit>
            <fdt:Ranges>
                <fdt:Range>
                    <fdt:LowerRange>
                        <fdt:NumberData number="11"/>
                    </fdt:LowerRange>
                    <fdt:UpperRange>
                        <fdt:NumberData number="22"/>
                    </fdt:UpperRange>
                </fdt:Range>
            </fdt:Ranges>
        </DocumentVariable>
        <DocumentVariable fdt:name="2nd Value" fdt:descriptor="Description of 2nd Value" fdt:display="3.1415"
fdt:statusFlag="warning"/>
        <DocumentVariable fdt:name="3rd" fdt:descriptor="Description of 3rd Value" fdt:display="1.717"
fdt:statusFlag="error"/>
        <DocumentVariable fdt:name="4th" fdt:descriptor="Description of 4th Value" fdt:display="42" fdt:statusFlag="ok"/>
        <GraphicReference title="GraphicReferenceTitle" fdt:descriptor="Descriptor GraphicReference" path="device.gif"/>
    </DocumentVariables>
</FDT>

```



```
</DocumentVariables>  
<DTMStyleForCompleteDocument>...</DTMStyleForCompleteDocument>  
<DTMSpecificXMLData>...</DTMSpecificXMLData>  
</FDT>
```

## 12.8 DTMProtocolsSchema

Used at: [IFdtCommunication::GetSupportedProtocols\(\)](#)

[IFdtChannel::GetChannelParameters\(\)](#)

[IFdtChannel::SetChannelParameters\(\)](#)

The XML document provides information about the supported fieldbus protocols of a specific device

Tag	Description
FDT	Root tag

```
<Schema name="DTMProtocolsSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml" >
  <!--Definition of Elements-->
  <ElementType name="FDT" content="eltOnly" model="closed">
    <element type="fdt:BusCategories" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>
```

Example:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMProtocolsSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <fdt:BusCategories>
    <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
    <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
  </fdt:BusCategories>
</FDT>
```

## 12.9 DTMSystemTagListSchema

Used at: [IFdtTopology::GetParentNodes\(\)](#)

[IFdtTopology::GetChildNodes\(\)](#)

The XML document contains a list of system tags. According to the usage, instances could contain a list of parent or child system tags.

Attribute	Description
systemTag	Refer to the definition of <a href="#">systemTag</a>

Tag	Description
DtmChildList	List of child nodes defined by system tags
DtmParentList	List of parent nodes defined by system tags
DtmPrimaryParentSystemTag	Primary parent of a DTM
DtmSystemTag	System tag
DtmSystemTagList	List of system tags
FDT	Root tag

```
<Schema name="DTMSystemTagListSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="systemTag" dt:type="string"/>
  <!--Definition of Elements-->
  <ElementType name="DtmSystemTag" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="systemTag" required="yes"/>
  </ElementType>
  <ElementType name="DtmPrimaryParentSystemTag" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="systemTag" required="yes"/>
  </ElementType>
  <ElementType name="DtmSystemTagList" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="DtmSystemTag" minOccurs="0" maxOccurs="*/>
  </ElementType>
  <ElementType name="DtmParentList" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="DtmPrimaryParentSystemTag" minOccurs="0" maxOccurs="1"/>
    <element type="DtmSystemTagList" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="DtmChildList" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="DtmSystemTagList" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
  </ElementType>
</Schema>
```

```
<group order="one" minOccurs="1" maxOccurs="1">
  <element type="DtmParentList"/>
  <element type="DtmChildList"/>
</group>
</ElementType>
</Schema>
```

Example IFdtTopology::GetParentNodes():

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMSystemTagListSchema.xml">
  <DtmParentList>
    <DtmPrimaryParentSystemTag systemTag = "DTM-1"/>
    <DtmSystemTagList>
      <DtmSystemTag systemTag = "DTM-2"/>
      <DtmSystemTag systemTag = "DTM-3"/>
      <DtmSystemTag systemTag = "DTM-4"/>
    </DtmSystemTagList>
  </DtmParentList>
</FDT>
```

Example IFdtTopology::GetChildNodes ():

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMSystemTagListSchema.xml">
  <DtmChildList>
    <DtmSystemTagList>
      <DtmSystemTag systemTag = "DTM-2"/>
      <DtmSystemTag systemTag = "DTM-3"/>
      <DtmSystemTag systemTag = "DTM-4"/>
    </DtmSystemTagList>
  </DtmChildList>
</FDT>
```

## 12.10 DTMAuditTrailSchema

Used at: [IDtmAuditTrailEvents::OnAuditTrailEvent\(\)](#)

The XML document contains the information about a changed variable, an executed function, or status event that has to be recorded by the Frame Application.

Attribute	Description
path	Path to an object that has to be embedded within the document like bitmaps or other graphical elements
title	Human readable title for the documentation

Tag	Description
AuditTrailDeviceStatusEvent	Description of the status of an device
AuditTrailEvent	Notification about a changed variable, executed function, or status event
AuditTrailFunctionEvent	Description about an executed function
AuditTrailVariable	Description of a changed variable
AuditTrailVariableEvent	Notification about a change variable
ChangedFrom	Last value of an audit trail variable
ChangedTo	New value of an audit trail variable
FDT	Root tag

```
<Schema name="DTMAuditTrailSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="path" dt:type="uri"/>
  <AttributeType name="title" dt:type="string"/>
  <!--Definition of Elements-->
  <ElementType name="AuditTrailDeviceStatusEvent" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:statusFlag" required="yes"/>
    <element type="fdt:StatusInformation" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="AuditTrailFunctionEvent" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:display" required="yes"/>
  </ElementType>
  <ElementType name="AuditTrailVariable" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:display" required="yes"/>
    <element type="fdt:Unit" minOccurs="0" maxOccurs="1"/>
    <element type="fdt:Ranges" minOccurs="0" maxOccurs="1"/>
    <attribute type="fdt:statusFlag" required="no"/>
    <element type="fdt:StatusInformation" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="ChangedFrom" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
  </ElementType>
```

```

    <element type="AuditTrailVariable" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="ChangedTo" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="AuditTrailVariable" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="AuditTrailVariableEvent" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:name" required="yes"/>
    <attribute type="fdt:descriptor" required="no"/>
    <element type="ChangedFrom" minOccurs="1" maxOccurs="1"/>
    <element type="ChangedTo" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="AuditTrailEvent" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:descriptor" required="no"/>
    <group order="one" minOccurs="1" maxOccurs="1">
      <element type="AuditTrailFunctionEvent"/>
      <element type="AuditTrailVariableEvent"/>
      <element type="AuditTrailDeviceStatusEvent"/>
    </group>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="AuditTrailEvent" minOccurs="0" maxOccurs="*/>
  </ElementType>
</Schema>

```

#### Example:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMAuditTrailSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <AuditTrailEvent fdt:descriptor="myDescription">
    <AuditTrailFunctionEvent fdt:display="myFunctionEvent"/>
  </AuditTrailEvent>
</FDT>

```

## 12.11 DTMDeviceStatusSchema

Used at: [IDtmOnlineDiagnosis::GetDeviceStatus\(\)](#)

The XML document contains the description of the current status of a device.

Attribute	Description
deviceInitiatedCommunication	device is currently using device initiated communication

Tag	Description
DtmDeviceStatus	Description of the current status of a device
FDT	Root tag

```
<Schema name="DTMDeviceStatusSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="deviceInitiatedCommunication" dt:type="boolean"/>
  <ElementType name="DtmDeviceStatus" content="mixed" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:statusFlag" required="yes"/>
    <attribute type="deviceInitiatedCommunication" required="no"/>
    <element type="fdt:StatusInformation" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="DtmDeviceStatus" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>
```

Example:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMDeviceStatusSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <DtmDeviceStatus fdt:statusFlag="error" deviceInitiatedCommunication="1">
    <fdt:StatusInformation>
      <fdt:EnumeratorEntry index="2" name="ErrorName1" descriptor="Thats error number 1"/>
      <fdt:EnumeratorEntry index="0815" name="ErrorName0815" descriptor="Thats error number 0815"/>
    </fdt:StatusInformation>
  </DtmDeviceStatus>
</FDT>
```

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMDeviceStatusSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <DtmDeviceStatus fdt:statusFlag="ok"/>
</FDT>
```

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMDeviceStatusSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <DtmDeviceStatus fdt:statusFlag="error">
    <fdt:StatusInformation>
```

```
<fdt:EnumeratorEntry index="2" name="ErrorName1" descriptor="Thats error number 1"/>
<fdt:EnumeratorEntry index="0815" name="ErrorName0815" descriptor="Thats error number 0815"/>
</fdt:StatusInformation>
</DtmDeviceStatus>
</FDT>
```



## 12.12 *DTMFunctionsSchema*

Used at: [IDtm::GetFunctions\(\)](#)

The XML document describes the functionalities which offers a DTM a Frame Application in a structured way. These functions could be standard (defined by application id) or additional (defined by DTM specific function id) functions. Furthermore the document contains information how the Frame Application can get access to these functions.

Attribute	Description
checked	Status of a menu entry
enabled	Status of a menu entry
functionId	Specifies an unique identifier for a function call within the namespace of a DTM
hasGUI	Specifies whether the DTM supports a user interface for a extended function
printable	Specifies whether the DTM supports a printable document for a function (access via <a href="#">IDtmDocumentation::GetDocumentation()</a> ).
help	Human readable help string for a function
hidden	Status of a menu entry
label	Human readable label of a function
path	Path to an object that has to be embedded for a function like bitmaps or other graphical elements
mime-type	Mime-type of a document provided by a DTM.
programName	Specifies the name of a document viewer program.
resizable	Specifies whether the DTM supports a resizable user interface
separator	Special menu entry
toggle	Status of a menu entry whether it is active or not
moduleId	Unique identifier for a module within the name space of the device instance. The same ids as defined in the DTMPParameter document must be used.
defaultFunctionId	Optional attribute, contains a function ID which can be used to open a default ActiveXControl or to raise a default function.

Tag	Description
FDT	Root tag
Function	Definition of a single function entry, which is not a standard function (concerning the applicationIds)
Document	Definition of a document, which is provided by a DTM and displayed by the Frame Application, if an user selects the entry.
Functions	Definition of a group of standard function entries. This entry has not an own label. The label must be supplied by the Frame Application. It can not contain standard functions
Icon	Icon for a function
StandardFunction	Definition of a single function entry, which is a standard function (concerning the applicationIds). This entry has not an own label. The label must be supplied by the Frame Application
Status	Description of a menu
MimeType	Mime-type of a document provided by a DTM. The mime-type is used by the Frame Application to determine an appropriated viewer for a document.
OpenWith	Name of the program, which should be used to open a docum" ooleanied by a DT".

```

<Schema name="DTMFunctionsSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:appld="x-schema:FDTApplicationIdSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="functionId" dt:type="i4"/>
  <AttributeType name="checked" dt:type="boolean"/>
  <AttributeType name="enabled" dt:type="boolean"/>
  <AttributeType name="hasGUI" dt:type="boolean" default="0"/>
  <AttributeType name="printable" dt:type="boolean" default="0"/>
  <AttributeType name="help" dt:type="string"/>
  <AttributeType name="hidden" dt:type="boolean"/>
  <AttributeType name="label" dt:type="string"/>
  <AttributeType name="path" dt:type="uri"/>
  <AttributeType name="mime-type" dt:type="string"/>
  <AttributeType name="programName" dt:type="string"/>
  <AttributeType name="resizable" dt:type="boolean" default="0"/>
  <AttributeType name="separator" dt:type="boolean"/>
  <AttributeType name="toggle" dt:type="boolean"/>
  <AttributeType name="printableStandardFunction" dt:type="boolean" default="1"/>
  <AttributeType name="resizableStandardFunction" dt:type="boolean" default="1"/>
  <AttributeType name="moduleId" dt:type="ui4"/>
  <AttributeType name="defaultFunctionId" dt:type="i4"/>

  <!--Definition of Elements-->
  <ElementType name="MimeType" content="empty" model="closed">
    <attribute type="mime-type" required="no"/>
  </ElementType>
  <ElementType name="OpenWith" content="empty" model="closed">
    <attribute type="programName" required="no"/>
  </ElementType>
  <ElementType name="Status" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="toggle" required="yes"/>
    <attribute type="checked" required="yes"/>
    <attribute type="enabled" required="yes"/>
    <attribute type="hidden" required="yes"/>
    <attribute type="separator" required="yes"/>
  </ElementType>
  <ElementType name="Icon" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="path" required="yes"/>
  </ElementType>
  <!--Definition of a single function entry, which is not a standard function (concerning the applicationIds)-->
  <!--This entry has its own label-->
  <ElementType name="Function" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="label" required="yes"/>
    <attribute type="fdt:name" required="yes"/>
    <attribute type="help" required="yes"/>
    <attribute type="hasGUI" required="no"/>
    <attribute type="printable" required="no"/>
    <attribute type="resizable" required="no"/>
    <attribute type="functionId" required="yes"/>
    <element type="Icon" minOccurs="0" maxOccurs="1"/>
    <element type="Status" minOccurs="0" maxOccurs="1"/>
    <element type="appld:FDTApplicationIds" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <!--Definition of a document entry, which specifies a path to a DTM provided document-->
  <!--This entry has its own label-->
  <ElementType name="Document" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="label" required="yes"/>
    <attribute type="help" required="yes"/>
    <attribute type="path" required="yes"/>
    <group order="one" minOccurs="1" maxOccurs="1">
      <element type="MimeType" minOccurs="1" maxOccurs="1"/>
      <element type="OpenWith" minOccurs="1" maxOccurs="1"/>
    </group>
    <element type="Icon" minOccurs="0" maxOccurs="1"/>
  </ElementType>

```

```

<!--Definition of a single function entry, which is a standard function (concerning the applicationIds)-->
<!--This entry has not an own label. The label must be supplied by the frame application-->
<ElementType name="StandardFunction" content="eltOnly" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <attribute type="fdt:name" required="yes"/>
  <attribute type="help" required="yes"/>
  <attribute type="printableStandardFunction" required="no"/>
  <attribute type="resizableStandardFunction" required="no"/>
  <attribute type="functionId" required="yes"/>
  <element type="Icon" minOccurs="0" maxOccurs="1"/>
  <element type="Status" minOccurs="0" maxOccurs="1"/>
  <element type="appld:ApplicationId" minOccurs="1" maxOccurs="1"/>
</ElementType>
<!--Definition of a group of standard function entries-->
<!--This entry has not an own label. The label must be supplied by the frame application. It can not contain standard functions-->
<ElementType name="StandardFunctions" content="eltOnly" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <element type="appld:ApplicationId" minOccurs="1" maxOccurs="1"/>
  <attribute type="fdt:name" required="yes"/>
  <attribute type="help" required="yes"/>
  <attribute type="moduleld" required="no"/>
  <element type="Status" minOccurs="0" maxOccurs="1"/>
  <group order="many" minOccurs="0" maxOccurs="*">
    <element type="Function" minOccurs="0" maxOccurs="*" />
    <element type="Functions" minOccurs="0" maxOccurs="*" />
  </group>
</ElementType>
<!--Definition of a group of function entries-->
<!--This entry has its own label and could contain standard, non standard functions and documents-->
<ElementType name="Functions" content="eltOnly" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <attribute type="label" required="yes"/>
  <attribute type="fdt:name" required="yes"/>
  <attribute type="help" required="yes"/>
  <attribute type="moduleld" required="no"/>
  <element type="Status" minOccurs="0" maxOccurs="1"/>
  <group order="many" minOccurs="0" maxOccurs="*">
    <group order="one" minOccurs="0" maxOccurs="*">
      <element type="Function" minOccurs="0" maxOccurs="*" />
      <element type="Document" minOccurs="0" maxOccurs="*" />
      <element type="StandardFunction" minOccurs="0" maxOccurs="*" />
    </group>
    <group order="one" minOccurs="0" maxOccurs="*">
      <element type="Functions" minOccurs="0" maxOccurs="*" />
      <element type="StandardFunctions" minOccurs="0" maxOccurs="*" />
    </group>
  </group>
</ElementType>
<ElementType name="FDT" content="eltOnly" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <attribute type="defaultFunctionId" required="no"/>
  <element type="Functions" minOccurs="1" maxOccurs="1"/>
</ElementType>
</Schema>

```

#### Example for simple structure:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMFunctionsSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:appld="x-
schema:FDTApplicationIdSchema.xml" defaultFunctionId="1">
  <Functions label="Standard Functions" fdt:name="" help="">
    <StandardFunction fdt:name="Observe" help="Help text for Observe" functionId="1">
      <Status toggle="1" checked="0" enabled="1" hidden="0" separator="0"/>
      <appld:ApplicationId applicationId="fdtObserve"/>
    </StandardFunction>
  </Functions>

```

```

<StandardFunction fdt:name="Configuration" help="Help text for Configuration" functionId="2">
  <Status toggle="1" checked="0" enabled="1" hidden="0" separator="0"/>
  <appld:ApplicationId applicationId="fdtConfiguration"/>
</StandardFunction>
<StandardFunction fdt:name="Diagnosis" help="Help text for Diagnosis" functionId="3">
  <Status toggle="1" checked="0" enabled="1" hidden="0" separator="0"/>
  <appld:ApplicationId applicationId="fdtDiagnosis"/>
</StandardFunction>
</Functions>
</FDT>

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMFunctionsSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:appld="x-
schema:FDTApplicationIdSchema.xml">
  <Functions label="External Functions" fdt:name="ExFuns1" help="Help text for External Functions 1">
    <Function label="External Function 1" fdt:name="ExFun1" help="Help text for External Function 1" functionId="32">
      <Status toggle="1" checked="0" enabled="1" hidden="0" separator="0"/>
      <appld:FDTApplicationIds>
        <appld:ApplicationId applicationId="fdtDiagnosis"/>
        <appld:ApplicationId applicationId="fdtForce"/>
        <appld:ApplicationId applicationId="fdtObserve"/>
      </appld:FDTApplicationIds>
    </Function>
  </Functions>
</FDT>

```

#### Example for complex structure:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMFunctionsSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:appld="x-
schema:FDTApplicationIdSchema.xml" defaultFunctionId="42">
  <Functions label="Functions" fdt:name="" help="">
    <!--Definition of a group of function entries below the standard applicationID Configuration-->
    <!--This entry has its own label and could contain only non standard functions-->
    <!--The menu should appear as-->
    <!--Configuration (label supplied by the frame application)-->
    <!-- Inputs (label supplied by DTM)-->
    <!-- Outputs (label supplied by DTM)-->
    <!--Extras (label supplied by the DTM)-->
    <!-- Extra entry 1 (label supplied by DTM)-->
    <!-- Extra entry 2 (label supplied by DTM)-->
    <!-- Modules (optional submenu supplied by the frame application)-->
    <!-- AI (module name supplied by the DTM through GetParameters)-->
    <!-- AI entry 1 (label supplied by DTM)-->
    <!-- AI entry 2 (label supplied by DTM)-->
    <!-- AO (module name supplied by the DTM through GetParameters)-->
    <!-- AO entry 1 (label supplied by DTM)-->
    <!-- AO entry 2 (label supplied by DTM)-->
    <StandardFunctions fdt:name="Configuration" help="Help text for Configuration">
      <appld:ApplicationId applicationId="fdtConfiguration"/>
      <Status toggle="1" checked="0" enabled="1" hidden="0" separator="0"/>
      <Function label="Inputs" fdt:name="Inputs" help="Help text for Configuration Inputs" functionId="42">
        <Status toggle="1" checked="0" enabled="1" hidden="0" separator="0"/>
        <appld:FDTApplicationIds>
          <appld:ApplicationId applicationId="fdtConfiguration"/>
        </appld:FDTApplicationIds>
      </Function>
      <Function label="Outputs" fdt:name="Outputs" help="Help text for Configuration Outputs" functionId="52">
        <Status toggle="1" checked="0" enabled="1" hidden="0" separator="0"/>
        <appld:FDTApplicationIds>
          <appld:ApplicationId applicationId="fdtConfiguration"/>
        </appld:FDTApplicationIds>
      </Function>
    </StandardFunctions>
    <Functions label="Extras" fdt:name="My extra functions" help="Help text for my extra functions">
      <Function label="Extra entry 1" fdt:name="Extra 1" help="Help for extra 1" hasGUI="1" functionId="65"/>
      <Function label="Extra entry 2" fdt:name="Extra 2" help="Help for extra 2" functionId="75"/>
      <Document label="Help for Extras" help="Help file for Extras" path="file://C:/MyDTMPPath/Extra.hlp">

```

```

        <OpenWith programName="WinHelp.exe"/>
    </Document>
</Functions>
<Functions label="AI functions" fdt:name="My AI functions" help="Help text for AI functions" moduleId="1">
    <Function label="AI entry 1" fdt:name="AI 1" help="Help for AI entry 1" hasGUI="1" functionId="1001"/>
    <Function label="AI entry 2" fdt:name="AI 2" help="Help for AI entry 2" functionId="1002"/>
    <Document label="Help for AI" help="Help file for AI" path="http://www.your_compoany.com/support/AI.hlp">
        <OpenWith programName="WinHelp.exe"/>
    </Document>
</Functions>
<Functions label="AO functions" fdt:name="My AO functions" help="Help text for AO functions" moduleId="2">
    <Function label="AO entry 1" fdt:name="AO 1" help="Help for AO entry 1" hasGUI="1" functionId="2001"/>
    <Function label="AO entry 2" fdt:name="AO 2" help="Help for AO entry 2" functionId="2002"/>
    <Document label="Help for AO" help="Help file for AO" path="ftp://ftp.your_compoany.com/support/AO.hlp">
        <OpenWith programName="WinHelp.exe"/>
    </Document>
</Functions>
</Functions>
</FDT>

```

## 12.13 DTMChannelFunctionsSchema

Used at: [IFdtChannelActiveXInformation::GetChannelFunctions\(\)](#)

The XML document describes the functionalities which offers a DTM channel object a Frame Application in a structured way. These functions could be standard (defined by application id) or additional (defined by DTM specific function id) functions. Furthermore the document contains information how the Frame Application can get access to these functions. Only user interface oriented functions are supported.

Attribute	Description
defaultFunctionId	Optional attribute, contains a function ID which can be used to open a default ActiveXControl or to raise a default function.
functionId	Specifies an unique identifier for a function call within the namespace of a DTM
printable	Specifies whether the DTM supports a printable document for a function (access via <a href="#">IDtmDocumentation::GetDocumentation()</a> ).
help	Human readable help string for a function
hidden	Status of a menu entry
label	Human readable label of a function
path	Path to an object that has to be embedded for a function like bitmaps or other graphical elements
mime-type	Mime-type of a document provided by a DTM.
programName	Specifies the name of a document viewer program.
resizable	Specifies whether the DTM supports a resizable user interface
separator	Special menu entry

Tag	Description
FDT	Root tag
Function	Definition of a single function entry which is not a standard function (concerning the applicationIds)
Functions	Definition of a group of standard function entries. This entry has not an

Tag	Description
	own label. The label must be supplied by the Frame Application. It can not contain standard functions
Icon	Icon for a function
StandardFunction	Definition of a single function entry which is a standard function (concerning the applicationIds). This entry has not an own label. The label must be supplied by the Frame Application
Status	Description of a menu

```

<Schema name="DTMChannelFunctionsSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:appld="x-schema:FDTApplicationIdSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="functionId" dt:type="i4"/>
  <AttributeType name="defaultFunctionId" dt:type="i4"/>
  <AttributeType name="enabled" dt:type="boolean"/>
  <AttributeType name="printable" dt:type="boolean" default="0"/>
  <AttributeType name="help" dt:type="string"/>
  <AttributeType name="hidden" dt:type="boolean"/>
  <AttributeType name="label" dt:type="string"/>
  <AttributeType name="path" dt:type="uri"/>
  <AttributeType name="mime-type" dt:type="string"/>
  <AttributeType name="programName" dt:type="string"/>
  <AttributeType name="resizable" dt:type="boolean" default="0"/>
  <AttributeType name="separator" dt:type="boolean"/>
  <!--Definition of Elements-->
  <ElementType name="MimeType" content="empty" model="closed">
    <attribute type="mime-type" required="no"/>
  </ElementType>
  <ElementType name="OpenWith" content="empty" model="closed">
    <attribute type="programName" required="no"/>
  </ElementType>
  <ElementType name="Status" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="enabled" required="yes"/>
    <attribute type="hidden" required="yes"/>
    <attribute type="separator" required="yes"/>
  </ElementType>
  <ElementType name="Icon" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="path" required="yes"/>
  </ElementType>
  <!--Definition of a single function entry, which is not a standard function (concerning the applicationIds)-->
  <!--This entry has its own label-->
  <ElementType name="Function" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="label" required="yes"/>
    <attribute type="fdt:name" required="yes"/>
    <attribute type="help" required="yes"/>
    <attribute type="printable" required="no"/>
    <attribute type="resizable" required="no"/>
    <attribute type="functionId" required="yes"/>
    <element type="Icon" minOccurs="0" maxOccurs="1"/>
    <element type="Status" minOccurs="0" maxOccurs="1"/>
    <element type="appld:FDTApplicationIds" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <!--Definition of a document entry, which specifies a path to a DTM provided document-->
  <!--This entry has its own label-->
  <ElementType name="Document" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="label" required="yes"/>
    <attribute type="help" required="yes"/>
    <attribute type="path" required="yes"/>
    <group order="one" minOccurs="1" maxOccurs="1">
      <element type="MimeType" minOccurs="1" maxOccurs="1"/>
      <element type="OpenWith" minOccurs="1" maxOccurs="1"/>
    </group>
    <element type="Icon" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <!--Definition of a single function entry, which is a standard function (concerning the applicationIds)-->

```

```

<!--This entry has not an own label. The label must be supplied by the frame application-->
<ElementType name="StandardFunction" content="eltOnly" model="closed">
  <attribute type="fdt:nodelId" required="no"/>
  <attribute type="fdt:name" required="yes"/>
  <attribute type="help" required="yes"/>
  <attribute type="functionId" required="yes"/>
  <element type="Icon" minOccurs="0" maxOccurs="1"/>
  <element type="Status" minOccurs="0" maxOccurs="1"/>
  <element type="appld:ApplicationId" minOccurs="1" maxOccurs="1"/>
</ElementType>
<!--Definition of a group of standard function entries-->
<!--This entry has not an own label. The label must be supplied by the frame application. It can not contain standard functions-->
<ElementType name="StandardFunctions" content="eltOnly" model="closed">
  <attribute type="fdt:nodelId" required="no"/>
  <element type="appld:ApplicationId" minOccurs="1" maxOccurs="1"/>
  <attribute type="fdt:name" required="yes"/>
  <attribute type="help" required="yes"/>
  <element type="Status" minOccurs="0" maxOccurs="1"/>
  <group order="many" minOccurs="0" maxOccurs="*">
    <element type="Function" minOccurs="0" maxOccurs="*" />
    <element type="Functions" minOccurs="0" maxOccurs="*" />
  </group>
</ElementType>
<!--Definition of a group of function entries-->
<!--This entry has its own label and could contain standard, non standard functions and documents-->
<ElementType name="Functions" content="eltOnly" model="closed">
  <attribute type="fdt:nodelId" required="no"/>
  <attribute type="label" required="yes"/>
  <attribute type="fdt:name" required="yes"/>
  <attribute type="help" required="yes"/>
  <element type="Status" minOccurs="0" maxOccurs="1"/>
  <group order="many" minOccurs="0" maxOccurs="*">
    <group order="one" minOccurs="0" maxOccurs="*">
      <element type="Function" minOccurs="0" maxOccurs="*" />
      <element type="Document" minOccurs="0" maxOccurs="*" />
      <element type="StandardFunction" minOccurs="0" maxOccurs="*" />
    </group>
    <group order="one" minOccurs="0" maxOccurs="*">
      <element type="Functions" minOccurs="0" maxOccurs="*" />
      <element type="StandardFunctions" minOccurs="0" maxOccurs="*" />
    </group>
  </group>
</ElementType>
<ElementType name="FDT" content="eltOnly" model="closed">
  <attribute type="fdt:nodelId" required="no"/>
  <attribute type="defaultFunctionId" required="no"/>
  <element type="Functions" minOccurs="1" maxOccurs="1"/>
</ElementType>
</Schema>

```

### Example:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMChannelFunctionsSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
  xmlns:appld="x-schema:FDTApplicationIdSchema.xml">
  <Functions label="Standard Functions" fdt:name="" help="">
    <StandardFunction fdt:name="Configuration" help="Help text for Configuration" functionId="2">
      <Status enabled="1" hidden="0" separator="0"/>
      <appld:ApplicationId applicationId="fdtConfiguration"/>
    </StandardFunction>
    <StandardFunction fdt:name="Diagnosis" help="Help text for Diagnosis" functionId="3">
      <Status enabled="1" hidden="0" separator="0"/>
      <appld:ApplicationId applicationId="fdtDiagnosis"/>
    </StandardFunction>
  </Functions>
</FDT>

```



## 12.14 DTMOnlineCompareSchema

Used at: [IDtmOnlineDiagnosis::Compare\(\)](#)

The XLM document describes whether the data stored in database and the data uploaded from the device could be compared. If the data could be compared the result shows whether the data are equal or not. Otherwise the document contains the communication error.

Attribute	Description
statusFlag	Describes whether the data are equal or not

Tag	Description
CompareResult	Contains the compare result or a communication error
FDT	Root tag

```
<Schema name="DTMOnlineCompareSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="statusFlag" dt:type="enumeration" dt:values="equal notEqual"/>
  <!--Definition of Elements-->
  <ElementType name="CompareResult" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="statusFlag" required="yes"/>
    <element type="fdt:StatusInformation" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <group order="one">
      <element type="CompareResult"/>
      <element type="fdt:CommunicationError"/>
    </group>
  </ElementType>
</Schema>
```

Example:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMOnlineCompareSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <CompareResult statusFlag="equal"/>
</FDT>

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMOnlineCompareSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <CompareResult statusFlag="notEqual">
    <fdt:StatusInformation>
      <fdt:EnumeratorEntry index="2" name="ErrorName1" descriptor="Thats error number 1"/>
      <fdt:EnumeratorEntry index="0815" name="ErrorName0815" descriptor="Thats error number 0815"/>
    </fdt:StatusInformation>
  </CompareResult>
</FDT>
```



```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMOnlineCompareSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <fdt:CommunicationError communicationError="noConnection" tag="00PGH10EB004"/>
</FDT>

```

## 12.15 FDTFailSafeDataSchema

Used at: `IFdtFunctionBlockData::GetFBInstanceData()`

The XLM document describes the failsafe data of the associated function block.

Attribute	Description
fsBulkData	Failsafe data
fsBusMasterTag	Device tag of the according master
fsDeviceFBTag	Device tag of the according function block
fsDeviceTag	Device tag of the failsafe device

Tag	Description
FDTFailSafeData	Contains the failsafe data of a function block
FDT	Root tag

```

<Schema name="FDTFailSafeDataSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="fsDeviceTag" dt:type="string"/>
  <AttributeType name="fsBusMasterTag" dt:type="string"/>
  <AttributeType name="fsDeviceFBTag" dt:type="string"/>
  <AttributeType name="fsBulkData" dt:type="bin.base64"/>
  <!--Definition of Elements-->
  <ElementType name="FDTFailSafeData" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fsDeviceTag" required="no"/>
    <attribute type="fsBusMasterTag" required="no"/>
    <attribute type="fsDeviceFBTag" required="yes"/>
    <attribute type="fsBulkData" required="yes"/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="FDTFailSafeData" minOccurs="0" maxOccurs="1"/>
  </ElementType>
</Schema>

```

Example:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTFailSafeDataSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <FDTFailSafeData fsDeviceTag="MyDevice" fsBusMasterTag="MyBusMasterTag" fsDeviceFBTag="MyDeviceFBTag"
fsBulkData="1"/>
</FDT>
```

## 12.16 DTMTopologyScanSchema

Used at: [IFdtChannelSubTopology::OnScanResponse\(\)](#)

The XML document contains the result of the topology scan

Tag	Description
TopologyScan	Collection of bus specific information concerning the sca" re"ult
FDT	Root tag

```
<?xml version="1.0"?>
<Schema name="DTMTopologyScanSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:fdtpfidevice="x-
schema:DTMProfibusDeviceSchema.xml" xmlns:fdthartdevice="x-schema:DTMHARTDeviceSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
  <!--Definition of Elements-->
  <ElementType name="TopologyScan" content="eltOnly" model="open">
    <attribute type="fdt:nodeId" required="no"/>
    <group order="one">
      <element type="fdthartdevice:HARTDevice" minOccurs="0" maxOccurs="*" />
      <element type="fdtpfidevice:ProfibusDevice" minOccurs="0" maxOccurs="*" />
    </group>
  </ElementType>
</Schema>
```

Example:

```
<?xml version="1.0"?>
<TopologyScan xmlns="x-schema:DTMTopologyScanSchema.xml" xmlns:fdthart="x-
schema:FDTHARTCommunicationSchema.xml" xmlns:dtminfo="x-schema:DTMInformationSchema.xml"
xmlns:fdthartdevice="x-schema:DTMHARTDeviceSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" >
  <fdthartdevice:HARTDevice fdthart:manufacturerId="22" fdthart:deviceTypeId="133" fdt:subDeviceType="TypA"
fdthart:shortAddress="2" fdt:tag=""/>
  <fdthartdevice:HARTDevice fdthart:manufacturerId="22" fdthart:deviceTypeId="133" fdt:subDeviceType="TypB"
fdthart:shortAddress="3" fdt:tag=""/>
</TopologyScan>
```

## 12.17 *FDTOperationPhaseSchema*

Used at: [IDtm::GetFunctions\(\)](#)

The XML document contains the definition of the operation phases.

Attribute	Description
operationPhase	Attribute containing the current operation phase (refer to chapter 2.9 Basic Operation phases)

Tag	Description
FDT	Root tag

```
<Schema name="FDTOperationPhaseSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <!--Definition of Attributes-->
  <AttributeType name="operationPhase" dt:type="enumeration" dt:values="notSupported engineering commissioning runtime service"/>
  <ElementType name="FDT" content="empty" model="closed">
    <attribute type="operationPhase" required="yes"/>
  </ElementType>
</Schema>
```

Example :

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTOperationPhaseSchema.xml" operationPhase="runtime"/>
```

## 12.18 *DTMInitSchema*

Used at: `IDtm::InitNew()`

The XML document contains information concerning the device type.

Tag	Description
FDT	Root tag

```
<Schema name="DTMInitSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:dtminfo="x-schema:DTMInformationSchema.xml">
  <ElementType name="FDT" content="eltOnly" model="closed">
    <element type="fdt:DtmDeviceType"/>
  </ElementType>
</Schema>
```

Example :

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMInitSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:dtminfo="x-schema:DTMInformationSchema.xml">
  <fdt:DtmDeviceType>
    <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
    <fdt:SupportedLanguages>
      <fdt:LanguageId languageId="1131"/>
    </fdt:SupportedLanguages>
    <fdt:BusCategories>
      <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
    </fdt:BusCategories>
    <fdt:DeviceTypeVariant deviceTypeVariant="PM255-EX" deviceTypeVariantInfo="PM255-EX Info"/>
  </fdt:DtmDeviceType>
</FDT>
```

## 12.19 *FDTUserMessageSchema*

Used at: [IFdtDialog::UserDialog\(\)](#)

The XML document contains the definition of a user message.

Attribute	Description
helpContext	Help context (e.g. the reference number within the help file)
helpFile	Definition of the help file
messageButtons	Definition of the button types which shall appear within the message box
messageDefault	Definition of the default button
messageType	Type of a message like exclamation, information, ...
resultMessage	Definition of the result of the user interaction
resultStatus	
title	Window title of the message box

Tag	Description
FDT	Root tag
FDTUserMessage	Definition of the whole message
TextLine	Definition of a single text line within the message

```
<Schema name="FDTUserMessageSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="messageType" dt:type="enumeration" dt:values="messageExclamation messageInformation messageQuestion messageStop"/>
  <AttributeType name="messageButtons" dt:type="enumeration" dt:values="buttonsAbortRetryIgnore buttonsOk buttonsOkCancel buttonsRetryCancel buttonsYesNo buttonsYesNoCancel"/>
  <AttributeType name="messageDefault" dt:type="enumeration" dt:values="buttonAbort buttonRetry buttonIgnore buttonOk buttonCancel buttonYes buttonNo"/>
  <AttributeType name="resultMessage" dt:type="enumeration" dt:values="nobutton buttonAbort buttonRetry buttonIgnore buttonOk buttonCancel buttonYes buttonNo"/>
  <AttributeType name="resultStatus" dt:type="enumeration" dt:values="notSupported denied systemResponse ok"/>
  <AttributeType name="title" dt:type="string"/>
  <AttributeType name="helpFile" dt:type="string"/>
  <AttributeType name="helpContext" dt:type="number"/>
  <!-- ApplicationId specifies the standard user interface called -->
  <ElementType name="TextLine" content="mixed" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:string" required="yes"/>
  </ElementType>
  <ElementType name="FDTUserMessage" content="mixed" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="messageType" required="yes"/>
    <attribute type="messageButtons" required="yes"/>
    <attribute type="messageDefault" required="yes"/>
    <attribute type="title" required="yes"/>
    <attribute type="helpFile" required="no"/>
    <attribute type="helpContext" required="no"/>
    <group order="many">
      <element type="TextLine" minOccurs="0" maxOccurs="*" />
      <element type="fdt:DtmVariable" minOccurs="0" maxOccurs="*" />
    </group>
  </ElementType>
</Schema>
```

```

    </group>
    <attribute type="resultMessage" required="no"/>
    <attribute type="resultStatus" required="no"/>
  </ElementType>
  <ElementType name="FDT" content="mixed" model="closed">
    <attribute type="fdt:nodId" required="no"/>
    <element type="FDTUserMessage" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>

```

Example:

```

<?xml version="1.0" encoding="UTF-8"?>
<FDT xmlns="x-schema:FDTUserMessageSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <FDTUserMessage messageType="messageQuestion" messageButtons="buttonsOkCancel"
messageDefault="buttonCancel" title="Information">
    <TextLine fdt:string="Please give in your name:"/>
    <TextLine fdt:string="Please give in your name:"/>
    <fdt:DtmVariable name="strUser">
      <fdt:Value>
        <fdt:Variant dataType="ascii">
          <fdt:StringData string="New Name"/>
        </fdt:Variant>
      </fdt:Value>
    </fdt:DtmVariable>
    <TextLine fdt:string="(more than 2 characters needed)"/>
  </FDTUserMessage>
</FDT>

```

## 12.20 DTMInfoListSchema

Used at: [IFdtTopology::GetDtmInfoList\(\)](#)

The XML document contains a list of DTM information

Tag	Description
DtmInfoList	Collection of elements of DtmInfo structures
FDT	Root tag

```

<Schema name="DTMInfoListSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:dtmInfo="x-schema:DTMInformationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Elements-->
  <ElementType name="DtmInfoList" content="eltOnly" model="closed" order="seq">
    <attribute type="fdt:nodId" required="no"/>
    <element type="dtmInfo:DtmInfo" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodId" required="no"/>
    <element type="DtmInfoList" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>

```

Example:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMInfoListSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:dtmInfo="x-schema:DTMInformationSchema.xml">
  <DtmInfoList>
    <dtmInfo:DtmInfo>

```

```

<dtmInfo:FDTVersion major="1" minor="2"/>
<fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-07-01"/>
<dtmInfo:DtmDeviceTypes>
  <fdt:DtmDeviceType>
    <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
    <fdt:SupportedLanguages>
      <fdt:LanguageId languageId="1131"/>
    </fdt:SupportedLanguages>
    <fdt:BusCategories>
      <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
    </fdt:BusCategories>
  </fdt:DtmDeviceType>
  <fdt:DtmDeviceType>
    <fdt:VersionInformation name="myname" vendor="myVendor" version="1.2" date="2000-03-05"/>
    <fdt:SupportedLanguages>
      <fdt:LanguageId languageId="1131"/>
    </fdt:SupportedLanguages>
    <fdt:BusCategories>
      <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
      <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
    </fdt:BusCategories>
  </fdt:DtmDeviceType>
</dtmInfo:DtmDeviceTypes>
</dtmInfo:DtmInfo>
<dtmInfo:DtmInfo>
  <dtmInfo:FDTVersion major="1" minor="2"/>
  <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-07-23"/>
  <dtmInfo:DtmDeviceTypes>
    <fdt:DtmDeviceType>
      <fdt:VersionInformation name="myname" vendor="myVendor" version="1.1" date="2000-03-05"/>
      <fdt:SupportedLanguages>
        <fdt:LanguageId languageId="1131"/>
      </fdt:SupportedLanguages>
      <fdt:BusCategories>
        <fdt:BusCategory busCategory="036D1498-387B-11D4-86E1-00E0987270B9"/>
      </fdt:BusCategories>
    </fdt:DtmDeviceType>
  </dtmInfo:DtmDeviceTypes>
</dtmInfo:DtmInfo>
</DtmInfoList>
</FDT>

```

## 12.21 *FDTTopologyImportExportSchema*

Used at: Private functions of the Frame Application. No interface defined within the FDT Specification needed.

The XML document describes the topology of system. It can be used for import and export of the topology.

Attribute	Description
category	Description of the structuring type
comment	Comment within the context of the element
dataStream	Description of the data
date	Date of the export
frameApplicationTag	Frame Application specific tag used for identification and navigation

Attribute	Description
gatewayBusCategory	Unique identifier for a supported bus type like Profibus or HART according to the FDT specific CATID
identifier	Unique identifier for the element
label	Human readable name of the object
proglId	ProglId of the DTM
projectName	Unique identifier for the project within the name space of the Frame Application
propertyName	Name of a property
propertyValue	Value of a property
propertyValueType	Type of a property
systemTag	System Tag
time	Time of the export
uuid	UUID of the DTM

Tag	Description
ChannelNode	Description of a channel within the topology
ChannelSubNodes	Sub nodes of a channel
Dataset	Dataset of the DTM
Description	General information of the topology
DTM	Description of the DTM
BTM	Description of the BTM
DTMNode	Description of a DTM within the topology
BTMNode	Description of a BTM within the topology
DTMSubNodes	Sub nodes of a DTM
FDT	Root tag
FDTChannelType	Description of the channel component
ProglId	ProglId of the DTM
Property	Description of a property within a PropertyBag
PropertyBag	Description of a PropertyBag
Source	Description of the source of the topology
Stream	Description of a stream
StructuringNode	Structuring information of the topology
StructuringSubNodes	Sub notes of a 'StructuringNode'
UUID	UUID of the DTM

A Frame Application aware of DTM instances handling redundant slaves should use a BusInformation element within a DTM element of the import/export document.

```
<Schema name="FDTTopologyImportExportSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
  xmlns:dtmInfo="x-schema:DTMInformationSchema.xml"
  xmlns:btm="x-schema:BtmDataTypesSchema.xml"
  xmlns:btmInfo="x-schema:BtmInformationSchema.xml"
  xmlns:dtmParam="x-schema:DTMParameterSchema.xml"
>
  <!--Definition of Attributes-->
  <AttributeType name="date" dt:type="date"/>
  <AttributeType name="time" dt:type="time"/>
  <AttributeType name="comment" dt:type="string"/>
  <AttributeType name="identifier" dt:type="string"/>
  <AttributeType name="label" dt:type="string"/>
  <AttributeType name="category" dt:type="string"/>
  <AttributeType name="uuid" dt:type="uuid"/>
  <AttributeType name="proglId" dt:type="string"/>
  <AttributeType name="systemTag" dt:type="string"/>
  <AttributeType name="dataStream" dt:type="bin.hex"/>
  <AttributeType name="projectName" dt:type="string"/>
```



```

<AttributeType name="propertyName" dt:type="string"/>
<AttributeType name="propertyValue" dt:type="bin.hex"/>
<AttributeType name="propertyValueType" dt:type="ui4"/>
<AttributeType name="frameApplicationTag" dt:type="string"/>
<AttributeType name="gatewayBusCategory" dt:type="uuid"/>
<!--Definition of Elements-->
<ElementType name="ChannelSubNodes" content="eltOnly" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <element type="DTMNode" minOccurs="0" maxOccurs="*" />
  <element type="BTMNode" minOccurs="0" maxOccurs="*" />
  <element type="StructuringNode" minOccurs="0" maxOccurs="*" />
</ElementType>
<ElementType name="FDTChannelType" content="eltOnly" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1"/>
  <attribute type="gatewayBusCategory" required="no"/>
</ElementType>
<ElementType name="ChannelNode" content="eltOnly" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <attribute type="frameApplicationTag" required="no"/>
  <attribute type="label" required="yes"/>
  <attribute type="fdt:tag" required="yes"/>
  <attribute type="fdt:id" required="yes"/>
  <element type="FDTChannelType" minOccurs="1" maxOccurs="1"/>
  <element type="ChannelSubNodes" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="StructuringSubNodes" content="eltOnly" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <element type="DTMNode" minOccurs="0" maxOccurs="*" />
  <element type="BTMNode" minOccurs="0" maxOccurs="*" />
  <element type="StructuringNode" minOccurs="0" maxOccurs="*" />
</ElementType>
<ElementType name="StructuringNode" content="eltOnly" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <attribute type="identifier" required="yes"/>
  <attribute type="label" required="yes"/>
  <attribute type="category" required="no"/>
  <element type="StructuringSubNodes" minOccurs="1" maxOccurs="1"/>
</ElementType>
<ElementType name="Property" content="empty" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <attribute type="propertyName" required="yes"/>
  <attribute type="propertyValue" required="yes"/>
  <attribute type="propertyValueType" required="yes"/>
</ElementType>
<ElementType name="PropertyBag" content="eltOnly" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <element type="Property" minOccurs="0" maxOccurs="*" />
</ElementType>
<ElementType name="Stream" content="empty" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <attribute type="dataStream" required="yes"/>
</ElementType>
<ElementType name="ImportExport" content="empty" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <attribute type="dataStream" required="yes"/>
</ElementType>
<ElementType name="Dataset" content="eltOnly" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <attribute type="systemTag" required="yes"/>
  <group minOccurs="1" maxOccurs="1" order="one">
    <element type="PropertyBag" minOccurs="1" maxOccurs="1"/>
    <element type="Stream" minOccurs="1" maxOccurs="1"/>
    <element type="ImportExport" minOccurs="1" maxOccurs="1"/>
  </group>
</ElementType>
<ElementType name="ProglId" content="empty" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <attribute type="proglId" required="yes"/>
</ElementType>
<ElementType name="UUID" content="empty" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <attribute type="uuid" required="yes"/>
</ElementType>
<ElementType name="DTM" content="eltOnly" model="closed">
  <attribute type="fdt:nodeId" required="no"/>

```

```

<element type="dtmInfo:FDTVersion" minOccurs="1" maxOccurs="1"/>
<element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1" />
<element type="fdt:DtmDeviceType" minOccurs="1" maxOccurs="1"/>
<element type="dtmParam:BusInformation" minOccurs="0" maxOccurs="1"/>
<group order="one" minOccurs="1" maxOccurs="1">
  <element type="UUID"/>
  <element type="Proglid"/>
</group>
</ElementType>
<ElementType name="BTM" content="eltOnly" model="closed">
  <attribute type="fdt:nodelid" required="no"/>
  <element type="dtmInfo:FDTVersion" minOccurs="1" maxOccurs="1"/>
  <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1" />
  <element type="btm:BtmBlockType" minOccurs="1" maxOccurs="1"/>
  <group order="one" minOccurs="1" maxOccurs="1">
    <element type="UUID"/>
    <element type="Proglid"/>
  </group>
</ElementType>
<ElementType name="DTMSubNodes" content="eltOnly" model="closed">
  <attribute type="fdt:nodelid" required="no"/>
  <element type="ChannelNode" minOccurs="1" maxOccurs="1"/>
</ElementType>
<ElementType name="DTMNode" content="eltOnly" model="closed">
  <attribute type="fdt:nodelid" required="no"/>
  <attribute type="identifier" required="yes"/>
  <attribute type="label" required="yes"/>
  <element type="DTM" minOccurs="1" maxOccurs="1"/>
  <element type="Dataset" minOccurs="0" maxOccurs="1"/>
  <element type="DTMSubNodes" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="BTMNode" content="eltOnly" model="closed">
  <attribute type="fdt:nodelid" required="no"/>
  <attribute type="identifier" required="yes"/>
  <attribute type="label" required="yes"/>
  <element type="BTM" minOccurs="1" maxOccurs="1"/>
  <element type="Dataset" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="Source" content="empty" model="closed">
  <attribute type="fdt:nodelid" required="no"/>
  <attribute type="identifier" required="no"/>
  <attribute type="label" required="yes"/>
  <attribute type="projectName" required="yes"/>
</ElementType>
<ElementType name="Description" content="eltOnly" model="closed">
  <attribute type="fdt:nodelid" required="no"/>
  <attribute type="date" required="yes"/>
  <attribute type="time" required="yes"/>
  <attribute type="comment" required="no"/>
  <element type="Source" minOccurs="1" maxOccurs="1"/>
</ElementType>
<ElementType name="FDT" content="eltOnly" model="closed">
  <attribute type="fdt:nodelid" required="no"/>
  <element type="Description" minOccurs="1" maxOccurs="1"/>
  <group order="one" minOccurs="1" maxOccurs="1">
    <element type="DTMNode" minOccurs="1" maxOccurs="1"/>
    <element type="BTMNode" minOccurs="1" maxOccurs="1"/>
    <element type="StructuringNode" minOccurs="1" maxOccurs="1"/>
  </group>
</ElementType>
</Schema>

```

#### Example:

```

<?xml version="1.0" ?>
<FDT xmlns="x-schema:FDTTopologyImportExportSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
xmlns:dtmInfo="x-schema:DTMInformationSchema.xml">
  <Description date="2002-11-20" time="11:11:11">
    <Source identifier="FBB" label="ABB Fieldbus Builder" projectName="LEBUILDPC-WAL System"/>
  </Description>
  <StructuringNode identifier="B13A8777-AFEB-457e-A7DF-A5BEEE498CAC" label="Root" category="Domain">
    <StructuringSubNodes>
      <DTMNode identifier="2888AF1B-2D50-446D-86BE-9D16D78DF580" label="Modem 1">
        <DTM>

```

```

        <dtmInfo:FDTVersion major="1" minor="0"/>
        <fdt:VersionInformation name="HARTCommunication" vendor="ABB Automation GmbH" version="1.0"
date="2000-12-20"/>
        <fdt:DtmDeviceType>
            <fdt:VersionInformation name="HARTCommunication" vendor="ABB Automation GmbH" version="1.0"
date="2000-12-20"/>
            <fdt:SupportedLanguages>
                <fdt:LanguageId languageId="1131"/>
            </fdt:SupportedLanguages>
            <fdt:BusCategories>
                <fdt:BusCategory busCategory="036D1498-387B-11D4-86E1-00E0987270B9"/>
            </fdt:BusCategories>
        </fdt:DtmDeviceType>
        <ProgId progId="HARTComm.ABBHARTCommDTM.1"/>
    </DTM>
    <DTMSubNodes>
        <ChannelNode frameApplicationTag="Channel 1" label="HART Channel" fdt:tag="HARTChannel" fdt:id="1">
            <FDTChannelType gatewayBusCategory="036D1498-387B-11D4-86E1-00E0987270B9">
                <fdt:VersionInformation name="HARTCommunication" vendor="ABB Automation GmbH"
version="1.0" date="2000-12-20"/>
            </FDTChannelType>
            <ChannelSubNodes>
                <DTMNode identifier="3E58A0F2-890F-4118-9463-78C880B0810E" label="GH 1" >
                    <DTM>
                        <dtmInfo:FDTVersion major="1" minor="2"/>
                        <fdt:VersionInformation name="HARTCommunication" vendor="ABB Automation GmbH"
version="1.0" date="2000-12-20"/>
                        <fdt:DtmDeviceType readAccess="1" writeAccess="0" manufacturerId="0" deviceTypeId="0"
deviceTypeInfo="-"/>
                        <fdt:VersionInformation readAccess="1" writeAccess="0" name="Generic HART"
vendor="ABB" version="1.0" date="2002-11-17"/>
                        <fdt:SupportedLanguages>
                            <fdt:LanguageId languageId="1033"/>
                        </fdt:SupportedLanguages>
                        <fdt:BusCategories>
                            <fdt:BusCategory busCategory="036D1498-387B-11D4-86E1-00E0987270B9"/>
                        </fdt:BusCategories>
                    </fdt:DtmDeviceType>
                    <ProgId progId="GenericHARTDtm.Main"/>
                </DTM>
                <Dataset systemTag="3E58A0F2-890F-4118-9463-78C880B0810E">
                    <Stream
dataStream="28000000500065007200730069007300740065006e00740050006100720061006d00650074006500720000000800a0
3200003c003f0078006d006c002000760065007200730069006f006e003d00220031002e00300022003f003e000d000a003c004600
44005400200078006d006c006e0073003d00220078002d0073006300680065006d0061003a00440054004d0050006100720061006
d"/>
                    </Dataset>
                </DTMNode>
            </ChannelSubNodes>
        </DTMSubNodes>
    </DTMNode>
</StructuringSubNodes>
</StructuringNode>
</FDT>

```

## 12.22 DTMDDeviceListSchema

Used at: [IFdtChannelSubTopology2::SetChildrenAddresses\(\)](#)

The XML document contains a list of DTM system tags and corresponding device addresses to set.

Attribute	Description
busAddress	Bus address to set.
errorDescription	Detailed error information in case of dtmSpecificError
errorInfo	<p>To be used when DTMDDeviceListSchema is returned to indicate</p> <ul style="list-style-type: none"> <li>error summary if used as attribute of DeviceList element and</li> <li>error information for a specific address when used in Device element.</li> </ul> <p>Enumeration:</p> <ol style="list-style-type: none"> <li>“ok” Address was set successfully – used in DeviceList as well as in Device</li> <li>“failedToSet” – used in Device to indicate failed SetParameter.</li> <li>“failedDuplicateAddress” - used to indicate that the address is already available and couldn't be set.</li> <li>“cancelled” – used to indicate that the setting was cancelled by user.</li> <li>“dtmSpecificError” – indicates an DTM specific error. In this case error description text is to be used to give more detailed error information.</li> </ol>
showUserInterface	<p>Indicates if the Communication Channel should open a user interface in order to get a protocol specific address selection by decision of the user.</p> <p>Enumeration:</p> <ol style="list-style-type: none"> <li>openUserInterface – user interface should be opened to request the address from user</li> <li>noUserInterface – no user interface should be opened</li> <li>setNextValidAddress - Communication Channel has to set the next valid address without using a user interface. In this case the busAddress is not used by Communication Channel.</li> </ol> <p>If this attribute is not set, NoUserInterface is assumed.</p>

### Comments

The attribute busAddress is in string format and can be accessed by the Frame Application in protocol independent way. The contents of the busAddress string is protocol specific. It has to be handled according to the protocol rules.

Element	Description
BusAddress	Contains a single busAddress
BusAddressList	<p>Contains BusAddressList</p> <p>In order to support devices with more than one addresses (one bus address and optional redundancy addresses), it must be possible to provide more than one address.</p>
Device	Contains device identification information and system tag of corresponding DTM.
DeviceList	<p>Collection of Devices and optional error summary.</p> <p>The attribute errorInfo must be set if at least one Device of the list got an errorInfo.</p>

```

<Schema name="DTMDDeviceListSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <AttributeType name="errorDescription" dt:type="string"/>
  <AttributeType name="showUserInterface" dt:type="enumeration" dt:values=" openUserInterface noUserInterface setNextValidAddress"/>
  <AttributeType name="errorInfo" dt:type="enumeration" dt:values="ok failedToSet failedDuplicateAddress cancelled dtmSpecificError"/>
  <ElementType name="BusAddress" content="empty" model="closed">
    <attribute type="fdt:nodelid" required="no"/>
  
```

```

    <attribute type="fdt:busAddress" required="yes"/>
  </ElementType>
  <ElementType name="BusAddressList" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="BusAddress" minOccurs="1" maxOccurs="*/>
  </ElementType>
  <ElementType name="Device" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:systemTag" required="yes"/>
    <attribute type="errorInfo" required="no"/>
    <attribute type="errorDescription" required="no"/>
    <element type="BusAddressList" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="DeviceList" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="errorInfo" required="no"/>
    <attribute type="errorDescription" required="no"/>
    <attribute type="showUserInterface" required="no"/>
    <element type="Device" minOccurs="1" maxOccurs="*/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="DeviceList" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>

```

#### Examples:

```

<?xml version="1.0" encoding="UTF-8"?>
<devList:FDT xmlns:devList="x-schema:DTMDeviceListSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <devList:DeviceList>
    <devList:Device fdt:systemTag="DTM-1">
      <devList:BusAddressList>
        <devList:BusAddress fdt:busAddress="1"/>
      </devList:BusAddressList>
    </devList:Device>
    <devList:Device fdt:systemTag="DTM-2">
      <devList:BusAddressList>
        <devList:BusAddress fdt:busAddress="2"/>
      </devList:BusAddressList>
    </devList:Device>
    <devList:Device fdt:systemTag="DTM-3">
      <devList:BusAddressList>
        <devList:BusAddress fdt:busAddress="3"/>
      </devList:BusAddressList>
    </devList:Device>
  </devList:DeviceList>
</devList:FDT>

```

```

<?xml version="1.0" encoding="UTF-8"?>
<devList:FDT xmlns:devList="x-schema:DTMDeviceListSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <devList:DeviceList>
    <devList:Device fdt:systemTag="DTM-1" errorInfo="failedToSet" errorDescription="DTM-1 not available">
      <devList:BusAddressList>
        <devList:BusAddress fdt:busAddress="1"/>
      </devList:BusAddressList>
    </devList:Device>
    <devList:Device fdt:systemTag="DTM-2" errorInfo="failedDuplicateAddress">
      <devList:BusAddressList>
        <devList:BusAddress fdt:busAddress="2"/>
      </devList:BusAddressList>
    </devList:Device>
    <devList:Device fdt:systemTag="DTM-3" errorInfo="ok">
      <devList:BusAddressList>
        <devList:BusAddress fdt:busAddress="3"/>
      </devList:BusAddressList>
    </devList:Device>
  </devList:DeviceList>
</devList:FDT>

```

## 12.23 *DTMSystemGuiLabelSchema*

Used at: [IDtm2::SetSystemGuiLabel\(\)](#)

The XML document provides a system label.

Tag	Description
SystemGuiLabel	Description of the system label

```
<Schema name="DTMSystemGuiLabelSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
```

```
  <!--Definition of Elements-->
  <ElementType name="SystemGuiLabel" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:systemGuiLabel" required="no"/>
  </ElementType>

  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="SystemGuiLabel"/>
  </ElementType>
</Schema>
```

Example:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMSystemGuiLabelSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <SystemGuiLabel fdt:nodeId="myId" fdt:systemGuiLabel="This is the system label of the DTM instance" />
</FDT>
```

## 12.24 *DTMStateSchema*

Element	Description
PreviousState	previous state of DTM state machine
CurrentState	actual state of DTM state machine
DtmStateTransition	state transition between DTM state machine states

```
<Schema name="DTMStateSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
```

```
  <!--Definition of Attributes-->
  <ElementType name="PreviousState" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:dtmStateMachine" required="yes"/>
  </ElementType>
  <ElementType name="CurrentState" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:dtmStateMachine" required="yes"/>
  </ElementType>
```

```

</ElementType>
<ElementType name="DtmStateTransition" content="eltOnly" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <element type="PreviousState" minOccurs="1" maxOccurs="1"/>
  <element type="CurrentState" minOccurs="1" maxOccurs="1"/>
  <element type="fdt:CommunicationError" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="FDT" content="eltOnly" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <element type="DtmStateTransition" minOccurs="1" maxOccurs="1"/>
</ElementType>
</Schema>

```

Example:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMStateSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <DtmStateTransition>
    <PreviousState fdt:dtmStateMachine="online"/>
    <CurrentState fdt:dtmStateMachine="communicationSet"/>
    <fdt:CommunicationError tag="myTag" communicationError="noConnection" descriptor="device not found"/>
  </DtmStateTransition>
</FDT>

```

## 12.25 DTMEnvironmentSchema

Tag	Description
FrameVersion	Describes the version of the Frame Application

```

<Schema name="DTMEnvironmentSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:fdtvers="x-schema:DTMInformationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <ElementType name="FrameVersion" content="eltOnly" model="closed" order="seq">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="fdtvers:FDTVersion" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="FrameVersion" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>

```

Example:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMEnvironmentSchema.xml"
  xmlns:fdtvers="x-schema:DTMInformationSchema.xml"
  xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
  xmlns:fdtappid="x-schema:FDTApplicationIdSchema.xml">
  <FrameVersion>
    <fdtvers:FDTVersion major="1" minor="2" release="0" build="1" />
  </FrameVersion>
</FDT>

```

## 12.26 FDTConnectResponseSchema

Tag	Description
ParentVersion	Describes the version of the parent component

```
<Schema name="FDTConnectResponseSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:fdtvers="x-schema:DTMInformationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <ElementType name="ParentVersion" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="fdtvers:FDTVersion" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="ParentVersion" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>
```

Example:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTConnectResponseSchema.xml"
  xmlns:fdtvers="x-schema:DTMInformationSchema.xml"
  xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
  xmlns:fdtappid="x-schema:FDTApplicationIdSchema.xml">
  <ParentVersion>
    <fdtvers:FDTVersion major="1" minor="2" release="0" build="1" />
  </ParentVersion>
</FDT>
```

## 12.27 *TypeRequestSchema*

Used as parameter at: [GetDeviceIdentificationInformation\(\)](#) to define the type for which the identification is requested.

Element	Description
FDT	root tag

```
<Schema name="TypeRequestSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:btm="x-schema:BtmDataTypesSchema.xml"
  xmlns:dtminfo="x-schema:DTMInformationSchema.xml">
  <ElementType name="FDT" content="eltOnly" model="closed">
    <group order="one" minOccurs="1" maxOccurs="1">
      <element type="fdt:DtmDeviceType"/>
      <element type="btm:BtmBlockType"/>
    </group>
  </ElementType>
</Schema>
```



## 12.28 *FDTScanRequestSchema*

Used at: [IFdtChannelScan::ScanRequest\(\)](#)

The XML document contains the range of addresses to scan. Following options are provided:

1. Address range, which could be used also for one address.
2. All addresses provided by fieldbus.
3. Protocol UUID to identify the protocol in case more than one protocol is supported.

Attribute	Description
busAddressBegin	Bus address to be used as start address for topology scanning. Format according scan response.
busAddressEnd	Bus address to be used as end address. Format according scan response.
scanMode	Enumeration defines if <ul style="list-style-type: none"> <li>• the complete address range should be scanned or if</li> <li>• the CommDTM should open a user interface to request an address selection from the user.</li> </ul>

Tag	Description
BusAddressRange	Contains start and end address of requested bus scan.
BusAddressRanges	Containing list of bus address ranges
FDT	Root tag
ScanMode	Element defines if complete bus scan is requested or address is to be selected via user interface.

```
<?xml version="1.0"?>
<Schema name="FDTTopologyScanRequestSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->

  <AttributeType name="busAddressBegin" dt:type="string"/>
  <AttributeType name="busAddressEnd" dt:type="string"/>
  <AttributeType name="scanMode" dt:type="enumeration" dt:values="allAddresses openGUI"/>

  <!--Definition of Elements-->

  <ElementType name="BusAddressRange" content="empty" model="closed">
    <attribute type="busAddressBegin" required="yes"/>
    <attribute type="busAddressEnd" required="no"/>
  </ElementType>

  <ElementType name="BusAddressRanges" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:busCategory" required="yes"/>
    <element type="BusScanAddressRange" minOccurs="1" maxOccurs="*/>
  </ElementType>

  <ElementType name="ScanMode" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="scanMode" required="yes"/>
    <attribute type="fdt:busCategory" required="yes"/>
  </ElementType>

  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
```

```

    <group order="one">
      <element type="BusScanAddressRanges"/>
      <element type="ScanMode"/>
    </group>
  </ElementType>
</Schema>

```

### Example 1 : Scan all addresses

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTTopologyScanRequestSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <ScanMode scanMode="allAddresses" busCategory="036D1498-387B-11D4-86E1-00E0987270B9"/>
</FDT>

```

### Example 2 : Display addresses to user for selection of the address range

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTTopologyScanRequestSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <ScanMode scanMode="openGUI" busCategory="036D1498-387B-11D4-86E1-00E0987270B9"/>
</FDT>

```

### Example 3 : Scan address range e.g. 5 - 10

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTTopologyScanRequestSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <BusScanAddressRanges busCategory="036D1498-387B-11D4-86E1-00E0987270B9">
    <BusScanAddressRange busAddressBegin="5" busAddressEnd="10"/>
  </BusScanAddressRanges>
</FDT>

```

### Example 4 : Scan 2 address ranges

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTTopologyScanRequestSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <BusScanAddressRanges busCategory="036D1498-387B-11D4-86E1-00E0987270B9">
    <FieldbusScanAddressRange busAddressBegin="5" busAddressEnd="10"/>
    <FieldbusScanAddressRange busAddressBegin="15" busAddressEnd="20"/>
  </BusScanAddressRanges>
</FDT>

```

### Example 5 : Scan only one address

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTTopologyScanRequestSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <BusScanAddressRanges busCategory="036D1498-387B-11D4-86E1-00E0987270B9">
    <FieldbusScanAddressRange busAddressBegin="5"/>
  </BusScanAddressRanges>
</FDT>

```

## 12.29 FDTxxxIdentSchema

Protocol specific base schema. The protocol specific namespace for [FDTxxxxDeviceTypeIdentSchema](#) and [FDTxxxxScanIdentSchema](#) is defined in this schema.

Example: FDTHARTIdentSchema.xml

## **12.30      *FDTxxxDeviceTypelidentSchema***

Used at [IDtmInformation2::GetDeviceIdentificationInformation\(\)](#)

This schema is created for each protocol especially. The XML document is transformed by the protocol specific xsl file. The result of the transformation must validate against [DTMDeviceTypelidentSchema](#).

This schema references to the protocol specific namespace defined in [FDTxxxIdentSchema](#).

The DTM must provide at least one attribute or the regular expression element for the required identification elements

Example : [FDTHARTDeviceTypelidentSchema.xml](#)

## **12.31      *FDTxxxScanIdentSchema***

Used at: [IDtmScanEvents::OnScanResponse\(\)](#)

This schema is protocol specific. The XML document is transformed by the [protocol specific xsl](#) file. The result of the transformation must validate against [DTMScanIdentSchema](#).

This schema references to the protocol specific namespace defined in [FDTxxxIdentSchema](#).

Example : [FDTHARTScanIdentSchema.xml](#)

## **12.32      *DTMIdentSchema***

The protocol independent identification namespace for [DTMScanIdentSchema](#) and [DTMDeviceTypelidentSchema](#) is defined in this schema.

Attribute	Description
Match	Attribute contains a regular expression string, which must match with an attribute provided by scan result.
Nomatch	Attribute contains a regular expression string, which must not match with an attribute provided by scan result.
Name	Contains the name of one single identification information
Value	Contains the value of one single identification information
protocolSpecificName	Fieldbus protocol specific name. This attribute provides a reference to the fieldbus specification.
idDTMSupportLevel	<p>Defines the support level of a DTMDeviceType for a physical device. This attribute can be used by a Frame Application to display the support level of a DTM to the user. Users may use this information for an assignment decision.</p> <p><b>genericSupport:</b> DTMDeviceType applies to all kind of physical devices of the corresponding fieldbus protocol</p> <p><b>profileSupport:</b> DTMDeviceType applies to physical devices of a certain profile of the fieldbus protocol.</p> <p><b>blockspecificProfileSupport</b> DTMDeviceType applies to blocks included in physical device types (e.g.: Pressure TransducerBlock in Profibus PA)</p> <p><b>specificSupport</b> (DTMDeviceType is developed for this physical device type.</p> <p><b>identSupport:</b> The DTMDeviceType is capable to identify the physical device in a vendor specific manner and to propose a better DTMDeviceType</p>

Tag	Description
RegExpr	Contains one or more regular expression attributes (refer to <a href="#">Regular Expression Specification</a> chapter) of type match or nomatch.

```

<Schema name="DTMIdentSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="match" dt:type="string"/>
  <AttributeType name="nomatch" dt:type="string"/>
  <AttributeType name="name" dt:type="string"/>
  <AttributeType name="value" dt:type="string"/>
  <AttributeType name="protocolSpecificName" dt:type="string"/>
  <AttributeType name="idDTMSupportLevel" dt:type="enumeration" dt:values="genericSupport profileSupport
blockspecificProfileSupport specificSupport identSupport"/>
  <!--Definition of Elements-->
  <ElementType name="RegExpr" content="empty" model="closed">
    <attribute type="match" required="no"/>
    <attribute type="nomatch" required="no"/>
  </ElementType>
</Schema>

```

## 12.33 DTMScanIdentSchema

The protocol specific XML document provided by ScanResponse is to be transformed by the protocol specific XSLT. The transformation result must validate against [DTMScanIdentSchema](#).

Attribute	Description
resultState	Marks the XML document as first provisional result provided by ScanResponse(). DTM will send further provisional XMLs (resultState = "provisional") and one result document at the end of the scan operation (resultState = "final"). In case of an error the result indicates this by setting resultState = error.
configuredState	<p>The attribute is only used for protocols, which have a master configuration like Profibus. For all other protocols "notApplicable" is to be used.</p> <p>This attributes defines, if a scanned module connected to this bus address is in configured state or . The attribute is optionally used by Frame Applications to display the information to the user.</p> <p>Following values are valid in the enumeration:</p> <ul style="list-style-type: none"> <li>• configuredAndPhysicallyAvailable</li> <li>• configuredAndNotPhysicallyAvailable</li> <li>• availableButNotConfigured</li> <li>• notApplicable</li> </ul>

Tag	Description
IdAddress	Contains the field bus address of a device or a block address. This attribute is only available in scan online process and for information only. It cannot be used for matching.
IdBusProtocol	Provides the protocol variant, e.g. Profibus PA
IdBusProtocolVersion	Contains the version of the protocol, e.g.: 5 or 6 in case of HART, or 3.0 in case of Profibus PA.
IdManufacturer	Identifies the manufacturer of the device or block
IdTypeID	Identifies the device or block type
IdSoftwareRevision	Contains the software version of the device or block
IdDeviceTag	Contains the tag of the scanned device or block instance. This attribute is only available in scan result and for information only. It cannot be used for matching.
IdHardwareRevision	Contains the hardware version of the device or block
IdSerialNumber	<p>Contains the serial number of a scanned device or block. This attribute is only available in scan online process and for information only. It cannot be used for matching.</p> <p><u>Note:</u> This serialnumber might be only unique for one manufacturer and one devicetype.</p> <p><b>Note:</b></p> <p>In order to present a worldwide unique device identification, a Frame Application has to combine IdManufacturer, IdTypeID and IdSerialNumber</p>
IdValue	Contains name value pair for one identification parameter without semantic information for the Frame Application..
IdValues	List of identification elements, which are not specifically defined by Idxxx elements listed above. No further protocol independent semantic information is available for those identification elements.
ScanIdentification	Contains all identification elements for one single scanned device or block. This includes elements with well defined semantic as well as IdValues.
ScanIdentifications	List of identifications for several scanned physical devices or blocks.

```

<Schema name="DTMScanIdentSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:ident="x-schema:DTMIdentSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="resultState" dt:type="enumeration" dt:values="provisional final error"/>
  <AttributeType name="configuredState" dt:type="enumeration" dt:values="configuredAndPhysicallyAvailable
configuredAndNotPhysicallyAvailable availableButNotConfigured notApplicable"/>
  <ElementType name="IdAddress" content="empty" model="closed">
    <attribute type="ident:value" required="yes"/>
    <attribute type="ident:protocolSpecificName" required="yes"/>
  </ElementType>
  <ElementType name="IdBusProtocol" content="empty" model="closed">
    <attribute type="ident:value" required="yes"/>
    <attribute type="ident:protocolSpecificName" required="yes"/>
  </ElementType>
  <ElementType name="IdBusProtocolVersion" content="empty" model="closed">
    <attribute type="ident:value" required="no"/>
    <attribute type="ident:protocolSpecificName" required="yes"/>
  </ElementType>
  <ElementType name="IdManufacturer" content="empty" model="closed">
    <attribute type="ident:value" required="no"/>
    <attribute type="ident:protocolSpecificName" required="yes"/>
  </ElementType>
  <ElementType name="IdTypeID" content="empty" model="closed">
    <attribute type="ident:value" required="no"/>
    <attribute type="ident:protocolSpecificName" required="yes"/>
  </ElementType>
  <ElementType name="IdSoftwareRevision" content="empty" model="closed">
    <attribute type="ident:value" required="no"/>
    <attribute type="ident:protocolSpecificName" required="yes"/>
  </ElementType>
  <ElementType name="IdDeviceTag" content="empty" model="closed">
    <attribute type="ident:value" required="no"/>
    <attribute type="ident:protocolSpecificName" required="yes"/>
  </ElementType>
  <ElementType name="IdHardwareRevision" content="empty" model="closed">
    <attribute type="ident:value" required="no"/>
    <attribute type="ident:protocolSpecificName" required="yes"/>
  </ElementType>
  <ElementType name="IdSerialNumber" content="empty" model="closed">
    <attribute type="ident:value" required="no"/>
    <attribute type="ident:protocolSpecificName" required="yes"/>
  </ElementType>
  <ElementType name="IdValue" content="empty" model="closed">
    <attribute type="ident:name" required="yes"/>
    <attribute type="ident:value" required="yes"/>
    <attribute type="ident:protocolSpecificName" required="yes"/>
  </ElementType>
  <ElementType name="IdValues" content="eltOnly" model="closed">
    <element type="IdValue" minOccurs="0" maxOccurs="*/>
  </ElementType>
  <ElementType name="ScanIdentification" content="eltOnly" model="closed">
    <attribute type="configuredState" required="no"/>
    <element type="fdt:CommunicationError" minOccurs="0" maxOccurs="1"/>
    <element type="IdBusProtocol" minOccurs="1" maxOccurs="1"/>
    <element type="IdBusProtocolVersion" minOccurs="1" maxOccurs="1"/>
    <element type="IdAddress" minOccurs="1" maxOccurs="1"/>
    <element type="IdManufacturer" minOccurs="1" maxOccurs="1"/>
    <element type="IdTypeID" minOccurs="1" maxOccurs="1"/>
    <element type="IdSoftwareRevision" minOccurs="1" maxOccurs="1"/>
    <element type="IdHardwareRevision" minOccurs="1" maxOccurs="1"/>
    <element type="IdDeviceTag" minOccurs="1" maxOccurs="1"/>
    <element type="IdSerialNumber" minOccurs="1" maxOccurs="1"/>
    <element type="IdValues" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="ScanIdentifications" content="eltOnly" model="closed">
    <attribute type="fdt:busCategory" required="yes"/>
    <attribute type="resultState" required="yes"/>
    <element type="ScanIdentification" minOccurs="0" maxOccurs="*/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <element type="ScanIdentifications" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>

```

## 12.34 *DTMDeviceTypeIdentSchema*

Used to validate the transformed response of: [IDTMInformation2::GetDeviceIdentificationInformation\(\)](#).

The XML document contains DTMDeviceType identification information of one DTMDeviceType. The document is validated by Frame Application after XSLT transformation into an protocol independent format.

Tag	Description
IdBusProtocol	See corresponding description in <a href="#">DTMScanIdentSchema.xml</a>
IdBusProtocolVersion	
IdManufacturer	
IdTypeID	
IdSoftwareRevision	
IdHardwareRevision	
IdDTMSupportLevel	This element is used only in <a href="#">DTMDeviceTypeIdentSchema</a> and not in <a href="#">DTMScanIdentSchema</a> . It cannot be used for matching. It can be displayed by Frame Applications applications for an assignment decision by user.
IdValue	Contains name value pair for one identification parameter without semantic information for the Frame Application.
IdValues	List of identification elements, which are not specifically defined by Idxxx elements listed above. No further protocol independent semantic information is available for those identification elements.
DeviceTypeIdentification	Contains all identification elements of a device type for a physical device type or group.
DeviceTypeIdentifications	List of identifications of several device types for physical device types. The DeviceTypeIdentifications element should not be used in context of XML returned by <a href="#">IDtmInformation2::GetDeviceIdentificationInformation()</a> . The element should only be used in context of XMLs which could contain identification information for several device types (i.e. <a href="#">IDtmChannelSubTopology2</a> , <a href="#">IDtmHardwareIdentification</a> ),

```
<Schema name="DTMDeviceTypeIdentSchema.xml" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:ident="x-schema:DTMIdentSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <ElementType name="IdBusProtocol" content="eltOnly" model="closed">
    <attribute type="ident:value" required="no"/>
    <attribute type="ident:protocolSpecificName" required="yes"/>
    <element type="ident:RegExpr" minOccurs="0" maxOccurs="*" />
  </ElementType>
  <ElementType name="IdBusProtocolVersion" content="eltOnly" model="closed">
    <attribute type="ident:value" required="no"/>
    <attribute type="ident:protocolSpecificName" required="yes"/>
    <element type="ident:RegExpr" minOccurs="0" maxOccurs="*" />
  </ElementType>
  <ElementType name="IdManufacturer" content="eltOnly" model="closed">
    <attribute type="ident:value" required="no"/>
    <attribute type="ident:protocolSpecificName" required="yes"/>
    <element type="ident:RegExpr" minOccurs="0" maxOccurs="*" />
  </ElementType>
```

```

<ElementType name="IdTypeID" content="eltOnly" model="closed">
  <attribute type="ident:value" required="no"/>
  <attribute type="ident:protocolSpecificName" required="yes"/>
  <element type="ident:RegExpr" minOccurs="0" maxOccurs="**"/>
</ElementType>
<ElementType name="IdSoftwareRevision" content="eltOnly" model="closed">
  <attribute type="ident:value" required="no"/>
  <attribute type="ident:protocolSpecificName" required="yes"/>
  <element type="ident:RegExpr" minOccurs="0" maxOccurs="**"/>
</ElementType>
<ElementType name="IdHardwareRevision" content="eltOnly" model="closed">
  <attribute type="ident:value" required="no"/>
  <attribute type="ident:protocolSpecificName" required="yes"/>
  <element type="ident:RegExpr" minOccurs="0" maxOccurs="**"/>
</ElementType>
<ElementType name="IdDTMSupportLevel" content="eltOnly" model="closed">
  <attribute type="ident:value" required="no"/>
  <attribute type="ident:protocolSpecificName" required="yes"/>
  <element type="ident:RegExpr" minOccurs="0" maxOccurs="**"/>
</ElementType>
<ElementType name="IdValue" content="eltOnly" model="closed">
  <attribute type="ident:name" required="yes"/>
  <attribute type="ident:value" required="no"/>
  <attribute type="ident:protocolSpecificName" required="yes"/>
  <element type="ident:RegExpr" minOccurs="0" maxOccurs="**"/>
</ElementType>
<ElementType name="IdValues" content="eltOnly" model="closed">
  <element type="IdValue" minOccurs="0" maxOccurs="**"/>
</ElementType>
<ElementType name="DeviceIdentification" content="eltOnly" model="closed">
  <attribute type="ident:idDTMSupportLevel" required="yes"/>
  <element type="IdBusProtocol" minOccurs="1" maxOccurs="1"/>
  <element type="IdBusProtocolVersion" minOccurs="1" maxOccurs="1"/>
  <element type="IdManufacturer" minOccurs="1" maxOccurs="1"/>
  <element type="IdTypeID" minOccurs="1" maxOccurs="1"/>
  <element type="IdSoftwareRevision" minOccurs="1" maxOccurs="1"/>
  <element type="IdHardwareRevision" minOccurs="1" maxOccurs="1"/>
  <element type="IdValues" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="DeviceIdentifications" content="eltOnly" model="closed">
  <element type="DeviceIdentification" minOccurs="1" maxOccurs="**"/>
</ElementType>
<ElementType name="FDT" content="eltOnly" model="closed">
  <element type="DeviceIdentifications" minOccurs="1" maxOccurs="1"/>
</ElementType>
</Schema>

```

### Example:

#### HART Example after XSLT transformation:

```

<?xml version="1.0"?>
<!-- This file is created by FDTxxxIdentTransformation.xsl after transformation of xxxDTMDeviceIdentificationInstance.xml -->
<FDT xmlns="x-schema:DTMDeviceTypeIdentSchema.xml" xmlns:ident="x-schema:DTMIdentSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
  <DeviceIdentifications>
    <DeviceIdentification ident:idDTMSupportLevel="specificSupport" xmlns:ident="x-schema:DTMIdentSchema.xml">
      <IdBusProtocol ident:value="HART" ident:protocolSpecificName="HART"/>
      <IdBusProtocolVersion ident:value="5" ident:protocolSpecificName="HART Revision"/>
      <IdManufacturer ident:value="17" ident:protocolSpecificName="Manufacturer Identification Code"/>
      <IdTypeID ident:value="200" ident:protocolSpecificName="Device Type Code"/>
      <IdSoftwareRevision ident:value="11" ident:protocolSpecificName="Software Revision"/>
      <IdHardwareRevision ident:protocolSpecificName="Hardware Revision">
        <ident:RegExpr match="[123].[0-9]"/>
      </IdHardwareRevision>
      <IdValues>
        <IdValue ident:name="DeviceCommandRevisionLevel" ident:value="2" ident:protocolSpecificName="Device
Revision Level"/>
        <IdValue ident:name="DeviceFlag" ident:value="0" ident:protocolSpecificName="Flags"/>
      </IdValues>
    </DeviceIdentification>
  </DeviceIdentifications>
</FDT>

```



## 12.35 *DTMItemListSchema*

Used in methods related to SingleDataAccess interfaces.

Attribute	Description																																								
moduleName	This attribute contains the name of the module																																								
itemErrorDescription	Enumeration describing an error: <ul style="list-style-type: none"> <li>• dtmSpecific: DTM specific error</li> <li>• noLock: instance data set could not be locked</li> <li>• notLongerValid: the item is not longer valid. This may happen due to configuration changes</li> <li>• outOfResources: The DTM has no resources to perform the request. This may happen if the DTM can not queue the request.</li> <li>• invalidValue: The requested value is invalid</li> </ul>																																								
itemType	Information whether the item follows the semantics defined via the general rules defined for a specific protocol (standard) or a DTM/vendor specific semantics (specific)																																								
label	Human readable name																																								
qualityBits	Quality status of the item (In addition refer OPC XML-DA Specification Version 1.0). If not available 'qualityBits' must be set to 'good'.																																								
limitBits	Limit status of the item (In addition refer OPC XML-DA Specification Version 1.0). If not available 'limitBits' must be set to 'none'.																																								
itemKind	<p>Identification of the item context. This is some kind of classification regarding the type of value</p> <p>Enumerations:</p> <table border="1"> <thead> <tr> <th>Entry</th><th>Specifies that item</th></tr> </thead> <tbody> <tr> <td>alarm</td><td>contains alarm limits</td></tr> <tr> <td>analogInput</td><td>is part of an analog input block</td></tr> <tr> <td>analogOutput</td><td>is part of an analog output block</td></tr> <tr> <td>computation</td><td>is part of a computation block</td></tr> <tr> <td>contained</td><td>represents the physical characteristics of the device</td></tr> <tr> <td>correction</td><td>is part of the correction block</td></tr> <tr> <td>device</td><td>represents the physical characteristics of the device</td></tr> <tr> <td>diagnostic</td><td>indicates the device status</td></tr> <tr> <td>digitalInput</td><td>is part of a digital input block</td></tr> <tr> <td>digitalOutput</td><td>is part of a digital output block</td></tr> <tr> <td>discrete</td><td>A discrete block contains the variables associated with a discrete output. It is not uncommon for there to be several discrete blocks.</td></tr> <tr> <td>discreteInput</td><td>is part of a discrete input block</td></tr> <tr> <td>discreteOutput</td><td>is part of a discrete output block</td></tr> <tr> <td>dynamic</td><td>is modified by the device without stimulus from the network</td></tr> <tr> <td>frequency</td><td>A frequency block contains the variables associated with a frequency output. There is typically at most one frequency block.</td></tr> <tr> <td>frequencyInput</td><td>is part of a frequency input block</td></tr> <tr> <td>frequencyOutput</td><td>is part of a frequency output block</td></tr> <tr> <td>hart</td><td>The HART block contains the variables associated with the HART interface. There is only one HART block.</td></tr> <tr> <td>Input</td><td>is part of an input block. An input block is a special kind of computation block which does unit conversions, scaling, and damping. The parameter of the input block parameters can be</td></tr> </tbody> </table>	Entry	Specifies that item	alarm	contains alarm limits	analogInput	is part of an analog input block	analogOutput	is part of an analog output block	computation	is part of a computation block	contained	represents the physical characteristics of the device	correction	is part of the correction block	device	represents the physical characteristics of the device	diagnostic	indicates the device status	digitalInput	is part of a digital input block	digitalOutput	is part of a digital output block	discrete	A discrete block contains the variables associated with a discrete output. It is not uncommon for there to be several discrete blocks.	discreteInput	is part of a discrete input block	discreteOutput	is part of a discrete output block	dynamic	is modified by the device without stimulus from the network	frequency	A frequency block contains the variables associated with a frequency output. There is typically at most one frequency block.	frequencyInput	is part of a frequency input block	frequencyOutput	is part of a frequency output block	hart	The HART block contains the variables associated with the HART interface. There is only one HART block.	Input	is part of an input block. An input block is a special kind of computation block which does unit conversions, scaling, and damping. The parameter of the input block parameters can be
Entry	Specifies that item																																								
alarm	contains alarm limits																																								
analogInput	is part of an analog input block																																								
analogOutput	is part of an analog output block																																								
computation	is part of a computation block																																								
contained	represents the physical characteristics of the device																																								
correction	is part of the correction block																																								
device	represents the physical characteristics of the device																																								
diagnostic	indicates the device status																																								
digitalInput	is part of a digital input block																																								
digitalOutput	is part of a digital output block																																								
discrete	A discrete block contains the variables associated with a discrete output. It is not uncommon for there to be several discrete blocks.																																								
discreteInput	is part of a discrete input block																																								
discreteOutput	is part of a discrete output block																																								
dynamic	is modified by the device without stimulus from the network																																								
frequency	A frequency block contains the variables associated with a frequency output. There is typically at most one frequency block.																																								
frequencyInput	is part of a frequency input block																																								
frequencyOutput	is part of a frequency output block																																								
hart	The HART block contains the variables associated with the HART interface. There is only one HART block.																																								
Input	is part of an input block. An input block is a special kind of computation block which does unit conversions, scaling, and damping. The parameter of the input block parameters can be																																								

Attribute	Description
	determined by the output of another block
local	is locally used by the an application. Local items are not stored in a device, but they can be sent to a device
localDisplay	is part of the local display block. A local display block contains the items associated with the local interface (keyboard, display, etc.) of the device
operate	is used to control a block's operation
output	is part of the output block. The values of output items may be accessed by another block input
sensorCorrection	A sensor correction block contains the variables used to correct the value returned by a sensor. There is a distinct sensor correction block for each sensor.
service	is used when performing routine maintenance
tune	is used to tune the algorithm of a block
others	Is used if all other entries do not match

Tag	Description
DtmlItemInfo	Describes for example a parameter or a process value that is available via the single data access interface. The information contains descriptive attributes like name as well as information how the item is accessible. The relation between item information and the item itself is realized via a unique id
DtmlItem	Contains for example the value of a parameter or a process value and some additional optional information like time stamp. The time information is provided based on UTC In case of an write requests without values the item carries no value, means fdt:Variant do not exist
DtmlItemInfoList	Containing a list of DTM item information and/or a list of item information groups
DtmlItemInfoGroup	Containing a list of DTM item information
DtmlItemList	Containing a list of DTM items
itemError	Containing a communication or non communication error
ItemErrorDescription	Description of non communication error
UnitDescription	Description of the unit of the item
ValueDescription	Description the value of the item
RangeDescriptions	Description of the ranges of the item. Range descriptions must be provided if available.
RangeDescription	Description of an range
UpperRangeDescription	Description of the upper range
LowerRangeDescription	Description of the lower range
ItemReference	Reference to an other item within the xml document
PossibleEnumerations	Possible enumerations of an item
Quality	Description of the quality of the item. For write requests: The Frame Application may define a Quality.  <u>In case the underlying device/parameter supports the quality definitions, the value+quality should be handed over by DTM to the device, otherwise the DTM should only process values with good quality.</u>

```

<Schema name="DTMItemListSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="moduleName" dt:type="string"/>
  <AttributeType name="itemErrorDescription" dt:type="enumeration" dt:values="dtmSpecific noLock notLongerValid outOfResources invalidValue"/>
  <AttributeType name="itemType" dt:type="enumeration" dt:values="standard specific"/>

```

```

<AttributeType name="itemKind" dt:type="enumeration" dt:values="alarm analogInput analogOutput computation
contained correction device diagnostic digitalInput digitalOutput discrete discreteInput discreteOutput dynamic frequency
frequencyInput frequencyOutput hart input local localDisplay operate output sensorCorrection service tune others"/>
<AttributeType name="label" dt:type="string"/>
<AttributeType name="qualityBits" dt:type="enumeration" dt:values="bad badConfigurationError badNotConnected
badDeviceFailure badSensorFailure badLastKnownValue badCommFailure badOutOfService badWaitingForInitialData uncertain
uncertainLastUsableValue uncertainSensorNotAccurate uncertainEUExceeded uncertainSubNormal good goodLocalOverride"/>
<AttributeType name="limitBits" dt:type="enumeration" dt:values="none low high constant"/>
<!--Definition of Elements-->
<ElementType name="ItemKind" content="empty" model="closed" order="seq">
  <attribute type="fdt:nodeId" required="no"/>
  <attribute type="itemKind" required="yes"/>
</ElementType>
<ElementType name="Quality" content="empty" model="closed" order="seq">
  <attribute type="fdt:nodeId" required="no"/>
  <attribute type="qualityBits" required="yes"/>
  <attribute type="limitBits" required="no"/>
</ElementType>
<ElementType name="TimeStamp" content="empty" model="closed" order="seq">
  <attribute type="fdt:nodeId" required="no"/>
  <attribute type="fdt:time" required="yes"/>
</ElementType>
<ElementType name="ItemErrorDescription" content="empty" model="closed" order="seq">
  <attribute type="fdt:nodeId" required="no"/>
  <attribute type="itemErrorDescription" required="yes"/>
  <attribute type="fdt:descriptor" required="no"/>
</ElementType>
<ElementType name="ItemError" content="eltOnly" model="closed" order="seq">
  <attribute type="fdt:nodeId" required="no"/>
  <group order="one" minOccurs="0" maxOccurs="1">
    <element type="ItemErrorDescription"/>
    <element type="fdt:CommunicationError"/>
  </group>
</ElementType>
<ElementType name="PossibleEnumerations" content="eltOnly" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <group order="one" minOccurs="1" maxOccurs="1">
    <element type="fdt:EnumeratorEntries" minOccurs="0" maxOccurs="1"/>
    <element type="fdt:BitEnumeratorEntries" minOccurs="0" maxOccurs="1"/>
  </group>
</ElementType>
<ElementType name="ItemReference" content="empty" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <attribute type="fdt:idref" required="no"/>
</ElementType>
<ElementType name="LowerRangeDescription" content="eltOnly" model="closed" order="seq">
  <attribute type="fdt:nodeId" required="no"/>
  <group order="one" minOccurs="0" maxOccurs="1">
    <element type="ItemReference"/>
    <element type="fdt:StringData"/>
    <element type="fdt:NumberData"/>
    <element type="fdt:TimeData"/>
  </group>
</ElementType>
<ElementType name="UpperRangeDescription" content="eltOnly" model="closed" order="seq">
  <attribute type="fdt:nodeId" required="no"/>
  <group order="one" minOccurs="0" maxOccurs="1">
    <element type="ItemReference"/>
    <element type="fdt:StringData"/>
    <element type="fdt:NumberData"/>
    <element type="fdt:TimeData"/>
  </group>
</ElementType>
<ElementType name="RangeDescription" content="eltOnly" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <element type="LowerRangeDescription" minOccurs="1" maxOccurs="1"/>
  <element type="UpperRangeDescription" minOccurs="1" maxOccurs="1"/>
  <element type="fdt:LowerRawValue" minOccurs="0" maxOccurs="1"/>
  <element type="fdt:UpperRawValue" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="RangeDescriptions" content="eltOnly" model="closed">
  <attribute type="fdt:nodeId" required="no"/>
  <element type="RangeDescription" minOccurs="1" maxOccurs="*/>
</ElementType>
<ElementType name="ValueDescription" content="eltOnly" model="closed" order="seq">
  <attribute type="fdt:nodeId" required="no"/>

```

```

        <element type="PossibleEnumerations" minOccurs="1" maxOccurs="1"/>
    </ElementType>
    <ElementType name="UnitDescription" content="eltOnly" model="closed" order="seq">
        <attribute type="fdt:nodeId" required="no"/>
        <group order="one" minOccurs="0" maxOccurs="1">
            <element type="ItemReference" minOccurs="1" maxOccurs="1"/>
            <element type="fdt:EnumeratorEntry" minOccurs="1" maxOccurs="1"/>
        </group>
    </ElementType>
    <ElementType name="DtmlItemInfo" content="eltOnly" model="closed" order="seq">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:id" required="yes"/>
        <element type="fdt:SemanticInformation" minOccurs="0" maxOccurs="1"/>
        <attribute type="fdt:name" required="yes"/>
        <attribute type="fdt:dataType" required="yes"/>
        <attribute type="itemType" required="yes"/>
        <attribute type="fdt:descriptor" required="no"/>
        <attribute type="moduleName" required="no"/>
        <attribute type="fdt:readAccess" required="no"/>
        <attribute type="fdt:writeAccess" required="no"/>
        <attribute type="label" required="no"/>
        <element type="ItemKind" minOccurs="1" maxOccurs="1"/>
        <element type="UnitDescription" minOccurs="0" maxOccurs="1"/>
        <group order="one" minOccurs="0" maxOccurs="1">
            <element type="RangeDescriptions" minOccurs="0" maxOccurs="1"/>
            <element type="ValueDescription" minOccurs="0" maxOccurs="1"/>
        </group>
    </ElementType>
    <ElementType name="DtmlItemInfoGroup" content="eltOnly" model="closed" order="seq">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:name" required="yes"/>
        <element type="fdt:SemanticInformation" minOccurs="0" maxOccurs="1"/>
        <attribute type="label" required="no"/>
        <element type="DtmlItemInfo" minOccurs="0" maxOccurs="1"/>
        <element type="DtmlItemInfoGroup" minOccurs="0" maxOccurs="1"/>
    </ElementType>
    <ElementType name="DtmlItemSelection" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:id" required="yes"/>
    </ElementType>
    <ElementType name="DtmlItem" content="eltOnly" model="closed" order="seq">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:id" required="yes"/>
        <element type="TimeStamp" minOccurs="1" maxOccurs="1"/>
        <element type="Quality" minOccurs="1" maxOccurs="1"/>
        <group order="one" minOccurs="1" maxOccurs="1">
            <element type="fdt:Variant"/>
            <element type="ItemError"/>
        </group>
    </ElementType>
    <ElementType name="DtmlItemInfoList" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="DtmlItemInfo" minOccurs="0" maxOccurs="1"/>
        <element type="DtmlItemInfoGroup" minOccurs="0" maxOccurs="1"/>
    </ElementType>
    <ElementType name="DtmlItemList" content="eltOnly" model="closed" order="seq">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="DtmlItem" minOccurs="1" maxOccurs="1"/>
    </ElementType>
    <ElementType name="DtmlItemSelectionList" content="eltOnly" model="closed" order="seq">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="DtmlItemSelection" minOccurs="1" maxOccurs="1"/>
    </ElementType>
    <ElementType name="FDT" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <group order="one" minOccurs="1" maxOccurs="1">
            <element type="DtmlItemInfoList"/>
            <element type="DtmlItemList"/>
            <element type="DtmlItemSelectionList"/>
        </group>
    </ElementType>
</Schema>

```

Examples:

Info regarding the exposed items:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMItemListSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <DTMItemInfoList>
    <DTMItemInfo fdt:id="1" fdt:name="PV" fdt:dataType="float" itemType="standard">
      <fdt:SemanticInformation semanticId="CMD3B5B0L32" applicationDomain="FDT_HART"/>
      <ItemKind itemKind="analogInput"/>
      <UnitDescription>
        <ItemReference fdt:idref="2"/>
      </UnitDescription>
      <RangeDescriptions>
        <RangeDescription>
          <LowerRangeDescription>
            <ItemReference fdt:idref="4"/>
          </LowerRangeDescription>
          <UpperRangeDescription>
            <ItemReference fdt:idref="3"/>
          </UpperRangeDescription>
        </RangeDescription>
      </RangeDescriptions>
    </DTMItemInfo>
    <DTMItemInfo fdt:id="2" fdt:name="PV_UNITS" fdt:dataType="byte" itemType="standard">
      <fdt:SemanticInformation semanticId="CMD3B4B0L8" applicationDomain="FDT_HART"/>
      <ItemKind itemKind="contained"/>
      <ValueDescription>
        <PossibleEnumerations>
          <fdt:EnumeratorEntries>
            <fdt:EnumeratorEntry index="1" name="inH2O"/>
            <fdt:EnumeratorEntry index="2" name="inHg"/>
            <fdt:EnumeratorEntry index="3" name="ftH2O"/>
            <fdt:EnumeratorEntry index="4" name="mmH2O"/>
            <fdt:EnumeratorEntry index="5" name="mmHg"/>
            <fdt:EnumeratorEntry index="6" name="psi"/>
            <fdt:EnumeratorEntry index="7" name="bar"/>
            <fdt:EnumeratorEntry index="8" name="mbar"/>
            <fdt:EnumeratorEntry index="9" name="g/Sqcm"/>
            <fdt:EnumeratorEntry index="10" name="kg/Sqcm"/>
            <fdt:EnumeratorEntry index="11" name="Pa"/>
            <fdt:EnumeratorEntry index="12" name="kPa"/>
            <fdt:EnumeratorEntry index="13" name="torr"/>
            <fdt:EnumeratorEntry index="14" name="atm"/>
          </fdt:EnumeratorEntries>
        </PossibleEnumerations>
      </ValueDescription>
    </DTMItemInfo>
    <DTMItemInfo fdt:id="3" fdt:name="PV_UPPER_RANGE_VALUE" fdt:dataType="float" itemType="standard">
      <fdt:SemanticInformation semanticId="CMD15B3B0L32" applicationDomain="FDT_HART"/>
      <ItemKind itemKind="contained"/>
      <UnitDescription>
        <ItemReference fdt:idref="5"/>
      </UnitDescription>
    </DTMItemInfo>
    <DTMItemInfo fdt:id="4" fdt:name="PV_LOWER_RANGE_VALUE" fdt:dataType="float" itemType="standard">
      <fdt:SemanticInformation semanticId="CMD15B7B0L32" applicationDomain="FDT_HART"/>
      <ItemKind itemKind="contained"/>
      <UnitDescription>
        <ItemReference fdt:idref="5"/>
      </UnitDescription>
    </DTMItemInfo>
    <DTMItemInfo fdt:id="5" fdt:name="PV_RANGE_VALUES_UNITS_CODE" fdt:dataType="byte" itemType="standard">
      <fdt:SemanticInformation semanticId="CMD15B2B0L8" applicationDomain="FDT_HART"/>
      <ItemKind itemKind="contained"/>
      <ValueDescription>
        <PossibleEnumerations>
          <fdt:EnumeratorEntries>
            <fdt:EnumeratorEntry index="1" name="inH2O"/>
            <fdt:EnumeratorEntry index="2" name="inHg"/>
            <fdt:EnumeratorEntry index="3" name="ftH2O"/>
            <fdt:EnumeratorEntry index="4" name="mmH2O"/>
            <fdt:EnumeratorEntry index="5" name="mmHg"/>
            <fdt:EnumeratorEntry index="6" name="psi"/>
            <fdt:EnumeratorEntry index="7" name="bar"/>
            <fdt:EnumeratorEntry index="8" name="mbar"/>
          </fdt:EnumeratorEntries>
        </PossibleEnumerations>
      </ValueDescription>
    </DTMItemInfo>
  </DTMItemInfoList>
</FDT>
```

```

<fdt:EnumeratorEntry index="9" name="g/Sqcm"/>
<fdt:EnumeratorEntry index="10" name="kg/Sqcm"/>
<fdt:EnumeratorEntry index="11" name="Pa"/>
<fdt:EnumeratorEntry index="12" name="kPa"/>
<fdt:EnumeratorEntry index="13" name="torr"/>
<fdt:EnumeratorEntry index="14" name="atm"/>
</fdt:EnumeratorEntries>
</PossibleEnumerations>
</ValueDescription>
</DtmlItemInfo>
</DtmlItemInfoList>
</FDT>

```

Selection of items:

```

<?xml version="1.0"?>
<!--Selection of PV and PV_UNITS-->
<FDT xmlns="x-schema:DTMItemListSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <DtmlItemSelectionList>
    <DtmlItemSelection fdt:id="1"/>
    <DtmlItemSelection fdt:id="2"/>
  </DtmlItemSelectionList>
</FDT>

```

Result:

```

<?xml version="1.0"?>
<!--Result: 10.1 mbar-->
<FDT xmlns="x-schema:DTMItemListSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <DtmlItemList>
    <DtmlItem fdt:id="1">
      <TimeStamp fdt:time="2005-12-01T18:01:01"/>
      <Quality qualityBits="good"/>
      <fdt:Variant dataType="float">
        <fdt:NumberData number="10.1"/>
      </fdt:Variant>
    </DtmlItem>
    <DtmlItem fdt:id="2">
      <TimeStamp fdt:time="2005-12-01T18:01:02"/>
      <Quality qualityBits="good"/>
      <fdt:Variant dataType="byte">
        <fdt:NumberData number="8"/>
      </fdt:Variant>
    </DtmlItem>
  </DtmlItemList>
</FDT>

```

## 12.36 *BtmDataTypesSchema*

The BTM data type schema is used as a global BTM definition. Data types of this schema are referenced via the prefix btm: within the other schemas.

Attribute	Description
blockCategory	Unique identifier for a block type like Analog Input, Resource Block.
blockCategoryName	Human readable block type name.
blockTag	Block Tag is a unique identifier for the block.
Index	An integer value that provides the offset from the beginning of the block

Attribute	Description
	or from the beginning of the structure
Label	Readable name. If the optional attribute 'label' exists, the attribute 'name' contains a language independent identifier
Profile	Number assigned by the Fieldbus Foundation that uniquely identifies the profile on which the block is based.
profileRevision	The revision number of the profile on which the block definition is based.
Slot	An integer provides parameter-grouping number. In Profibus, it can provide the slot number (if the device support slots). In Foundation fieldbus, it can provide the Application VFD number if the instrument supports more than one application VFD.
Usage	An indication of whether this object may be set by another function block (I), provides a value to another function block (O), or only provides communications support (C) to devices other than function blocks - of type Visible String, 1 Octet.

Tag	Description
BlockIcon	Icon for a Block
BlockTypeCategory	Type of the Block
BtmBlockType	Description of a block type
BtmVariable	Block variable description with name, index, value, range, etc
BtmVariables	Collection of BTM variables
FDT	Root Tag

```

<Schema name="BtmDataTypesSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
>

  <!--Definition of Attributes-->
  <AttributeType name="profile" dt:type="ui2"/>
  <AttributeType name="profileRevision" dt:type="ui2"/>
  <AttributeType name="blockTag" dt:type="string"/>
  <AttributeType name="blockCategory" dt:type="uuid"/>
  <AttributeType name="blockCategoryName" dt:type="string"/>
  <AttributeType name="usage" dt:type="enumeration" dt:values="contained input output eventSource eventSink"/>
  <AttributeType name="index" dt:type="ui4"/>
  <AttributeType name="slot" dt:type="ui4"/>
  <AttributeType name="label" dt:type="string" />

  <!--Definition of Elements-->
  <ElementType name="BtmVariable" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:name" required="yes"/>
    <attribute type="label" required="yes"/>
    <attribute type="index" required="yes"/>
    <attribute type="slot" required="no"/>
    <attribute type="fdt:descriptor" required="no"/>
    <attribute type="usage" required="yes"/>
    <element type="fdt:Value" minOccurs="1" maxOccurs="1"/>
    <element type="fdt:Unit" minOccurs="0" maxOccurs="1"/>
    <element type="fdt:Ranges" minOccurs="0" maxOccurs="1"/>
    <attribute type="fdt:statusFlag" required="no"/>
    <element type="fdt:StatusInformation" minOccurs="0" maxOccurs="1"/>
  </ElementType>

  <ElementType name="BtmVariables" content="eltOnly" model="closed" order="seq">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:name" required="yes"/>
    <attribute type="label" required="yes"/>
    <attribute type="index" required="yes"/>
    <group order="many">

```

```

        <element type="BtmVariables" minOccurs="0" maxOccurs="*" />
        <element type="BtmVariable" minOccurs="0" maxOccurs="*" />
    </group>
</ElementType>

<ElementType name="BlockIcon" content="empty" model="closed">
    <attribute type="fdt:nodId" required="no" />
    <attribute type="fdt:path" required="yes" />
</ElementType>

<ElementType name="BlockTypeCategory" content="empty" model="closed">
    <attribute type="fdt:nodId" required="no" />
    <attribute type="blockCategory" required="yes" />
    <attribute type="blockCategoryName" required="no" />
</ElementType>

<ElementType name="BtmBlockType" content="eltOnly" model="closed">
    <attribute type="fdt:nodId" required="no" />
    <attribute type="fdt:readAccess" required="no" />
    <attribute type="fdt:writeAccess" required="no" />
    <attribute type="fdt:manufacturerId" required="no" />
    <attribute type="profile" required="no" />
    <attribute type="profileRevision" required="no" />
    <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1" />
    <element type="fdt:SupportedLanguages" minOccurs="0" maxOccurs="1" />
    <element type="BlockTypeCategory" minOccurs="0" maxOccurs="1" />
    <element type="fdt:BusCategories" minOccurs="1" maxOccurs="1" />
    <element type="BlockIcon" minOccurs="0" maxOccurs="1" />
</ElementType>
</Schema>

```

## 12.37 BtmInformationSchema

Used at: [IBtmInformation::GetInformation\(\)](#)

The XML document provides BTM specific information.

Tag	Description
BtmBlockTypes	Collection of block types
BtmInfo	Describes the BTM itself
FDT	Root Tag

```

<Schema name="BtmInformationSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
  xmlns:dtminfo="x-schema:DTMInformationSchema.xml"
  xmlns:btm="x-schema:BtmDataTypesSchema.xml"
>

  <!--Definition of Elements-->

  <ElementType name="BtmBlockTypes" content="eltOnly" model="closed">
    <attribute type="fdt:nodId" required="no" />
    <element type="btm:BtmBlockType" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="BtmInfo" content="eltOnly" model="closed" order="seq">
    <attribute type="fdt:nodId" required="no" />
    <element type="dtminfo:FDTVersion" minOccurs="1" maxOccurs="1" />
    <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1" />
    <element type="BtmBlockTypes" minOccurs="1" maxOccurs="*" />
  </ElementType>

```



```

<ElementType name="FDT" content="eltOnly" model="closed">
  <attribute type="fdt:nodetd" required="no"/>
  <element type="BtmInfo" minOccurs="1" maxOccurs="1"/>
</ElementType>
</Schema>

```

Example:

```

<?xml version='1.0'?>
<FDT
  xmlns="x-schema:BtmInformationSchema.xml"
  xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
  xmlns:dtminfo="x-schema:DTMInformationSchema.xml"
  xmlns:btm="x-schema:BtmDataTypesSchema.xml"
>
  <BtmInfo>
    <dtminfo:FDTVersion major="1" minor="2"/>
    <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2002-06-20"/>
    <BtmBlockTypes>
      <btm:BtmBlockType profile="257" profileRevision="1">
        <fdt:VersionInformation name="AI" vendor="myVendor" version="1.0" date="2002-06-20"/>
        <fdt:SupportedLanguages>
          <fdt:LanguageId languageId="1131"/>
        </fdt:SupportedLanguages>
        <btm:BlockTypeCategory blockCategory="D8AF8561-F174-451d-AE87-E467CBE7907A"
blockCategoryName="Analog Input"/>
        <fdt:BusCategories>
          <fdt:BusCategory busCategory="036D1693-387B-11D4-86E1-00E0987270B9"/>
        </fdt:BusCategories>
      </btm:BtmBlockType>
    </BtmBlockTypes>
  </BtmInfo>
</FDT>

```

## 12.38 BtmParameterSchema

Used at: [IBtmParameter::GetParameters\(\)](#)  
[IBtmParameter::SetParameters\(\)](#)

The XML document provides all instance-specific information about the block.

Tag	Description
BtmBlock	Description of the Block instance.
BtmExportedVariables	Collection of BTM variables for common access. This means that the Frame Application, DTMs or other BTMs are allowed access to the data within this section
FDT	Root Tag

```

<Schema name="BtmParameterSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
  xmlns:btm="x-schema:BtmDataTypesSchema.xml"
>

  <ElementType name="BtmExportedVariables" content="eltOnly" model="closed">
    <attribute type="fdt:nodetd" required="no"/>
    <element type="btm:BtmVariables" minOccurs="1" maxOccurs="1"/>
  </ElementType>

```

```

<!--Definition of Elements-->
<ElementType name="BtmBlock" content="eltOnly" model="closed" order="seq">
  <attribute type="fdt:nodeld" required="no"/>
  <attribute type="fdt:readAccess" required="no"/>
  <attribute type="fdt:writeAccess" required="no"/>
  <attribute type="fdt:tag" required="yes"/>
  <element type="fdt:ChannelReferences" minOccurs="0" maxOccurs="1"/>
  <element type="BtmExportedVariables" minOccurs="0" maxOccurs="1"/>
</ElementType>

<ElementType name="FDT" content="eltOnly" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <attribute type="fdt:storageState" required="yes"/>
  <attribute type="fdt:dataSetState" required="yes"/>
  <element type="btm:BtmBlockType" minOccurs="1" maxOccurs="1"/>
  <element type="BtmBlock" minOccurs="1" maxOccurs="1"/>
</ElementType>
</Schema>

```

#### Example:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:BtmParameterSchema.xml"
  xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
  xmlns:btm="x-schema:BtmDataTypesSchema.xml"
  fdt:storageState="persistent" fdt:dataSetState="default">

  <btm:BtmBlockType profile="257" profileRevision="1">
    <fdt:VersionInformation name="AI" vendor="myVendor" version="1.0" date="2002-06-20"/>
    <fdt:SupportedLanguages>
      <fdt:LanguageId languageId="1131"/>
    </fdt:SupportedLanguages>
    <btm:BlockTypeCategory blockCategory="D8AF8561-F174-451d-AE87-E467CBE7907A" blockCategoryName="Analog
Input"/>
    <fdt:BusCategories>
      <fdt:BusCategory busCategory="036D1693-387B-11D4-86E1-00E0987270B9"/>
    </fdt:BusCategories>
  </btm:BtmBlockType>

  <BtmBlock fdt:tag="SomeTag">
    <BtmExportedVariables>
      <btm:BtmVariables fdt:name="FF_BLOCK" label="FF_BLOCK" index="528">
        <btm:BtmVariables fdt:name="BlockDescriptor" label="BlockDescriptor" index="528">
          <btm:BtmVariable fdt:name="BlockTag" label="BlockTag" index="528" slot="1" usage="contained">
            <fdt:Value>
              <fdt:Variant dataType="ascii" >
                <fdt:StringData string="MyBlock"/>
              </fdt:Variant>
            </fdt:Value>
          </btm:BtmVariable>
          <btm:BtmVariable fdt:name="Profile" label="Profile" index="528" slot="1" usage="contained">
            <fdt:Value>
              <fdt:Variant dataType="int" >
                <fdt:NumberData number="257"/>
              </fdt:Variant>
            </fdt:Value>
          </btm:BtmVariable>
          <btm:BtmVariable fdt:name="ProfileRevision" label="ProfileRevision" index="528" usage="contained">
            <fdt:Value>
              <fdt:Variant dataType="int" >
                <fdt:NumberData number="257"/>
              </fdt:Variant>
            </fdt:Value>
          </btm:BtmVariable>
          <btm:BtmVariable fdt:name="ExecutionTime" label="ExecutionTime" index="528" usage="contained">
            <fdt:Value>
              <fdt:Variant dataType="int" >
                <fdt:NumberData number="1280"/>
              </fdt:Variant>
            </fdt:Value>
          </btm:BtmVariable>
          <btm:BtmVariable fdt:name="PeriodOfExecution" label="PeriodOfExecution" index="528" usage="contained">

```

```

        <fdt:Value>
          <fdt:Variant dataType="int" >
            <fdt:NumberData number="1280"/>
          </fdt:Variant>
        </fdt:Value>
      </btm:BtmVariable>
    <btm:BtmVariable fdt:name="NextFBToExecute" label="NextFBToExecute" index="528" usage="contained">
      <fdt:Value>
        <fdt:Variant dataType="int" >
          <fdt:NumberData number="0"/>
        </fdt:Variant>
      </fdt:Value>
    </btm:BtmVariable>
    <btm:BtmVariable fdt:name="NumOfParms" label="NumOfParms" index="528" usage="contained">
      <fdt:Value>
        <fdt:Variant dataType="int" >
          <fdt:NumberData number="38"/>
        </fdt:Variant>
      </fdt:Value>
    </btm:BtmVariable>
  </btm:BtmVariables>
  <btm:BtmVariable fdt:name="ST_REV" label="Static Revision" index="529" usage="contained">
    <fdt:Value>
      <fdt:Variant dataType="int" >
        <fdt:NumberData number="15"/>
      </fdt:Variant>
    </fdt:Value>
  </btm:BtmVariable>
  <btm:BtmVariable fdt:name="TAG_DESC" label="Tag description" index="530" usage="contained">
    <fdt:Value>
      <fdt:Variant dataType="ascii" >
        <fdt:StringData string="MyBlock"/>
      </fdt:Variant>
    </fdt:Value>
  </btm:BtmVariable>
  <btm:BtmVariable fdt:name="And so on" label="..." index="530" usage="contained">
    <fdt:Value>
      <fdt:Variant dataType="ascii" >
        <fdt:StringData string="To be continued"/>
      </fdt:Variant>
    </fdt:Value>
  </btm:BtmVariable>
</btm:BtmVariables>
</BtmExportedVariables>
</BtmBlock>
</FDT>

```

## 12.39 BtmInitSchema

Used at: [IBtm::InitNew\(\)](#)

The XML document contains information concerning the block type.

Tag	Description
FDT	Root tag

```

<Schema name="BtmInitSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:btm="x-schema:BtmDataTypesSchema.xml"
>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <element type="btm:BtmBlockType"/>
  </ElementType>

```

```

    </ElementType>
</Schema>

```

Example:

```

<?xml version='1.0'?>
<FDT
  xmlns="x-schema:BtmInitSchema.xml"
  xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
  xmlns:btm="x-schema:BtmDataTypesSchema.xml"
>
  <btm:BtmBlockType profile="257" profileRevision="1">
    <fdt:VersionInformation name="AI" vendor="myVendor" version="1.0" date="2002-06-20"/>
    <fdt:SupportedLanguages>
      <fdt:LanguageId languageId="1131"/>
    </fdt:SupportedLanguages>
    <btm:BlockTypeCategory blockCategory="D8AF8561-F174-451d-AE87-E467CBE7907A" blockCategoryName="Analog
Input"/>
    <fdt:BusCategories>
      <fdt:BusCategory busCategory="036D1693-387B-11D4-86E1-00E0987270B9"/>
    </fdt:BusCategories>
  </btm:BtmBlockType>
</FDT>

```

## 12.40 *BtmInfoListSchema*

Tag	Description
BtmInfoList	List of BtmInfo

```

<Schema name="BtmInfoListSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:btm="x-schema:BtmInformationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Elements-->
  <ElementType name="BtmInfoList" content="eltOnly" model="closed" order="seq">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="btm:BtmInfo" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="BtmInfoList" minOccurs="1" maxOccurs="1" />
  </ElementType>
</Schema>

```

## 13 Appendix – FDT XML Styles

### 13.1 Documentation

Used at: `IDtmDocumentation::GetDocumentation()`

The XML document contains an example FDT style for documentation.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns:xslt="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/REC-html40"
  result-ns="">

  <xsl:template match="/">
    <xsl:apply-templates select="FDT"/>
    <B>2. Application Area<HR/>
    3. Operating Mode and System Structure<HR/>
    4. Input<HR/>
    5. Output<HR/>
    6. Characteristics<HR/>
    7. Operating Condition<HR/>
    8. Mechanical Design<HR/>
    9. Display and Control Interface <HR/>
    10. Auxiliary Supply<HR/>
    11. Certificates and Approvals<HR/>
    12. Ordering Information<HR/>
    13. External Standards and Technical Specifications<HR/>
    14. Device Variables</B>
    <BR/>
    <div>
      <TABLE>
        <xsl:apply-templates select="FDT/DocumentVariable"/>
      </TABLE>
      <br/>
      <dl>
        <xsl:for-each select="FDT/DocumentVariables">
          <dt> <b> <xsl:value-of select="@fdt.name"/> </b><br/></dt>
          <dd>
            <div>
              <table>
                <xsl:apply-templates select="DocumentVariable"/>
              </table>
            <xsl:apply-templates select="DocumentVariables"/>
          </div>
        </dd>
        </xsl:for-each>
      </dl>
    </div>
    <HR/>
    <xsl:apply-templates select="FDT/DTMStyleForCompleteDocument"/>
    <xsl:apply-templates select="FDT/DTMSpecificXMLData"/>
  </xsl:template>

  <xsl:template match="DocumentVariables">
    <dl>
      <dt> <b>
        <xsl:value-of select="@fdt.name"/> </b><br/></dt>
      <xsl:if test="./GraphicReference">
        <IMG>
          <xsl:attribute name="SRC">
            <xsl:value-of select="./GraphicReference/@path"/>
          </xsl:attribute>
        </IMG>
      </xsl:if>
    </dl>
  </xsl:template>

```

```

        </xsl:attribute>
        <xsl:attribute name="width">200</xsl:attribute>
    </IMG>
</xsl:if>
<dd><div>
    <table>
        <xsl:apply-templates select="DocumentVariable"/>
    </table>
    <xsl:apply-templates select="DocumentVariables"/>
</div>
</dd>
</dl>
<br/>
</xsl:template>

<xsl:template match="DocumentVariable">
    <TR>
        <TD>
            <xsl:value-of select="@fdt:name"/>
        </TD>
        <TD>&#160;</TD>
    </TR>
    <TR>
        <TD>&#160;</TD>
        <TD>
            <TABLE>
                <TR>
                    <TD>Value:</TD>
                    <TD>
                        <xsl:value-of select="@fdt:display"/>
                    </TD>
                    <TD>State:</TD>
                    <TD>
                        <xsl:value-of select="@fdt:statusFlag"/>
                    </TD>
                </TR>
                <TR>
                    <TD>
                        <BR/>
                    </TD>
                </TR>
                <TR>
                    <TD>Descriptor:</TD>
                    <TD>
                        <xsl:value-of select="@fdt:descriptor"/>
                    </TD>
                </TR>
                <xsl:if test="fdt:Unit">
                    <xsl:apply-templates select="fdt:Unit"/>
                </xsl:if>
                <xsl:if test="fdt:Ranges">
                    <xsl:apply-templates select="fdt:Ranges"/>
                </xsl:if>
            </TABLE>
        </TD>
    </TR>
    <TR>
        <TD>
            <HR/>
        </TD>
        <TD>
            <HR/>
        </TD>
    </TR>
</xsl:template>
<xsl:template match="fdt:Unit">
    <TR>
        <TD>
            <BR/>
        </TD>
    </TR>
    <TR>
        <TD>Unit:</TD>
        <TD>
            <xsl:value-of select="."/fdt:EnumeratorVariable/fdt:Variable/fdt:EnumeratorEntry/@name"/>
        </TD>
    </TR>

```

```

</TR>
<TR>
  <TD>Unit Descriptor:</TD>
  <TD>
    <xsl:value-of select="."/fdt:EnumeratorVariable/fdt:Variable/fdt:EnumeratorEntry/@descriptor"/>
  </TD>
</TR>
</xsl:template>
<xsl:template match="fdt:Ranges">
  <TR>
    <TD>
      <BR/>
    </TD>
  </TR>
  <TR>
    <TD>Ranges</TD>
  </TR>
  <TR>
    <TD>Lower Value:</TD>
    <TD>
      <xsl:value-of select="."/fdt:Range/fdt:LowerRange/fdt:NumberData/@number"/>
    </TD>
  </TR>
  <TR>
    <TD>Upper Value:</TD>
    <TD>
      <xsl:value-of select="."/fdt:Range/fdt:UpperRange/fdt:NumberData/@number"/>
    </TD>
  </TR>
</xsl:template>
<xsl:template match="FDT/DTMStyleForCompleteDocument">
  <B>15. Additional Vendor specific Documentation</B>
  <BR/>
  <xsl:value-of select="."/>
  <HR/>
</xsl:template>
<xsl:template match="FDT">
  <TABLE width="100%">
    <TR>
      <TD width="100%" valign="bottom">
        <H2>FDT Device Documentation</H2>
      </TD>
      <TD width="120">
        <IMG SRC="fdtlogo.gif" width="100" height="100"/>
      </TD>
    </TR>
    <TR>
      <TD>
        <H1>
          <xsl:value-of select="@title"/>
        </H1>
      </TD>
    </TR>
  </TABLE>
  <HR/>
  <B>1. Identification</B>
  <BR/>
  <TABLE>
    <TR>
      <TD>
        <TABLE>
          <TR>
            <TD>Manufacturer:</TD>
            <TD>
              <xsl:value-of select="@dtmi:manufacturerId"/>
            </TD>
          </TR>
          <TR>
            <TD>Classification:</TD>
            <TD>
              <xsl:value-of select="@fdt:classificationId"/>
            </TD>
          </TR>
        </TABLE>
      </TD>
    </TR>
  </TABLE>

```

```

        <TD>Device Type:</TD>
        <TD>
            <xsl:value-of select="@dtmi:deviceTypeId"/>
        </TD>
    </TR>
    <TR>
        <TD>Device Type Information:</TD>
        <TD>
            <xsl:value-of select="@dtmi:deviceTypeInfo"/>
        </TD>
    </TR>
    <TR>
        <TD>Date:</TD>
        <TD>
            <xsl:value-of select="@fdt:date"/>
        </TD>
    </TR>
    <TR>
        <TD>Descriptor:</TD>
        <TD>
            <xsl:value-of select="@fdt:descriptor"/>
        </TD>
    </TR>
    <TR>
        <TD>
            <BR/>
        </TD>
    </TR>
    <TR>
        <TD>Version:</TD>
        <TD>
            <xsl:value-of select="fdt:VersionInformation/@name"/>
        </TD>
    </TR>
    <TR>
        <TD>Vendor:</TD>
        <TD>
            <xsl:value-of select="fdt:VersionInformation/@vendor"/>
        </TD>
    </TR>
    <TR>
        <TD>Version's version:</TD>
        <TD>
            <xsl:value-of select="fdt:VersionInformation/@version"/>
        </TD>
    </TR>
    <TR>
        <TD>Version's date:</TD>
        <TD>
            <xsl:value-of select="fdt:VersionInformation/@date"/>
        </TD>
    </TR>
    <TR>
        <TD>Version's descriptor:</TD>
        <TD>
            <xsl:value-of select="fdt:VersionInformation/@descriptor"/>
        </TD>
    </TR>
</TABLE>
</TD>
<TD width="400" align="right">
    <xsl:if test="DocumentVariables/GraphicReference">
        <IMG>
            <xsl:attribute name="SRC">
                <xsl:value-of select="DocumentVariables/GraphicReference/@path"/>
            </xsl:attribute>
            <xsl:attribute name="width">200</xsl:attribute>
        </IMG>
    </xsl:if>
</TD>
</TR>
</TABLE>
<HR/>
</xsl:template>
</xsl:stylesheet>

```





## 14 Appendix – FDT XSL Transformation

### 14.1 Identification Transformation

In order to transform protocol specific XML documents to protocol independent XML documents, protocol specific XSL Transformations are to be used.

Syntax in this document: FDT~~xxx~~IdentTransformation.xsl where ~~xxx~~ is the protocol identification.

Examples:

- FDTHARTIdentTransformation.xsl
- FDTProfibusIdentTransformation.xsl

Frame Application has to copy this xsl file provided by Communication-DTMs into the protocol specific schema path and perform it to get protocol independent formatted identification documents, which can be validated using DTM[Scan|DeviceType]Schema.xml.

#### Hint:

It is strongly recommended for Communication-DTMs to use the official transformation style sheets from FDT-JIG.

Refer to best practice document for more information about tasks and rules.



The left picture describes the role of the XSLT for HART for the scan workflow. The XSLT is processed by a Frame Application. See XSLT implementations in protocol specific chapter.

The right picture describes the role of the XSLT for HART for the DTM device identification workflow.

Such a protocol specific transformation style sheet processes protocol specific XML documents returned by [OnScanResponse\(\)](#) as well as [GetDeviceIdentificationInformation\(\)](#) and creates protocol independent XML-documents.

- All protocol specific data formats are converted to strings.
  - All defined attributes with semantic meaning are transformed to defined protocol independent semantic tags.
- The semantic tag mapping rules are defined for each protocol in the corresponding protocol specific appendix.

## 15 Appendix – Channel schema

### 15.1 FDTBasicChannelParameterSchema

Channels that do not have any process related data (e.g. a pure Communication-Channel) should return a document based on [FDTBasicChannelParameterSchema](#).

It is recommended, to return instead of an empty document a document based on the [FDTBasicChannelParameterSchema](#).

Used at: [IFdtChannel::GetChannelParameters\(\)](#)  
[IFdtChannel::SetChannelParameters\(\)](#)

The XML document describes how to access a channel without process data.

Attribute	Description
gatewayBusCategory	Unique identifier for a supported bus type like Profibus or HART according to the FDT specific <a href="#">CATID</a>

Tag	Description
FDT	Root tag
FDTBasicChannel	Description of the channel
FDTChannelType	Description of the channel component in case of channels with gateway functionality

```
<Schema name="FDTBasicChannelParameterSchema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <!--Definition of Attributes-->
  <AttributeType name="gatewayBusCategory" dt:type="uuid"/>
  <!--Definition of Elements-->
  <ElementType name="FDTBasicChannel" content="empty" model="closed" order="seq">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:tag" required="yes"/>
    <attribute type="fdt:id" required="yes"/>
  </ElementType>
  <ElementType name="FDTChannelType" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1"/>
    <attribute type="gatewayBusCategory" required="no"/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="FDTChannelType" minOccurs="1" maxOccurs="1"/>
    <element type="FDTBasicChannel" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>
```

## 15.2 Template for Channel Schema

Channel objects are providing data via the interface IFdtChannel. The provided data are modeled within a bus protocol specific XML document e.g. based on FDTProfibusChannelParameterSchema schema or based on [FDTBasicChannelParameterSchema](#) schema.

The FDT<XXX>ParameterSchema is defining a basic structure containing a minimum set of attributes as a mandatory guideline regarding DTM specific channel documents.

Used at: [IFdtChannel::GetChannelParameters\(\)](#)  
[IFdtChannel::SetChannelParameters\(\)](#)

Attribute	Description
gatewayBusCategory	Unique identifier for a supported bus type like Profibus or HART according to the FDT specific <a href="#">CATID</a> or even DTM specific CATID
schemaVersion	Version of the schema

Tag	Description
FDT	Root tag
FDTXXXChannel	Description of the channel where 'XXX' contains a DTM specific part
FDTChannelType	Description of the channel component in case of channels with gateway functionality

```
<Schema name="FDT<XXX>ParameterSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:fdtinfo="x-schema:DTMInformationSchema.xml" >
  <!--Definition of Attributes-->
  <AttributeType name="gatewayBusCategory" dt:type="uuid"/>
  <AttributeType name="schemaVersion" dt:type="string" default="1.0"/>

  <!--Definition of Elements-->
  <ElementType name="FDT<XXX>Channel" content="empty" model="closed" order="seq">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:tag" required="yes"/>
    <attribute type="fdt:id" required="yes"/>
  </ElementType>
  <ElementType name="FDTChannelType" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1"/>
    <attribute type="gatewayBusCategory" required="no"/>
    <!--Additional channel specific attributes / elements to be place here-->
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdtinfo:FDTVersion" required="yes"/>
    <attribute type="schemaVersion" required="yes"/>
    <element type="FDTChannelType" minOccurs="1" maxOccurs="1"/>
    <element type="FDT<XXX>Channel" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>
```



## 16 Appendix – Update Hints

This chapter contains information about changes between version 1.2 and version 1.2.1

Description
Coversheet changed
Quick Reference removed
Implementation Hints moved to Best Practice documents
New chapter: DTM Registration
Additional description concerning Handling of SaveRequest()
DTM State Machine: New transition from 'up' to final state, to handle load errors, additional description in chapter 'Persistence Overview'
Additional description concerning XML attribute 'communicationError' (defined within FDTDataTypesSchema)
Additional description to clarify usage of some attributes and elements within HART Communication schema
Additional description to clarify behavior of IDtmOnlineParameter::CancelAction()
Additional comment for method IDtmEvents::2.1.1 OnOnlineStateChanged
Clarification regarding contradiction between chapter 4 'FDT Session Model and Use Cases' and 4.1 'Actors'
Clarification regarding combinations of CATID's, obsolete CATID's
Extension: DTM can provide documents like help files or technical documentation
Extension: DTM can provide different icons and bitmaps for different states of the device
Clarification of meaning of the delayTime and sequencetime elements in communication XML documents
Extension: DTM should provide GSD information as external file, clarification regarding language of GSD information
Additional description to clarify the use of address information in profibus channel XML documents
Additional description: Template for channel schematas
Extension: New schema for import/export of the topology
Extension: New interface which allow the Frame Application to trigger the address setting, automatic address setting without user interface
Extension: DTM gets unique human readable identifier of the DTM instance in the context of the Frame Application in order to set a system label e.g. for a message box
Clarification of chapter 'XXX Interface IFdtFunctionBlockData'
Extension of the ConnectRequest so that it identifies the source of the request by the systemTag.
Additional description: Handling of locking
Clarification of the operation phase for scanning
Clarification of DPV0 communication
Clarification of the mapping of DPV1 Data types to FDT data types
Additional hint for IDtm::GetFunctions regarding the possibility to limit the number of open ActiveX controls
Extension: Principles regarding Device initiated data transfer
Clarification regarding calling of Frame Application interfaces in dependency of DTM state
Extension: Association of functions to DTM modules
Extension: New interface which supports to open ActiveX controls in modal dialogs
Extension: default function introduced
Extension: Request to open/close an ActiveX Control for channels
Extension: HART Burst Mode
Extension: Channel mode introduced
Extension: Physical Layer information introduced to distinguish between physical differences (e.g. to handle PROFIBUS PA)
Improvements within XSL style sheet
Extension: Slave redundancy introduced

Description
Extension: Possibility to identify if parameter are changed in the device but not in the instance data set
Extension: Possibility to implement one ActiveX control for more than one channel functions introduced
Additional hint for Device-DTMs with more than one required protocol
Extension: DTM provides more information regarding the DTM state
Extension: Identification of the device: device type variant is a property of the physical device that has no influence to the software
Extension: for Process channels in order to enhance usage in different fields (i.e. Condition Monitoring, Asset Management, Tracing ...)
Extension: device and hardware identification introduced
Extension: New DTM classification Ids introduced
Extension: Mechanism for DTM upgrade introduced
Clarification/Extension regarding version interoperability
Extension: Handling of additional schemas which are needed for communication protocols
Extension: semantic information regarding the meaning and usage of a single value (modeled as exported variable or channel) by using an identifier introduced, scaling for ranges introduced
Extension: New scan mechanism introduced
Extension: New mechanism to access single data of a DTM
Update of the Required Runtime Environment
Extension: Block Type Manager introduced



© Copyright by:  
FDT Joint Interest Group  
[www.fdt-jig.org](http://www.fdt-jig.org)