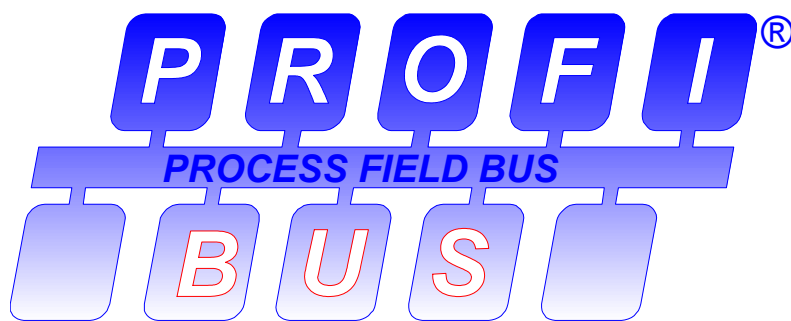


PROFIBUS



PROFIBUS Guideline

Specification for PROFIBUS Device Description and Device Integration

Volume 1: GSD V 3.1
Volume 2: EDDL V 1.0

Volume 3: FDT V 1.2

Mai 2001

PROFIBUS Guideline, Order No. 2.162

Specification for PROFIBUS Device Description and Device Integration

Volume 1: GSD V3.0

Volume 2: EDDL V1.0

Volume 3: FDT V1.2

Mai 2001

Prepared by the PROFIBUS Working Groups „GSD Specification“, „Device Description Language“ and „Engineering“ in the Technical Committee „System Integration“.

Publisher:
PROFIBUS Nutzerorganisation e.V.
Haid-und-Neu-Str. 7
D-76131 Karlsruhe

Phone: ++49 721 / 96 58 590
Fax: ++49 721 / 96 58 589
pi@profibus.com
www.profibus.com

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.

Preface

Synopsis:

This paper comprises the specifications for GSD (Basic Profibus Device Description), EDDL (Electronic Device Description Language) and FDT (Field Device Tool Interface). They are artefacts of working groups within Technical Committee 4 of the PROFIBUS Trade Organization.

Trademarks:

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

Abstract:

GSD, EDDL and FDT are representing the means to configure network devices and to parametrize and/or manipulate their operational modes. While GSD and EDD are based on human readable descriptions, FDT defines a set of interface to integrate device specific software components into engineering tools or other frameworks. GSD and EDDL are using device description languages and FDT defines a client/server relationship.

In order to meet the market requirements and the customer's needs this set of specifications is covering all the different aspects of complexity and usage, thus protecting the members' investments and providing scalable and compatible solutions.

Motivation

In process and manufacturing 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 provided by devices must be created and integrated into the function planning of the control system.

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 and hosting system environments. The electronic data exchange format is now standardized and the interfaces between those tools are defined.

Approach

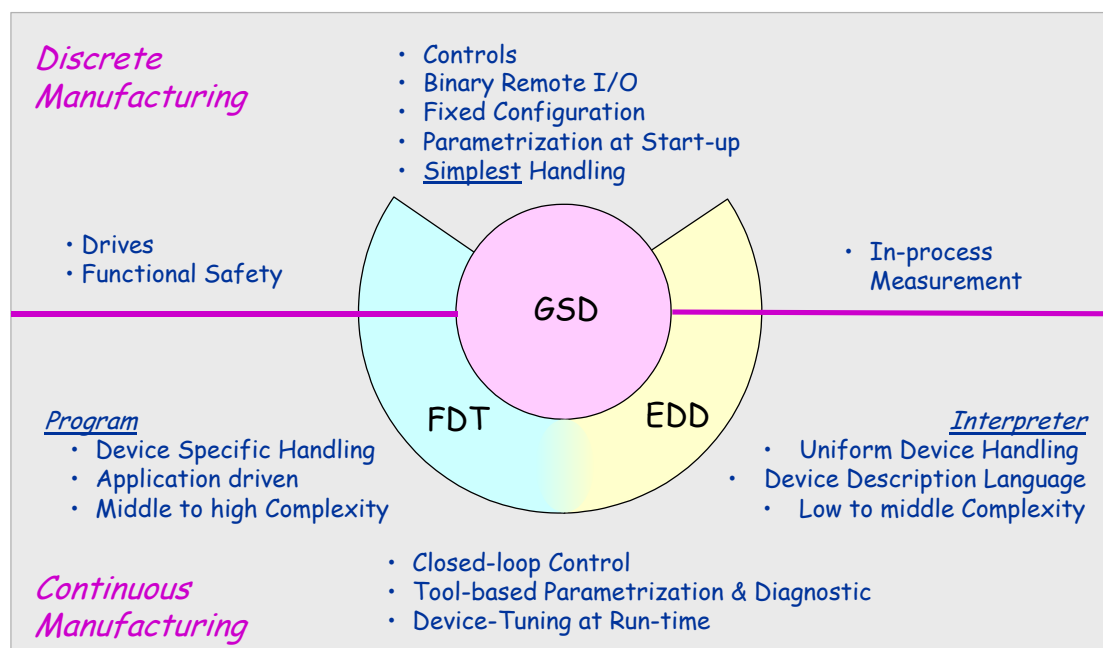


Fig. 1 GSD, EDD, FDT: A Scalable Solution for a Wide Set of Applications

GSD

PROFIBUS devices may have different behavior and performance characteristics. Features differ in regard to available functionality (i.e., number of I/O signals and diagnostic messages) or possible bus parameters such as baud rate and time monitoring. These parameters may vary individually for each device type and vendor and are usually documented in the technical manual. In order to achieve a simple Plug and Play configuration for PROFIBUS, electronic device data sheets (GSD files) are defined for the communication features of the devices. These GSD files allow easy configuration of PROFIBUS networks with devices from different manufacturers.

GSD is a human readable ASCII text file. Keywords are specified as mandatory or optional with the corresponding data type and their border values to support the configuration of PROFIBUS devices.

Based on the defined file format it is possible to realize vendor independent configuration tools for PROFIBUS systems. The configuration tool uses GSD files for testing the data. These were entered regarding limits and validity related to the performance of the individual device. New developments of PROFIBUS products will extend the functional range.

The manufacturer of a device is responsible for the functionality and the quality of its GSD file. The device certification procedure is requesting either a standard GSD file based on a PROFIBUS profile or a device specific GSD file.

EDDL

Up to now most of the devices have been configured by its own configuration tool. As a consequence the customer had to deal with as many tools as he was using device types. The Electronic Device Description Language has been designed to implement a vendor independent data set called EDD describing device configuration, maintenance and functionality. The EDDL defines the syntax (form) and the semantics (meaning) of the data and the behavior of a PROFIBUS device or component and the structure of the corresponding user interface. In its most basic form, the EDD source is human readable text written by device developers. The device manufacturer is responsible for completeness and correctness of his EDD source.

The EDD source can be easily incorporated into configuration tools just by reading it into an EDD interpreter (EDDI).

Configuration tool developers no longer need to be responsible for validation testing of all devices supported by their products. Device Description technology is state of the art for describing devices not only in the PROFIBUS arena but also in the environment of other fieldbus systems. Device Description Languages guarantee a uniform handling of all devices independent of the supplier and the type of the device. This means a user handles a temperature transmitter from a supplier A and a remote I/O system from a supplier B in the same way.

The EDDL specification provides a detailed description of the Electronic Device Description Language required for the development of an Electronic Device Description source file. The architecture of the EDD application and its usage during design and operational phases of a device are defined.

FDT

With the integration of fieldbusses into control systems, there are some more tasks that have to be performed. This applies to fieldbusses in general. Up to now there was no unified way to integrate device specific tools into engineering environments, console applications and diagnostic software. Especially within extensive and heterogeneous control systems, the unambiguous definition of interfaces with ease of use is getting growing importance. As simple as a new printer is added to a PC just by installing a driver, as simple should be the integration of a new device into an automation environment.

With the help of the FDT specification and its interface technology the user will be able to handle devices and their integration into engineering tools and other frameworks in a consistent manner. Due to the well-defined independance of system and device manufacturers the latter are enabled to support any innovative feature of their device without limitations.

This is done via a device-specific software component, called DTM (Device Type Manager). The device manufacturer is responsible for the functionality and the quality of a DTM. The DTM is integrated into engineering tools or other "frame applications" like stand-alone commissioning tools or web browsers that are providing the FDT interfaces. Even EDDI-Tools with the appropriate interfaces may be integrated this way. The approach to integration is in general open for all kind of fieldbusses (different protocols) and thus meets the requirements for integrating different kinds of devices into heterogeneous control systems.

An additional style guide is available for the development of DTMs in order to counteract the risk of proliferation of user interfaces.

Scalability via GSD, EDD and FDT

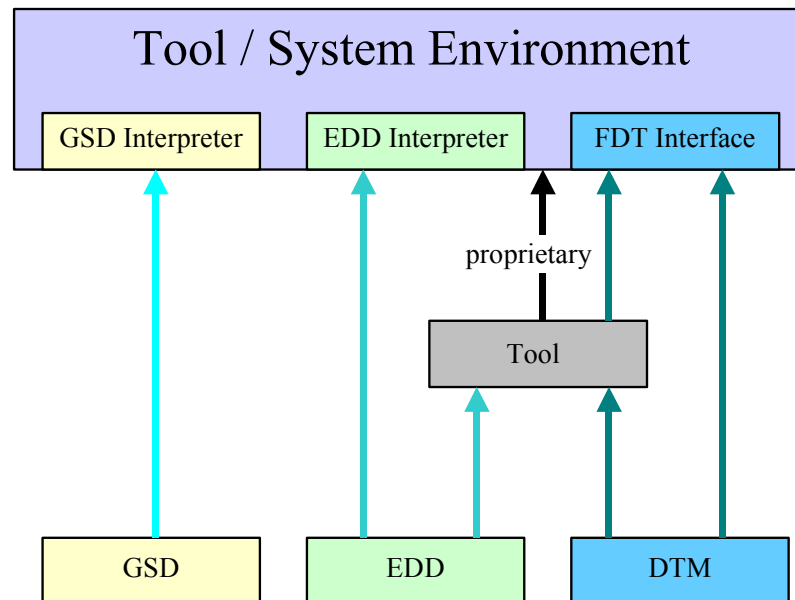


Fig. 2 Potential Integration Structures

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 this 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 a remote I/O or with fixed hardware but modular software blocks, e. g. a radar sensor
- D: Adjustable devices with modular hardware and programmable software blocks, for example a complex servo-drive

GSD, EDD and FDT are supporting all ranges of device complexity and integration levels into system environments.

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 Windows 98, Windows NT 4.0 Service Pack 4, Windows 2000, or later.

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.

To manage this task, control system manufacturers and device suppliers in ZVEI (The German Electrical and Electronic Manufacturers' Association) have formed a working group in order to work out a manufacturer-independent solution in a close dialogue with the PROFIBUS user organization (PROFIBUS Nutzer Organisation - PNO). The main aim of these activities is to specify a uniform software interface which on one hand, is easy to realize for the field device manufacturers, and, on the other hand, is just as easy to implement for the suppliers of process-control systems and engineering environments.

The activities of ZVEI and PNO were put together and located within the PNO to perform the specification work in a single organization.

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.

Table of Contents

1	INTRODUCTION	14
1.1	Problem	14
1.2	State of the Art	14
1.3	Aims	15
1.4	Technological Orientation	16
1.5	Solution Concept	17
1.6	Migration to DTM	18
1.7	Scope of Performance	19
1.8	Audience	19
2	FDT FUNDAMENTALS	20
2.1	FDT Overview	20
2.2	Where FDT Fits	21
2.3	General FDT Architecture and Components	22
2.4	Overview of Objects and Interfaces	23
2.4.1	The DeviceTypeManager (DTM)	23
2.4.2	The FDT Frame-Application	24
2.5	Synchronization and Serialization Issues	26
2.6	Parameter interchange via XML	26
2.6.1	Examples of usage	27
2.7	Persistent Storage Story	30
2.7.1	Persistence Overview	30
2.7.2	Persistence Interfaces	30
2.8	Basic features of a session model	31
2.9	Basic Operation phases	31
2.10	Abstract FDT Device Model	32
2.11	Fieldbus independent Integration	35
3	FDT INTERFACE	36
3.1	Overview of the FDT Interfaces	36
3.2	FDT Objects	37
3.2.1	FDT Object Model	37
3.2.2	DTM State Machine	39
3.3	Device Type Manager	41
3.3.1	Interface IDtm	41
3.3.2	Interface IDtmActiveXInformation	50
3.3.3	Interface IDtmApplication	52
3.3.4	Interface IDtmChannel	54
3.3.5	Interface IDtmDocumentation	55
3.3.6	Interface IDtmDiagnosis	56
3.3.7	Interface IDtmImportExport	59
3.3.8	Interface IDtmInformation	61
3.3.9	Interface IDtmOnlineDiagnosis	62
3.3.10	Interface IDtmOnlineParameter	64
3.3.11	Interface IDtmParameter	67
3.3.12	Interface IFdtCommunicationEvents	69
3.3.13	Interface IFdtEvents	72
3.4	DTM ActiveXControl	76
3.4.1	Interface IDtmActiveXControl	76
3.5	FDT Channel	78
3.5.1	Interface IFdtChannel	78
3.5.2	Interface IFdtChannelActiveXInformation	81
3.5.3	Interface IFdtCommunication	84
3.5.4	Interface IFdtCommunicationEvents	90
3.5.5	Interface IFdtChannelSubTopology	91
3.5.6	Interface IFdtFunctionBlockData	94
3.6	FDT Channel ActiveXControl	96
3.6.1	Interface IFdtChannelActiveXControl	96
3.7	FDT Container	98

3.7.1	Interface IDtmEvents	98
3.7.2	Interface IDtmAuditTrailEvents	108
3.7.3	Interface IFdtActiveX	110
3.7.4	Interface IFdtBulkData	112
3.7.5	Interface IFdtContainer	114
3.7.6	Interface IFdtDialog	117
3.7.7	Interface IFdtTopology	118
3.8	General Information	124
3.8.1	Version Interoperability	124
3.8.2	Ownership of Memory	125
3.8.3	Standard Interfaces	125
3.8.4	Dual Interfaces	125
3.8.5	COM Server Registration	125
3.8.6	Unicode	125
3.8.7	Nul Strings and Null Pointers	126
3.8.8	Asynchronous vs. Synchronous Behavior	126
3.8.9	Parameter and Data Types	126
3.8.10	ProglDs	126
4	FDT SESSION MODEL AND USE CASES	127
4.1	Actors	127
4.2	Use cases	129
4.2.1	Observation	129
4.2.2	Operation	130
4.2.3	Maintenance	137
4.2.4	Planning	145
4.2.5	OEM Service	150
4.2.6	Administrator	151
4.3	DTM Realization Elements	152
4.4	Frame-Application Realization Elements	157
5	FDT SEQUENCE CHARTS	160
5.1	DTM Peer To Peer Communication	160
5.1.1	Establish a Connection between DTM and Device	160
5.1.2	Asynchronous Connect	160
5.1.3	Asynchronous Disconnect	161
5.1.4	Asynchronous Transaction	162
5.2	Nested Communication	163
5.2.1	Generate Systemtopology	163
5.2.2	Establish a Connection between DTM and Device	165
5.2.3	Asynchronous Transaction	167
5.3	Topology Scan	168
5.4	Configuration of a Fieldbus Master	169
5.5	Starting and Releasing Applications	170
5.6	Channel Access	171
5.7	DCS Channel Assignment	172
5.8	Printing of DTM specific Documents	176
5.9	Printing of Frame specific Documents	177
5.10	Propagation of Changes	178
5.11	Locking	179
5.11.1	Locking for non synchronized DTMs	179
5.11.2	Locking for synchronized DTMs	181
5.12	Instantiation and Release	183
5.12.1	Instantiation of a new DTM	183
5.12.2	Instantiation of a existing DTM	183
5.12.3	Instantiation of a DTM User Interface	184
5.12.4	Release of a DTM User Interface	185
5.13	Persistent Storage of a DTM	186
5.13.1	State machine of instance data	186
5.13.2	Saving instance data of a DTM	187
5.13.3	Reload of a DTM object for another instance	188
5.14	Versioning	189

5.14.1	Copy of a DTM	189
5.15	Audit Trail	190
5.15.1	Frame-application supports Audit Trail	190
5.15.2	Frame-application doesn't support Audit Trail	191
5.16	Comparison of two Instance Data Sets	192
5.16.1	Comparison without User Interface	192
5.16.2	Comparison with User Interface	193
5.17	Failsafe Data Access	194
6	INSTALLATION ISSUES	195
6.1	FDT Registry and Device Information	195
6.1.1	Visibility of business objects of a DTM	195
6.1.2	Component Categories	195
6.1.3	Registry Entries	197
6.1.4	Installation Issues	197
6.1.5	Microsoft's Standard Component Categories Manager	197
6.1.6	Building a Frame-Application-Database of available DTMs and Supported Devices	198
7	DESCRIPTION OF DATA TYPES, PARAMETERS AND STRUCTURES	199
7.1	Ids	199
7.2	Data Type Definitions	200
8	GLOSSARY OF TERMS	201
9	LITERATURE	203
10	APPENDIX – FDT IDL	204
11	FDT QUICK REFERENCE	216
11.1	Standard COM Interfaces used by FDT	216
11.2	Custom Interfaces of FDT	216
11.3	Device Type Manager	216
11.3.1	Interface IDtm	216
11.3.2	Interface IDtmActiveXInformation	218
11.3.3	Interface IDtmApplication	218
11.3.4	Interface IDtmChannel	219
11.3.5	Interface IDtmDocumentation	219
11.3.6	Interface IDtmDiagnosis	219
11.3.7	Interface IDtmImportExport	220
11.3.8	Interface IDtmInformation	220
11.3.9	Interface IDtmOnlineDiagnosis	220
11.3.10	Interface IDtmOnlineParameter	220
11.3.11	Interface IDtmParameter	221
11.3.12	Interface IFdtCommunicationEvents	221
11.3.13	Interface IFdtEvents	222
11.4	DTM ActiveXControl	222
11.4.1	Interface IdtmActiveXControl	222
11.5	FDT Channel	223
11.5.1	Interface IFdtChannel	223
11.5.2	Interface IFdtChannelActiveXInformation	223
11.5.3	Interface IFdtCommunication	224
11.5.4	Interface IFdtCommunicationEvents	225
11.5.5	Interface IFdtChannelSubTopology	225
11.5.6	Interface IFdtFunctionBlockData	226
11.6	FDT Channel ActiveXControl	227
11.6.1	Interface IfdtChannelActiveXControl	227
11.7	FDT Container	227
11.7.1	Interface IDtmEvents	227
11.7.2	Interface IDtmAuditTrailEvents	229
11.7.3	Interface IFdtActiveX	229
11.7.4	Interface IFdtBulkData	230

11.7.5	Interface IFdtContainer	230
11.7.6	Interface IFdtDialog	231
11.7.7	Interface IFdtTopology	231
12	APPENDIX – IMPLEMENTATION HINTS	233
12.1	Initialization of a new DTM instance by calling IpersistXX::InitNew	233
12.2	Implementation of Ipersist Interfaces using Visual Basic	233
12.3	Instantiation of DTMs using Visual Basic	233
12.4	Access to Ipersist Interfaces using Visual Basic	233
12.5	Access to Import/Export Interface using Visual Basic	233
12.6	Implementation of Stream objects using Visual Basic	235
12.7	Usage of Microsoft XML-parser	235
12.8	DTM Documentation with Graphical Elements	235
12.9	Microsoft Category Manager access	237
12.10	DTMs running in Single Threaded Apartment	241
13	APPENDIX – FDT XML SCHEMAS	243
13.1	FDT Data Types	243
13.2	FDT Application Id	250
13.3	FDT User Information	251
13.4	DTM Information	253
13.5	DTM Functions	255
13.6	Parameter access	256
13.7	Documentation	263
13.8	Supported Fieldbus Protocols	266
13.9	Topology	267
13.10	Audit Trail	269
13.11	Device Status	271
13.12	Functions of a DTM	272
13.13	Functions of a DTM Channel Object	275
13.14	Online Compare of DTM Data	279
13.15	Failsafe	281
13.16	Scan	282
13.17	Operation Phase	283
13.18	DTM Initialization	284
13.19	User Message	285
13.20	DTM Information List	287
14	APPENDIX – FDT XML STYLES	289
14.1	Documentation	289
15	APPENDIX – PROFIBUS	293
15.1	Communication Schema	294
15.1.1	DPV0 Communication	294
15.1.2	DPV1 Communication	298
15.2	Channel Schema	301
15.3	Topology Scan Schema	304
15.4	Master-Bus Parameter Set (DP)	305
15.5	Slave Bus Parameter Set (DP)	306
15.6	Module and Channel Data	307
15.7	ProfiSafe	311
15.7.1	Motivation	311
15.7.2	General Parameter Handling	311
15.7.3	ProfiSafe i-Parameter	312
15.8	GSD Information	313
16	APPENDIX – HART	314
16.1	Communication Schema	314
16.2	Channel Schema	318
16.3	Topology Scan Schema	321
17	APPENDIX – UPDATE HINTS	322

1 Introduction

In this chapter, the motivation for the project, 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.

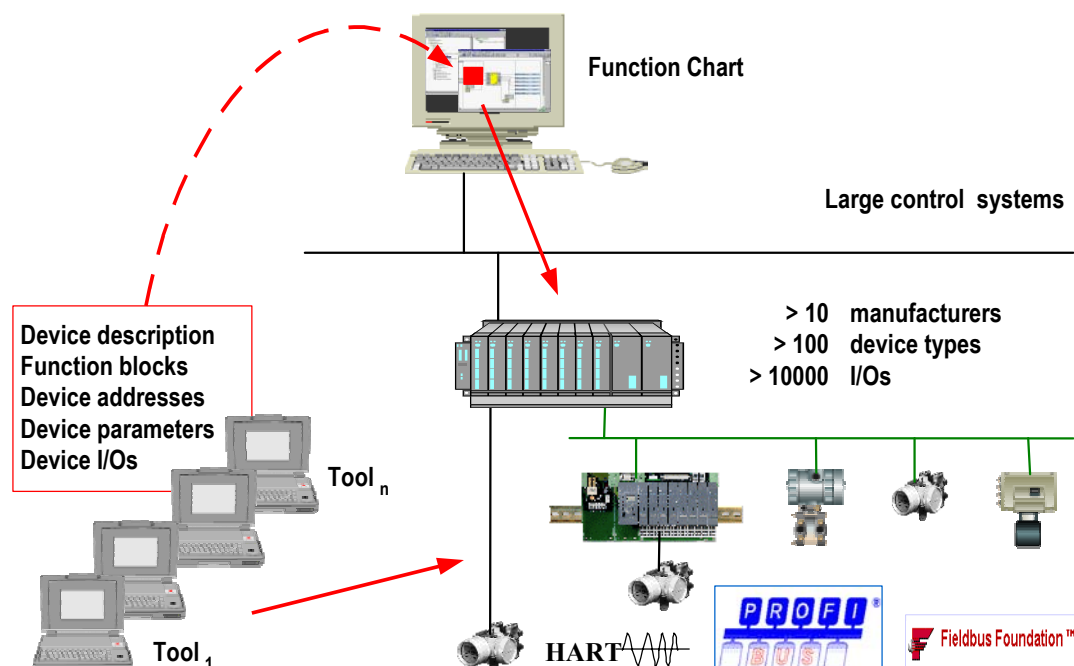


Figure 1: Different tools and multiple data input have determined field device integration to date

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.

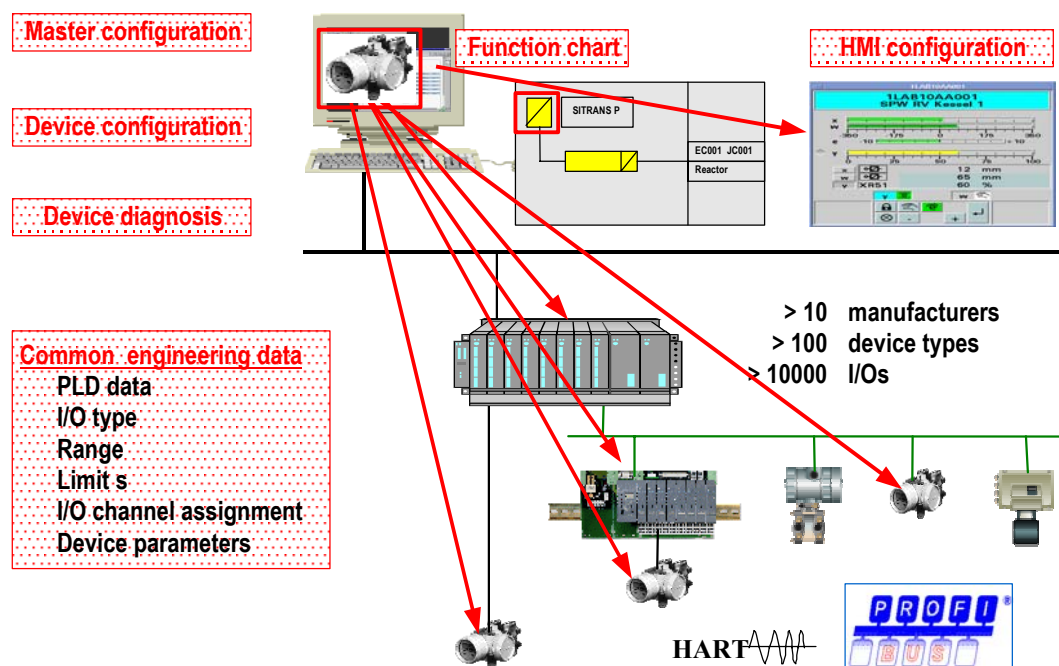


Figure 2: The potential of the field bus technology cannot be used until the field bus has been homogeneously integrated into the engineering systems.

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 in the first step 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.

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.

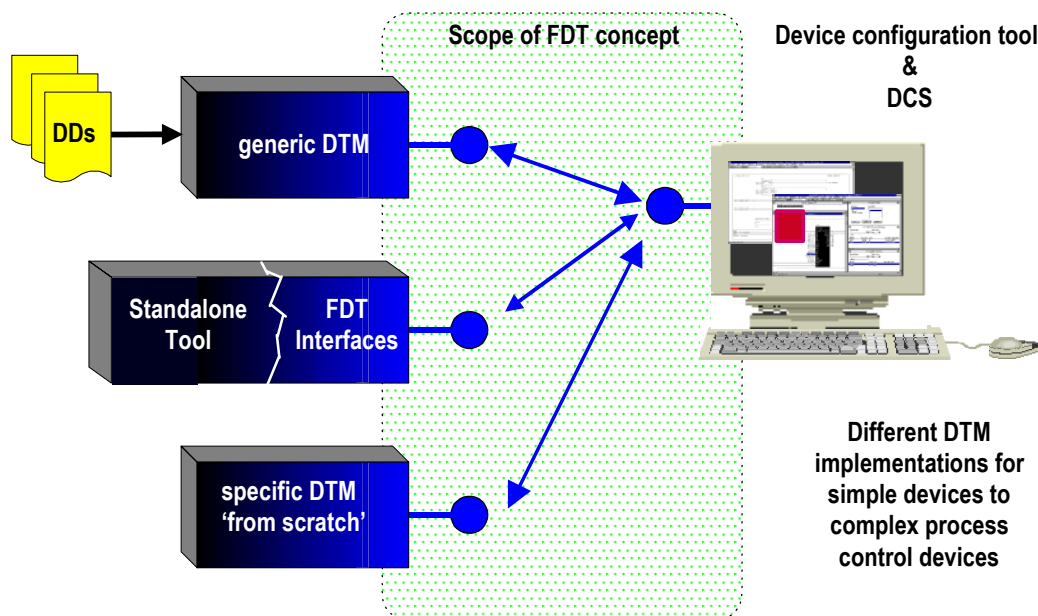


Figure 3: DTM - Implementations

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 lies on the PROFIBUS and the HART protocol. The corresponding XML schemas are included. Due to the scalable structure of FDT it will be easy to add new schemas in order to extend the scope to other protocols like Foundation Fieldbus or Modbus etc.

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.

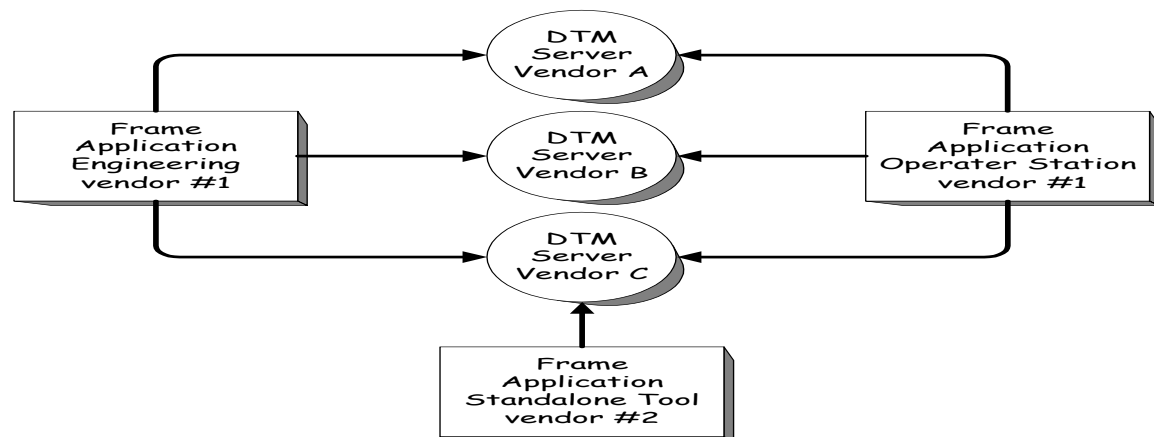


Figure 2-1 Generally FDT Client/Server Relationship

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.

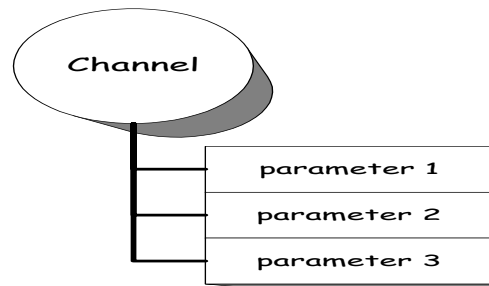


Figure 2-2 - Channel/Parameter Relationship

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 3.2.

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.

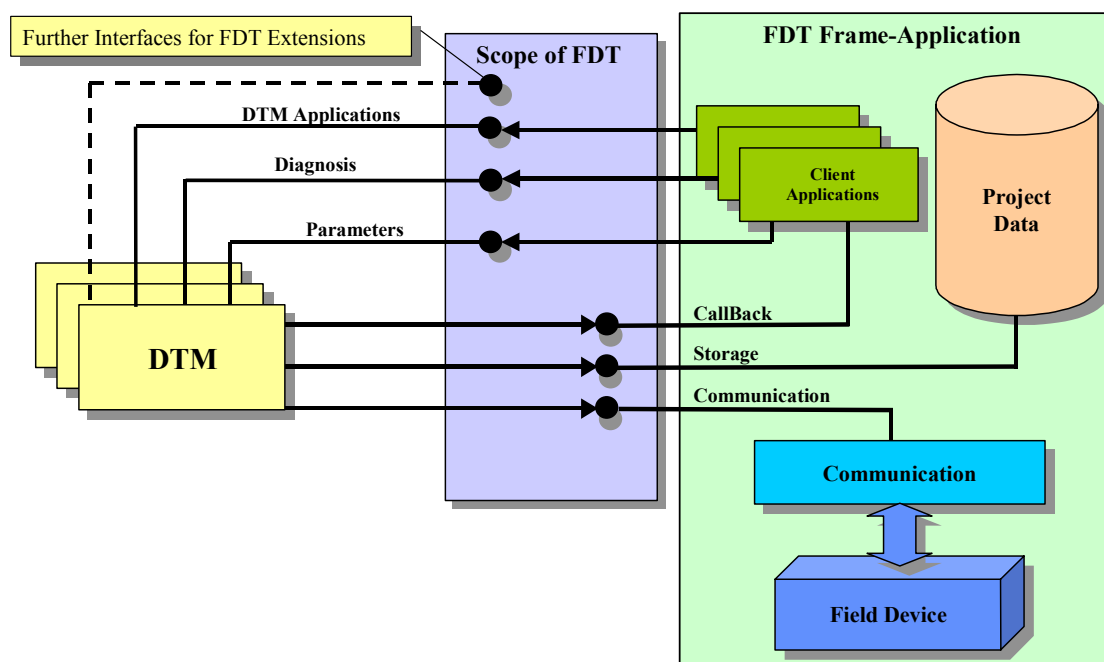


Figure 2-3 - The FDT Interfaces

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-applications.

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 DeviceTypeManager (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.

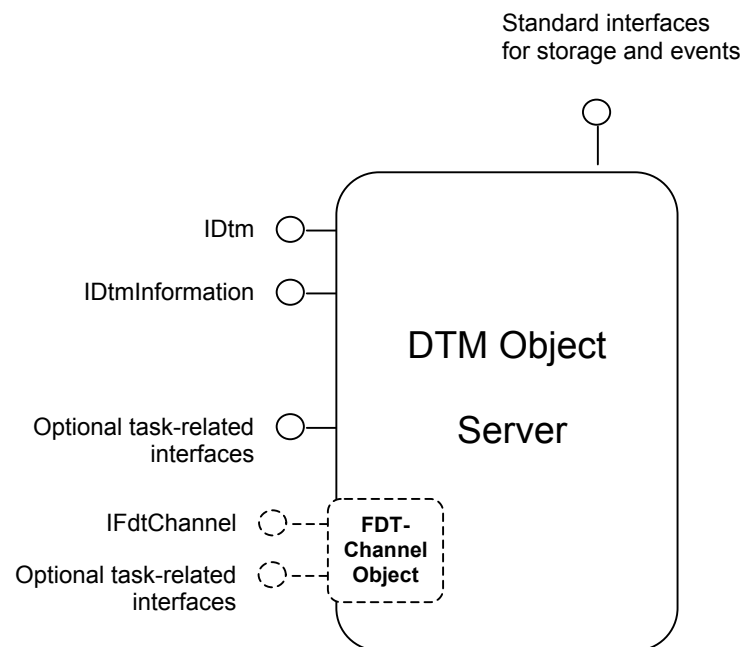


Figure 2-4 - The DTM Interfaces

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. 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 supports gateway functionality, like a HART to PROFIBUS gateway, the FDT-Channel object must implement further interfaces for the communication via this channel. Each interface of the FDT-Channel object will be described in detail at its own chapter.

2.4.2 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

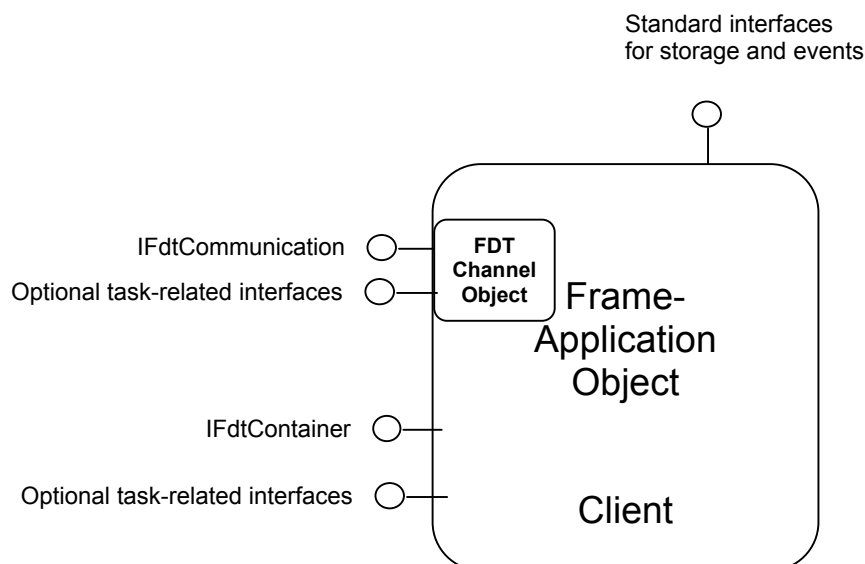


Figure 2-5 - The Frame-Application Interfaces

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 do 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 handle 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- (gateway) 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- (gateway) 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- (gateway) channels.

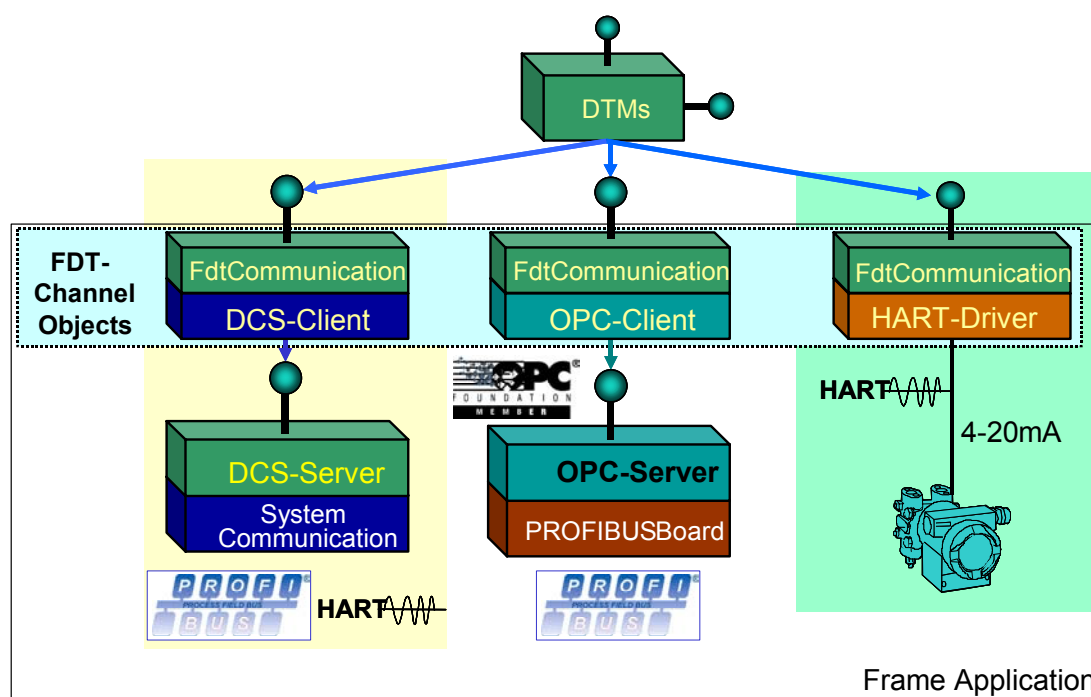


Figure 2-6 - The FDT Communication Layer

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 or fragments of a document. 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.

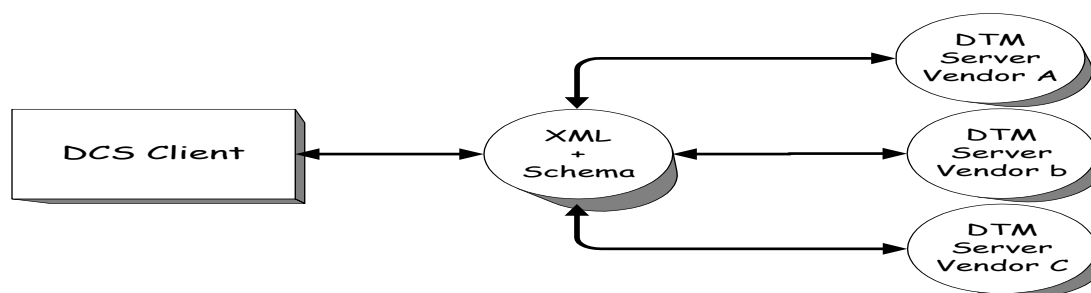


Figure 2-7 FDT Client/Server Relationship via XML

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.

A XML parser usually generates the DOM. Microsoft provides such a XML parser which ships with the Internet explorer 5. 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.

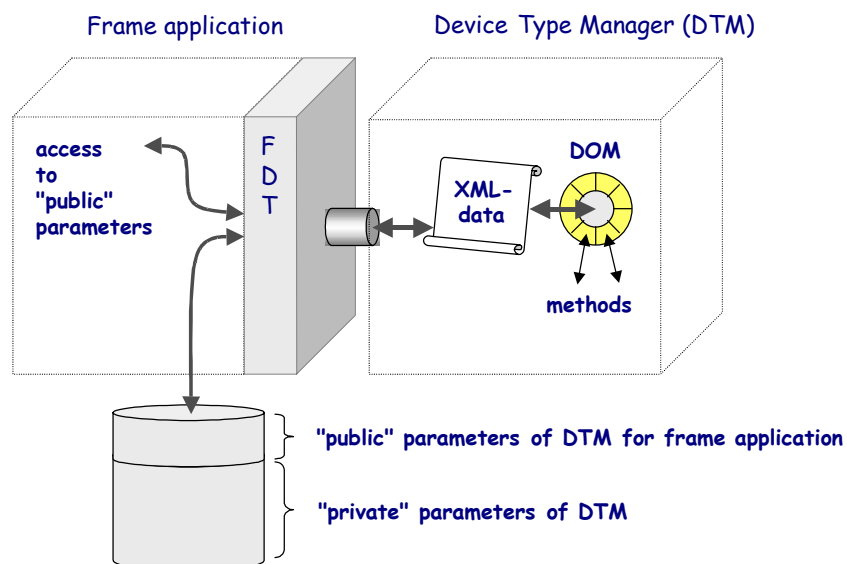


Figure 2-7-1 Data access and storage

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.

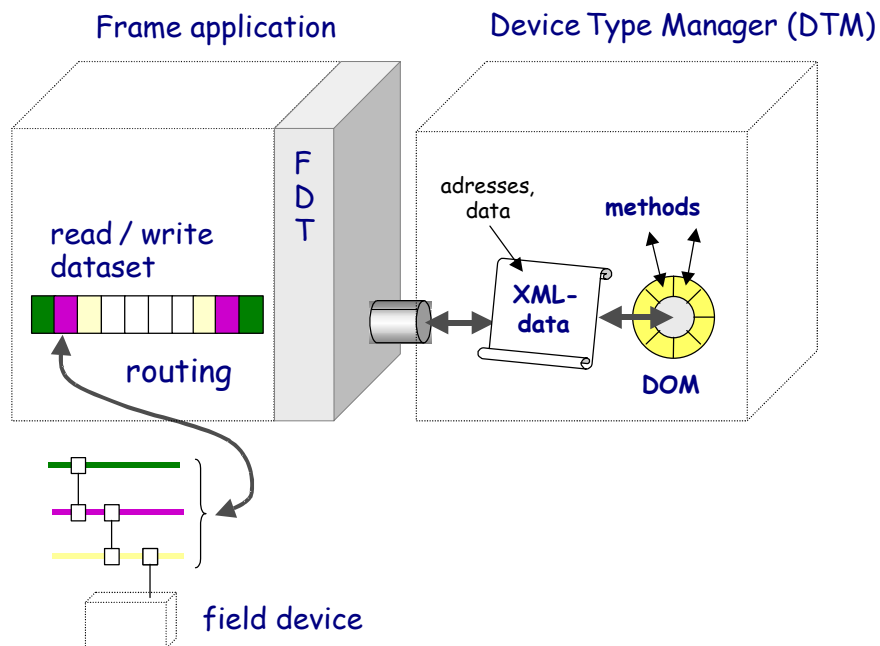


Figure 2-7-2 Communication

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.

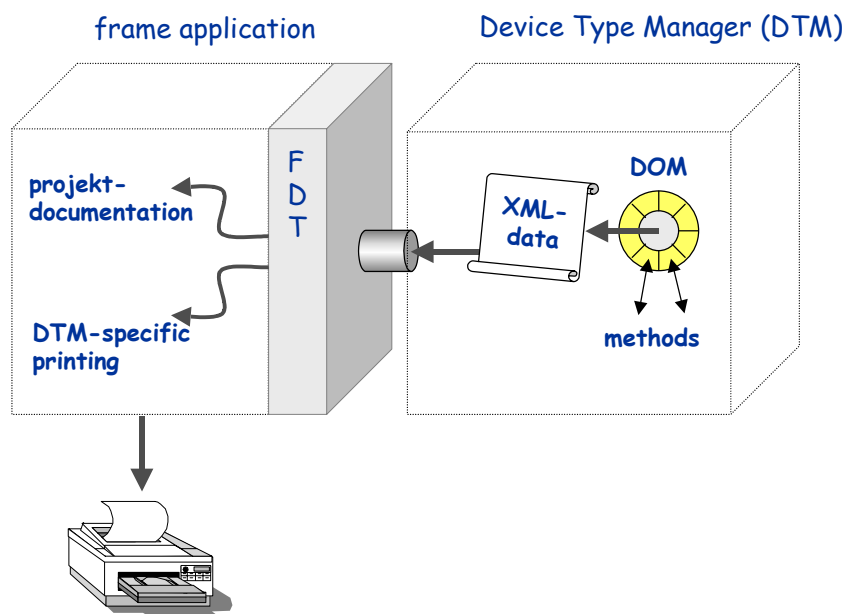


Figure 2-7-3 Documentation

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 a XML document.

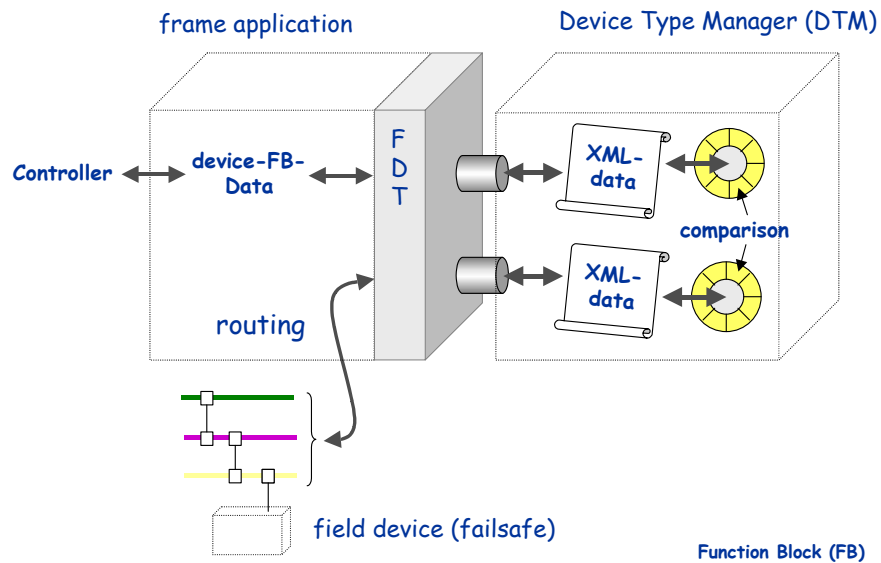


Figure 2-7-4 Parameter verification in case of failsafe devices

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-applications. 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.

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

If a DTM is started by the frame-application it needs all necessary information about the context which he is started in, so that it is able to determine the access rights for its parameters and methods:

- The current user actor (observer, operator, maintenance,...)
- The current use case
- The current basic operation phase.

There are three basic operation phases:

- **Engineering**
Planning and configuring of a plant,
no online access to the plant
- **Commissioning, workshop**
Installing the plant and the devices,
• 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.
Replacing defect devices.
Reading process values and diagnosis information.

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 or it forces a audit trail protocol of this action.

2.10 Abstract FDT Device 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 and FDTChannels serves the access to this information. Only DTM, 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 and channel objects is available via XML documents and exchanged via COM interfaces of these objects.

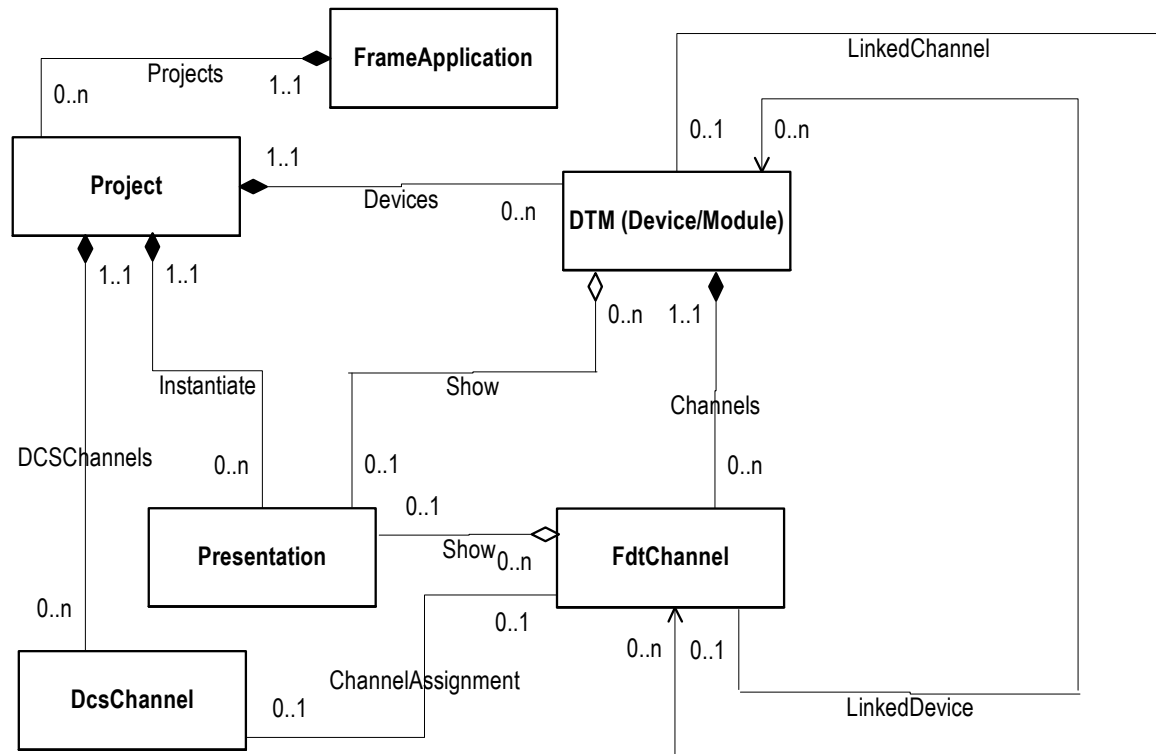


Figure 4-1 – FDT Objects-Device related

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.</p> <p>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.</p>
FdtChannel	<p>The fundamental part of a device or module is a channel which 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.</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.

	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 GetChildNodes()</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 GetParentNodes()</p> <p>The frame-application (project) is responsible for handling this association.</p>

2.11 Fieldbus independent Integration

The fieldbus independent integration into a frame-application is realized by category ids (**CATID**) defined within this specification. These category ids are defined within the FDT specification as UUIDs.

On one hand they are used at the Windows registry to specify the fieldbus type a DTM or channel supports. The registration with a category id allows the frame-application to pre-select a DTM before instantiation.

On the other hand 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.

3 FDT Interface

3.1 *Overview of the FDT Interfaces*

The FDT interface specification includes the following objects:

- DTM
- DTMActiveXControl
- FdtChannel
- FdtChannelActiveXControl
- FdtContainer

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-applications 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.

3.2 FDT Objects

3.2.1 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, 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.

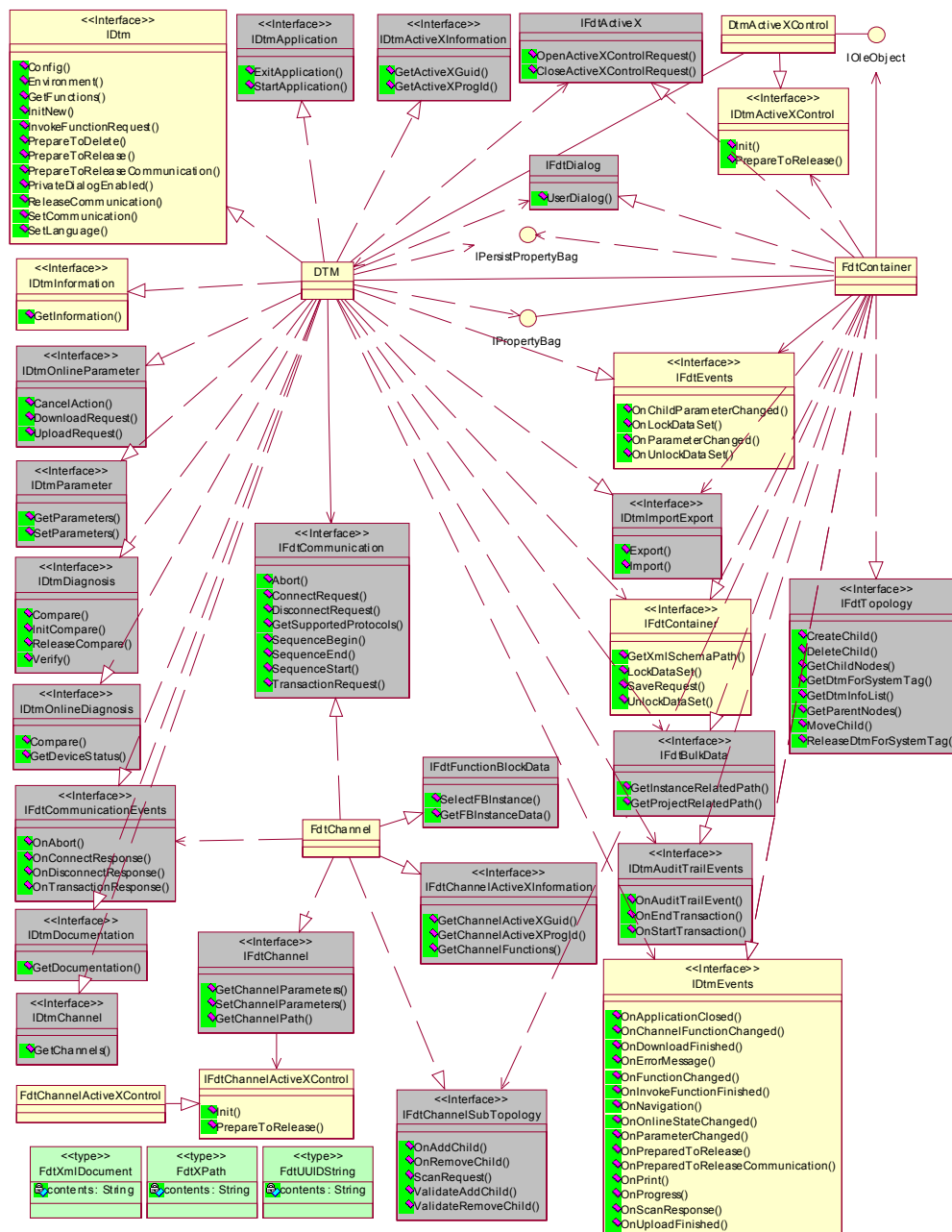
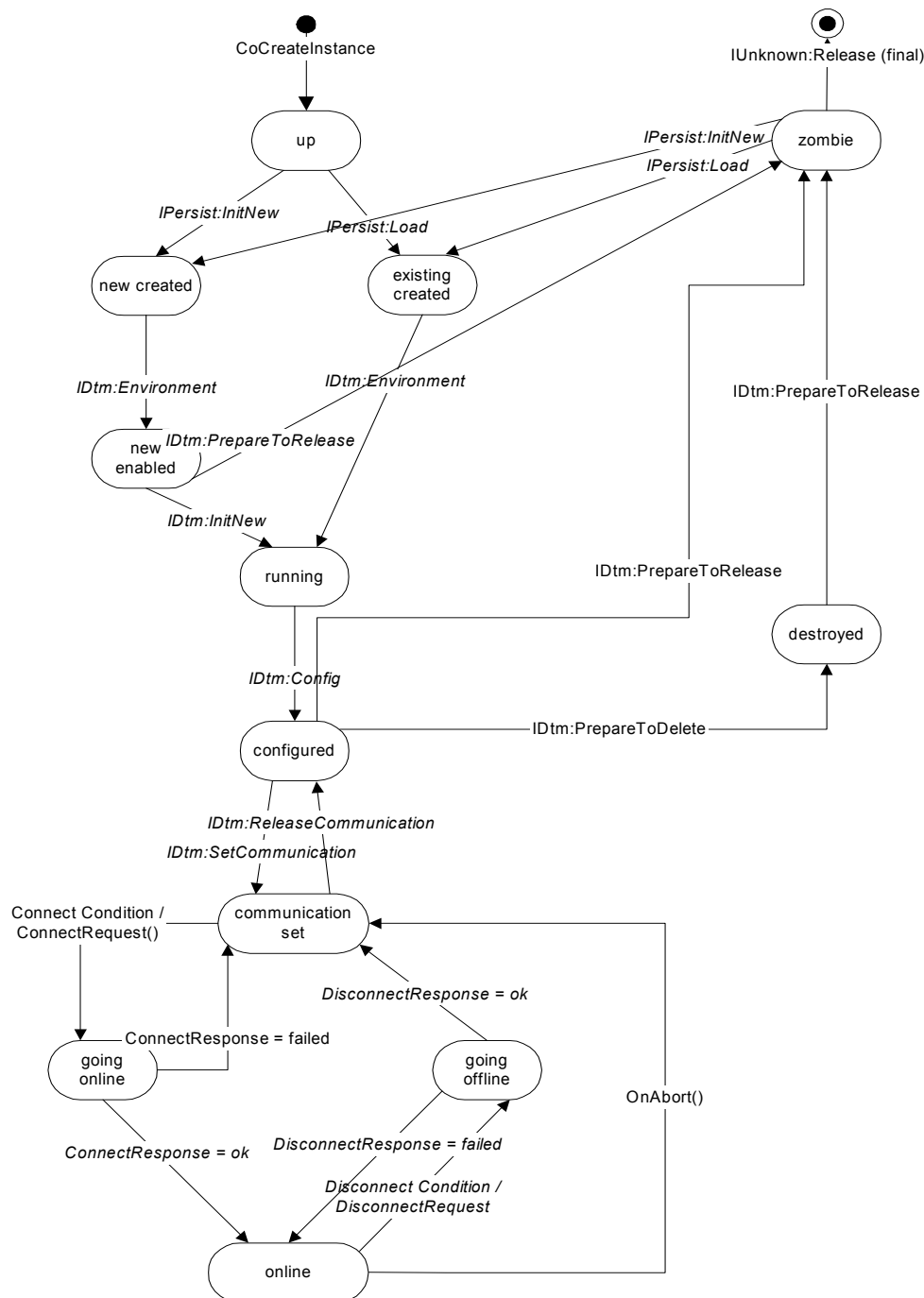


Figure 4-2 – FDT Objects-Task related

3.2.2 DTM State Machine

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



The table below signs the interfaces which can be used 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					√	√	√	√	√		√	
IDtmInformation			√	√	√	√	√	√	√		√	
IDtmActiveXInformation			√	√	√	√	√	√	√		√	
IDtmApplication						√	√	√	√		√	
IDtmChannel					√	√	√	√	√		√	
IDtmDocumentation						√	√	√	√		√	
IDtmDiagnosis						√	√	√	√		√	
IDtmOnlineDiagnosis								√	√		√	
IDtmOnlineParameter								√	√		√	
IDtmParameter						√	√	√	√		√	
IFdtCommunicationEvents							√	√	√		√	
IFdtEvents					√	√	√	√	√		√	
IFdtChannel					√	√	√	√	√		√	
IFdtChannelActiveXInformation			√	√	√	√	√	√	√		√	
IFdtChannelSubTopology												
OnAddChild					√	√	√	√	√		√	
OnRemoveChild					√	√	√	√	√		√	
ScanRequest								√	√		√	
ValidateAddChild					√	√	√	√	√		√	
ValidateRemoveChild					√	√	√	√	√		√	
IFdtCommunication							√	√	√		√	
IFdtFunctionBlockData						√	√	√	√		√	

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

3.3 Device Type Manager

3.3.1 Interface IDtm

This interface is the main interface of a DTM. 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.

3.3.1.1 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 FDTUserInformationSchema

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

-/-

3.3.1.2 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
systemTag	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. At this state all functions of the frame-application are available for a DTM.

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).

3.3.1.3 GetFunctions

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

Description

Returns a 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 FDTOperationPhaseSchema
----------------	--

Return Value

Return Value	Description
result	XML document containing actual supported functions specified by the DTMFunctionSchema

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 [StartApplication\(\)](#), [GetActiveXGuid\(\)](#) or [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.

Comments

-/-

3.3.1.4 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 DTMInitSchema

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

-/-

3.3.1.5 InvokeFunctionRequest

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

Description

Starts a function of a DTM.

Parameters	Description
invokeId	Identifier for the started function.
functionCall	XML document containing the DTM specific function id for the requested function or user interface specified by the DTMFunctionCallSchema

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 [StartApplication\(\)](#), [GetActiveXGuid\(\)](#) or [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.

3.3.1.6 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 no user interfaces of the DTM are open and that no communication is active. 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

-/-

3.3.1.7 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.

3.3.1.8 PrepareToReleaseCommunication

HRESULT [PrepareToReleaseCommunication](#)(
[out, retval] VARIANT_BOOL* result);

Description

Used to inform the DTM that it has to release its 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

-/-

3.3.1.9 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 [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

-/-

3.3.1.10 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

-/-

3.3.1.11 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 gateway-channel

Return Value

Return Value	Description
TRUE	Pointer accepted
FALSE	Invalid communication pointer

Behavior

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

The gateway-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 gateway-channels. This check can be done to ensure the link or in case of communication problems.

Comments

-/-

3.3.1.12 SetLanguage

```
HRESULT SetLanguage(
    [in] long languageld,
    [out, retval] VARIANT_BOOL* result);
```

Description

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

Parameters	Description
languageld	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. (see also WIN32/Platform SDK)

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](#) 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](#). On DTM instance is always initialized with one language. It is up to a DTM whether it can change the current language during runtime.

3.3.2 Interface IDtmActiveXInformation

This interface provides the user interface of a DTM as ActiveX controls for embedding within a frame-application.

3.3.2.1 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 DTMFunctionCallSchema

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 NUL string](#).

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

Comments

-/-

3.3.2.2 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 DTMFunctionCallSchema

Return Value

Return Value	Description
result	UUID 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

-/-

3.3.3 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.

3.3.3.1 ExitApplication

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

Description

Notification to a DTM to close all user interfaces or just a single user interface identified by the invoke id.

Parameters	Description
invokeId	Identifier for the started application. Same value as provided in the corresponding call of StartApplication()

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.

3.3.3.2 StartApplication

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

Description

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

	Description
invokeId	Identifier for the started application.
functionCall	XML document containing the function id for the requested function or user interface specified by the DTMFunctionCallSchema
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.

[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

-/-

3.3.4 Interface IDtmChannel

This interface is used for channel assignment and Nested Communication functionality. 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 gateway-channels, these channel objects are used to build the communication chain for Nested Communication.

3.3.4.1 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 additional supplies the communication interface for nested communication.

Comments

-/-

3.3.5 Interface IDtmDocumentation

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

3.3.5.1 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 DTMFunctionCallSchema

Return Value

Return Value	Description
result	XML document containing the requested documentation specified by the DTMDocumentationSchema

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 output is based on the Extensible Style sheet Language (XSL, see chapter 'Literature' for related documentation and examples). Within FDT a default style is defined (DTMDocumentationStyle). Nesting DTM specific style sheets can extend the default style. Within these style sheets also hyperlinks to additional documents or into the World Wide Web can be placed.

Comments

-/-

3.3.6 Interface IDtmDiagnosis

This interface provides the base diagnosis functions required by a frame-application for DTMs with configuration parameters.

3.3.6.1 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 [IDTMEvent::OnErrorMessage\(\)](#).

Comments

-/-

3.3.6.2 InitCompare

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

Description

Initializes a DTM for comparison of two device instances

Parameters	Description
systemTag	SystemTag 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\(\)](#).

3.3.6.3 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.

3.3.6.4 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

-/-

3.3.7 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 IDtmImportExport::Save(). If a DTM does not offer an IDtmImportExport interface the frame-application must use one of the IPersistXXX interfaces to retrieve and restore the instance data.

3.3.7.1 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 [IDtmImportExport](#) interface. If this interface is not provided by a DTM the frame-application uses one of the IPersistXXX interfaces for export/import.

Comments

-/-

3.3.7.2 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

-/-

3.3.8 Interface IDtmInformation

This interface is the second main interface of a DTM. 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.

3.3.8.1 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 DTMInformationSchema

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

-/-

3.3.9 Interface IDtmOnlineDiagnosis

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

3.3.9.1 Compare

```
HRESULT Compare(  
    [out, retval] FdtXMLDocument* result);
```

Description

Returns a 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 DTMOnlineCompareSchema

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.

Comments

-/-

3.3.9.2 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 DTMDeviceStatusSchema

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

-/-

3.3.10 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.

3.3.10.1 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 canceled an action it returns TRUE and will not fire the IDtmEvents::OnDownloadFinished() or IDtmEvents::OnUploadFinished() events.

If the DTM cannot cancel the selected action, it returns FALSE and will fire one of the *finished events* when the action is finished.

Comments

-/-

3.3.10.2 DownloadRequest

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

Description

Sends the request to write online data to the device.

Parameters	Description
invokeld	Identifier of the request
parameterPath	FdtXPath within the XML document for data fragment interchange

Return Value

Return Value	Description
TRUE	Request accepted
FALSE	Request cannot be performed

Behavior

Asynchronous function call that sends a 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 [IDTMEvent::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

-/-

3.3.10.3 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 for data fragment interchange

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 download was successful will be provided by [IDtmEvents::OnUploadFinished\(\)](#).

In case of errors the DTM should inform the frame-application via the callback interface [IDTMEvent::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

-/-

3.3.11 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.

3.3.11.1 GetParameters

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

Description

Returns a XML document with the device specific parameters.

Parameters	Description
parameterPath	FdtXPath within the XML document for data fragment interchange

Return Value

Return Value	Description
result	XML document with the device specific parameters specified by the DTMParameterSchema

Behavior

Returns a 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](#).

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](#)

3.3.11.2 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 for data fragment interchange
xmlDocument	XML document for data fragment interchange specified by the DTMParameterSchema

Return Value

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

Behavior

Returns TRUE if the DTM has accepted the complete document. FALSE means that the DTM has rejected all transferred changes. In this case the DTM informs the frame-application about the error via the callback interface [IDTMEvent::OnErrorMessage\(\)](#).

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

Comments

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

3.3.12 Interface IFdtCommunicationEvents

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

3.3.12.1 OnAbort

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

Description

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

Parameters	Description
communicationReference	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 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.

3.3.12.2 OnConnectResponse

```
HRESULT OnConnectResponse(
    [in] FdtUUIDString invokeld,
    [in] FdtXMLDocument response);
```

Description

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

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

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

-/-

3.3.12.3 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 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

-/-

3.3.12.4 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 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

-/-

3.3.13 Interface IFdtEvents

This interface is the callback-interface for the frame-application.

3.3.13.1 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
systemTag	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 a 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.

3.3.13.2 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
systemTag	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.

3.3.13.3 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
systemTag	Identifier of the device instance
parameter	XML document containing the changed parameters

Return Value

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

Behavior

If a DTM has changed any data it has to call [IDtmEvents::OnParameterChanged\(\)](#) with a 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.

Comments

-/-

3.3.13.4 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
systemTag	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 actual 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

-/-

3.4 DTM ActiveXControl

3.4.1 Interface IDtmActiveXControl

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

3.4.1.1 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 DTMFunctionCallSchema
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 it 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

-/-

3.4.1.2 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 failed.

Behavior

Releases the callback pointer of an ActiveX control to the according 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.

Comments

-/-

3.5 FDT Channel

3.5.1 Interface IFdtChannel

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

3.5.1.1 GetChannelParameters

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

Description

Returns a XML document with fieldbus dependent channel specific parameters.

Parameters	Description
parameterPath	FdtXPath within the XML document for data fragment interchange
protocolId	UUID of a fieldbusprotocol. This UUID must be one of the UUIDs returned by GetSupportedProtocols() and specified by DTMProtocolsSchema

Return Value

Return Value	Description
result	XML document with the channel specific parameters specified by a schema like FDTHARTChannelParameterSchema or FDTProfibusChannelParameterSchema

Behavior

Returns a 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.

Comments

-/-

3.5.1.2 GetChannelPath

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

Description

Returns an identifier for a channel.

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

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 gateway-channels there are some special rules for building the channelPath.

How to generate the channelPath of a gateway-channel, which receives data from a channel of a child device, depends on the functionality of the channel.

If the gateway-channel is passive, that means that it receives its data from a child device and provides this data without any changes within one of its own channels, 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 gateway-channel is active, that means that it receives its data from a child device for processing and provides result within one of its own channels, 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.

Also if only the channelPath has changed the DTM has to send [OnParameterChanged\(\)](#) with respect to the behavior.

Comments

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

3.5.1.3 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 for data fragment interchange
protocolId	UUID of a fieldbusprotocol. This UUID must be one of the UUIDs returned by GetSupportedProtocols() and specified by DTMProtocolsSchema
xmlDocument	XML document for data fragment interchange specified by a schema 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 a XML document with the channel specific parameters specified by a fieldbus specific schema. The fieldbus is selected 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 [IDTMEvent::OnErrorMessage\(\)](#).

Comments

-/-

3.5.2 Interface IFdtChannelActiveXInformation

Usually an FdtChannel does not have a user-interface, but in case of gateway-channels it may be required to have one. Depending on the fieldbus protocol and the capability of the gateway-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 gateway-channel and can only be configured by a gateway specific user-interface

3.5.2.1 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 DTMFunctionCallSchema

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 gateway-channel, it would be the user interface to set communication specific parameters.

Comments

-/-

3.5.2.2 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 DTMFunctionCallSchema

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 gateway-channel, it would be the user interface to set communication specific parameters.

Comments

-/-

3.5.2.3 GetChannelFunctions

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

Description

Returns a 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 FDTOperationPhaseSchema

Return Value

Return Value	Description
result	XML document containing actual supported functions specified by the DTMChannelFunctionsSchema

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

-/-

3.5.3 Interface IFdtCommunication

This interface is the communication entry point of a channel with gateway 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 gateway-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.

3.5.3.1 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 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

-/-

3.5.3.2 ConnectRequest

HRESULT [ConnectRequest](#)(

[in] [IFdtCommunicationEvents](#)* callBack,

[in] FdtUUIDString [invokeId](#),

[in] FdtUUIDString protocolId,

[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 like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

Return Value

Return Value	Description
TRUE	Request sent
FALSE	Request refused

Behavior

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

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 [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.

Comments

See also chapter FDT 'Use Cases and Scenarios'.

3.5.3.3 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 like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

Return Value

Return Value	Description
TRUE	Request sent
FALSE	Request refused

Behavior

The method passes a 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'.

3.5.3.4 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 DTMProtocolsSchema 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 a 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.

3.5.3.5SequenceBegin

```
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 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 communication component decides by the associated hardware whether it can support this function.

Returns FALSE if the last of the following communication components, possibly caused by the according hardware, does not supported this functionality.

The actual communication starts with the call to [IFdtSequenceCommunication::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.

3.5.3.6 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 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.

3.5.3.7 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 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.

3.5.3.8TransactionRequest

```
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 like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

Return Value

Return Value	Description
TRUE	Request sent
FALSE	Request refused

Behavior

The method passes a 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\(\)](#).

Comments

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

3.5.4 Interface IFdtCommunicationEvents

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

It is identical to [IFdtCommunicationEvents](#) of a DTM. The event interface of a channel is especially used for nested communication and must be provided by gateway-channels only.

3.5.5 Interface IFdtChannelSubTopology

This interface must be supported by gateway-channels and allows validating the sub-topology beneath a channel. A frame-application has always to configure the sub-topology of a channel via the interface IFdtTopology, but it has to call this interface so that the channel or at least the according DTM can validate the connections.

3.5.5.1 OnAddChild

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

Description

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

Parameters	Description
childSystemTag	SystemTag of the newly added child instance

Return Value

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

Behavior

The channel will be informed that a new device, specified by its systemTag, has been added to 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

-/-

3.5.5.2 OnRemoveChild

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

Description

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

Parameters	Description
childSystemTag	SystemTag of the child DTM which was removed

Return Value

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

Behavior

The channel will be 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

-/-

3.5.5.3 ScanRequest

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

Description

Requests the asynchronously scan of the sub topology.

Parameters	Description
invokeld	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

-/-

3.5.5.4 ValidateAddChild

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

Description

Validates if a given device, specified by its systemTag, can be to the sub-topology

Parameters	Description
childSystemTag	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

-/-

3.5.5.5 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
childSystemTag	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

-/-

3.5.6 Interface IFdtFunctionBlockData

This interface must be supported by DTMs for failsafe devices.

The interface is part of the channel of a bus-master-DTM to allow an extendable topology. The DTM that carries at least the failsafe function blocks (FB) would be part of the frame-application and would be a DCS specific.

Gateway-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 F-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 i-parameter block packed within a XML frame. The XML frame provides information about the bus master and the device FB.

3.5.6.1 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 Device-FB, located in the bus master.

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

Parameters	Description
systemTag	Identifier of the device instance

Return Codes

Return Code	Description
result	XML document containing the parameters of the failsafe device specified by the FDTFailSafeDataSchema

Behavior

The channel object routes upward in the topology to the bus master that contains the corresponding Device-FB of the DTM, reads the i-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 device-FB, at first it executes the browsing function as it is described for [SelectFBInstance\(\)](#)

Comments

-/-

3.5.6.2 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 Device-FB in the host (bus master) to the DTM which contains the parameters of the failsafe device.

Parameters	Description
systemTag	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 Device-FBs that contain i-parameter data of devices. The user has to select the Device-FB of the corresponding failsafe device of the DTM. The bus-master-DTM stores the assignment of the Device-FB to the calling DTM-Instance.

Comments

-/-

3.6 FDT Channel ActiveXControl

3.6.1 Interface IFdtChannelActiveXControl

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

3.6.1.1 Init

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

Description

Sets the callback pointer of an ActiveX control to the according FdtChannel.

Parameters	Description
invokeId	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

-/-

3.6.1.2 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

-/-

3.7 FDT Container

3.7.1 Interface IDtmEvents

This interface is the callback-interface for the DTMs.

3.7.1.1 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

3.7.1.2 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	Download successfully finished

Return Value

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

Behavior

Notification by a DTM, whether the asynchronously download function call [IDtmOnlineParameter::DownloadRequest\(\)](#) is successfully finished.

Comments

-/-

3.7.1.3 OnErrorMessage

HRESULT [OnErrorMessage](#)(
 [in] BSTR [systemTag](#),
 [in] BSTR errorMessage);

Description

Notification by a DTM about errors during an asynchronously function call.

Parameters	Description
systemTag	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

-/-

3.7.1.4 OnFunctionChanged

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

Description

Notification of a DTM that the information about its current available additional functionality has changed

Parameters	Description
systemTag	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

-/-

3.7.1.5 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
systemTag	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

-/-

3.7.1.6 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.

	Description
invokeld	Identifier of the closed application
success	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

-/-

3.7.1.7 OnNavigation

```
HRESULT OnNavigation(  
    [in] BSTR systemTag);
```

Description

A DTM sends an notification to cause the navigation to a frame-specific application.

Parameters	Description
systemTag	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 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

-/-

3.7.1.8 OnOnlineStateChanged

```
HRESULT OnOnlineStateChanged(  
    [in] BSTR systemTag,  
    [in] VARIANT_BOOL onlineState);
```

Description

A DTM sends an notification about its online state.

Parameters	Description
systemTag	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

-/-

3.7.1.9 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
systemTag	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

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.

Comments

-/-

3.7.1.10 OnPreparedToRelease

```
HRESULT OnPreparedToRelease(
    [in] BSTR systemTag);
```

Description

A DTM sends an notification that it can be released.

Parameters	Description
systemTag	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

-/-

3.7.1.11 OnPreparedToReleaseCommunication

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

Description

A DTM sends an notification that its communication pointer can be released.

Parameters	Description
systemTag	Identifier of the device instance

Return Value

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

Behavior

The DTM has released all references of the communication pointer set during [SetCommunication](#)(). After the frame-application has received this notification it can call [IDtm::ReleaseCommunication](#)() and release the communication pointer itself.

Comments

-/-

3.7.1.12 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
systemTag	Identifier of the device instance
functionCall	XML document containing the DTM specific function id for the requested document specified by the

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

-/-

3.7.1.13 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 an asynchronously function call.

Parameters	Description
systemTag	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

The method is useful if the DTM works without a user interface. If a DTM works without user interface it cannot display any progress information.
It's up to the frame-application how to handle the information.

Comments

-/-

3.7.1.14 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 DTMTopologyScanSchema .

Return Value

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

Behavior

Returns a 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

-/-

3.7.1.15 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	Upload successfully finished

Return Value

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

Behavior

Notification by a DTM, whether the asynchronously upload function call [IDtmOnlineParameter::UploadRequest\(\)](#) is successfully finished.

Comments

-/-

3.7.2 Interface IDtmAuditTrailEvents

This interface is 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.

3.7.2.1 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
systemTag	Identifier of the device instance
logEntry	XML document containing the changes specified by the DTMAuditTrailSchema

Return Value

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

Behavior

A DTM calls this function at the frame-application if it wants 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.

3.7.2.2 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
systemTag	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.

3.7.2.3 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
systemTag	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

-/-

3.7.3 Interface IFdtActiveX

This interface must be provided by a frame-application that uses DTMs which have a user interface based on the ActiveX controls

3.7.3.1 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

-/-

3.7.3.2 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
systemTag	Identifier of the device instance
functionCall	XML document containing the DTM specific function id for the requested user interface specified by the DTMFunctionCallSchema

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.

3.7.4 Interface IFdtBulkData

The Bulk Data Interface should offer a DTM the possibility, to store a big amount of additional data like protocols of measured values or historical data of configuration changes. The DTM should be able to do that in a private way.

It must not impact the instance specific configuration data set of a DTM if the access to these data is not possible. 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.

3.7.4.1 GetInstanceRelatedPath

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

Description

Returns the instance related path for bulk data.

Parameters	Description
systemTag	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.

3.7.4.2 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
systemTag	Identifier of the device instance

Return Value

Parameters	Description
result	Project related path (directory-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.

3.7.5 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.

3.7.5.1 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

-/-

3.7.5.2 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 unique read/write access for the currently loaded data set.

Parameters	Description
systemTag	Identifier of the device instance

Return Value

Return Value	Description
TRUE	Data set is locked for read/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.

Comments

Within a single user system the method returns always TRUE.

3.7.5.3 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
systemTag	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

Comments

Concerning multi user access the frame-application must reject the save request if the DTM has no write access rights.

3.7.5.4 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
systemTag	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.

Comments

Within a single user system the method returns always TRUE.

3.7.6 Interface IFdtDialog

This interface provides a functionality which allows a DTM to display messages like error, warning, and information.

3.7.6.1 UserDialog

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

Description

Call the Container to display a message

Parameters	Description
systemTag	Identifier of the device instance
userMessage	XML document according to the message specified by the FDTUserMessageSchema

Return Value

Return Value	Description
result	XML document according to the user action specified by the FDTUserMessageSchema

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

-/-

3.7.7 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.

3.7.7.1 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 DTMLnitSchema
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 5.12.1 Instantiation of a new DTM. It is also in the responsibility of the frame-application to insert the created DTM into the topology.

Comments

-/-

3.7.7.2 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
systemTag	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 PrepareToDelete() with respect of the behavior. Only instance data sets which are locked by the frame-application can be deleted. The operation will also fail, if a sub topology exists.

Comments

-/-

3.7.7.3 GetChildNodes

```
HRESULT GetChildNodes(
    [in] BSTR systemTag,
    [in] FdtXPath channelPath,
    [out, retval] FdtXMLDocument* result);
```

Description

Returns a XML document containing the systemTags of all child DTMs of the DTM identified by its system tag and the channel path.

Parameters	Description
systemTag	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 DTMSystemTagListSchema

Behavior

Returns a 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 interface.

3.7.7.4 GetDtmForSystemTag

```
HRESULT GetDtmForSystemTag(  
    [in] BSTR systemTag,  
    [out, retval] IDtm** result);
```

Description

Return the associated DTM according the given system tag

Parameters	Description
systemTag	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

-/-

3.7.7.5 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 DTMInfoListSchema

Behavior

Returns an XML-document containing a list of DTM related information. This information could be used to create a 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.

3.7.7.6 GetParentNodes

```
HRESULT GetParentNodes(
    [in] BSTR systemTag,
    [out, retval] FdtXMLDocument* result);
```

Description

Returns an XML-document containing al list of system tags of all parent DTMs of the DTM identified by its system tag.

Parameters	Description
systemTag	Identifier of the device instance

Return Value

Return Value	Description
result	XML document containing a list of system tags DTMSystemTagListSchema

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.

3.7.7.7 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.

	Description
systemTag	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(). The operation will fail if the instance data set could not be locked by the frame-application (if the frame-application supports multi user environment)

Comments

-/-

3.7.7.8 ReleaseDtmForSystemTag

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

Description

Release the associated DTM according the given system tag

Parameters	Description
systemTag	Identifier of the device to release

Return Value

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

Behavior

Release the associated DTM according the given system tag. This method is used only in combination with [GetDtmForSystemTag\(\)](#).

Comments

-/-

3.8 General Information

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.

3.8.1 Version Interoperability

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.

Required	1.20	User Interface	ActiveX Control User Interface	Device with Online Data	Fieldbus with Master	Device with Gateway Channel
Device Type Manager						
IDtm	Mandatory					
IDtmActiveXInformation	Optional		Mandatory			
IDtmApplication	Optional	Mandatory				
IDtmChannel	Optional				Mandatory	Mandatory
IDtmDocumentation	Optional					
IDtmDiagnosis	Optional					
IDtmImportExport	Optional					
IDtmInformation	Mandatory					
IDtmOnlineDiagnosis	Optional					
IDtmOnlineParameter	Optional			Mandatory		
IDtmParameter	Optional			Mandatory		
IFdtCommunicationEvents	Optional			Mandatory		
IFdtEvents	Mandatory					
DTM ActiveXControl						
IDtmActiveXControl	Mandatory		Mandatory			
FDT Channel						
IFdtChannel	Optional				Mandatory	
IFdtChannelActiveXInformation	Optional		Mandatory			
IFdtChannelSubTopology	Optional					Mandatory
IFdtCommunication	Optional					Mandatory
IFdtCommunicationEvents	Optional					Mandatory
IFdtFunctionBlockData	Optional					Mandatory
FDT Channel ActiveXControl						
IFdtChannelActiveXControl	Mandatory		Mandatory			
FDT Container						
IDtmEvents	Mandatory					
IDtmAuditTrailEvents	Optional					
IFdtActiveX	Optional		Mandatory			
IFdtBulkData	Optional					
IFdtContainer	Mandatory					
IFdtDialog	Optional		Mandatory			
IFdtTopology	Optional					Mandatory

Furthermore, the prefixes FDT and DTM 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.

3.8.2 Ownership of Memory

Per COM specification, clients must free all memory associated with 'out' or 'in/out' parameters. This includes memory that is pointed to by elements within any structures. This is very important for client writers to understand. Otherwise they will experience memory leaks that are difficult to find. See the IDL files to determine which parameters are out parameters.

The recommended approach is for a client to create a subroutine that is used for freeing each type of structure properly.

For further details refer to the standard COM documentation like the Microsoft MSDN library

3.8.3 Standard Interfaces

Per COM specification, all methods must be implemented on each required interface.

Per COM specification, any optional interfaces that are supported must implement all functions of that interface, even if the implementation is only a stub implementation returning E_NOTIMPL

3.8.4 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.

3.8.5 COM Server Registration

COM server registration has to be done by following the rules defined by Microsoft's COM specification. Generally, both DLL and EXE server FDT components must support self-registration.

Due to a bug in NTs CreateProcess(), EXE COM servers should be registered with their short pathname and since LoadLibrary() has also a bug, all DLL COM servers should be registered with their long pathname. Depending on your development environment (ATL within VC++ 6-SP3 is safe, but earlier version are not) you can get errors registering or using your COM servers in installation directories with paths including blanks.

For details, see Microsoft knowledge base articles Q218475, Q201318, Q179690.

3.8.6 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.

3.8.7 Nul Strings and Null Pointers

Both of these terms are used below. They are NOT the same things. A NULL Pointer is an invalid pointer (0) which will cause an exception if used. A NUL String is a valid (non zero) pointer to a 1-character array where that character is a NUL (i.e. 0). If a NUL string is returned from a method as an [out] parameter (or as an element of a structure) it must be freed, otherwise the memory containing the NUL will be lost. Also note that a NULL pointer cannot be passed for an [in,string] argument due to COM marshalling restrictions. In this case a pointer to a NUL string should be passed to indicate an omitted parameter.

3.8.8 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.

3.8.9 Parameter and Data Types

In most cases the automation data types are enough to describe a parameter. Especially for fieldbus communication there are several exceptions like 12 bit real or integer values or Motorola format. Therefore, the FDT specification defines a special data type which has to be set at the XML document for each parameter if the automation value has to be converted.

3.8.10 ProgIds

The usage of progIds is limited to 39 characters. This decision was made to support C++, Visual Basic, Java and other COM compliant development languages.

4 FDT Session Model and Use Cases

The following figure shows the main use cases.

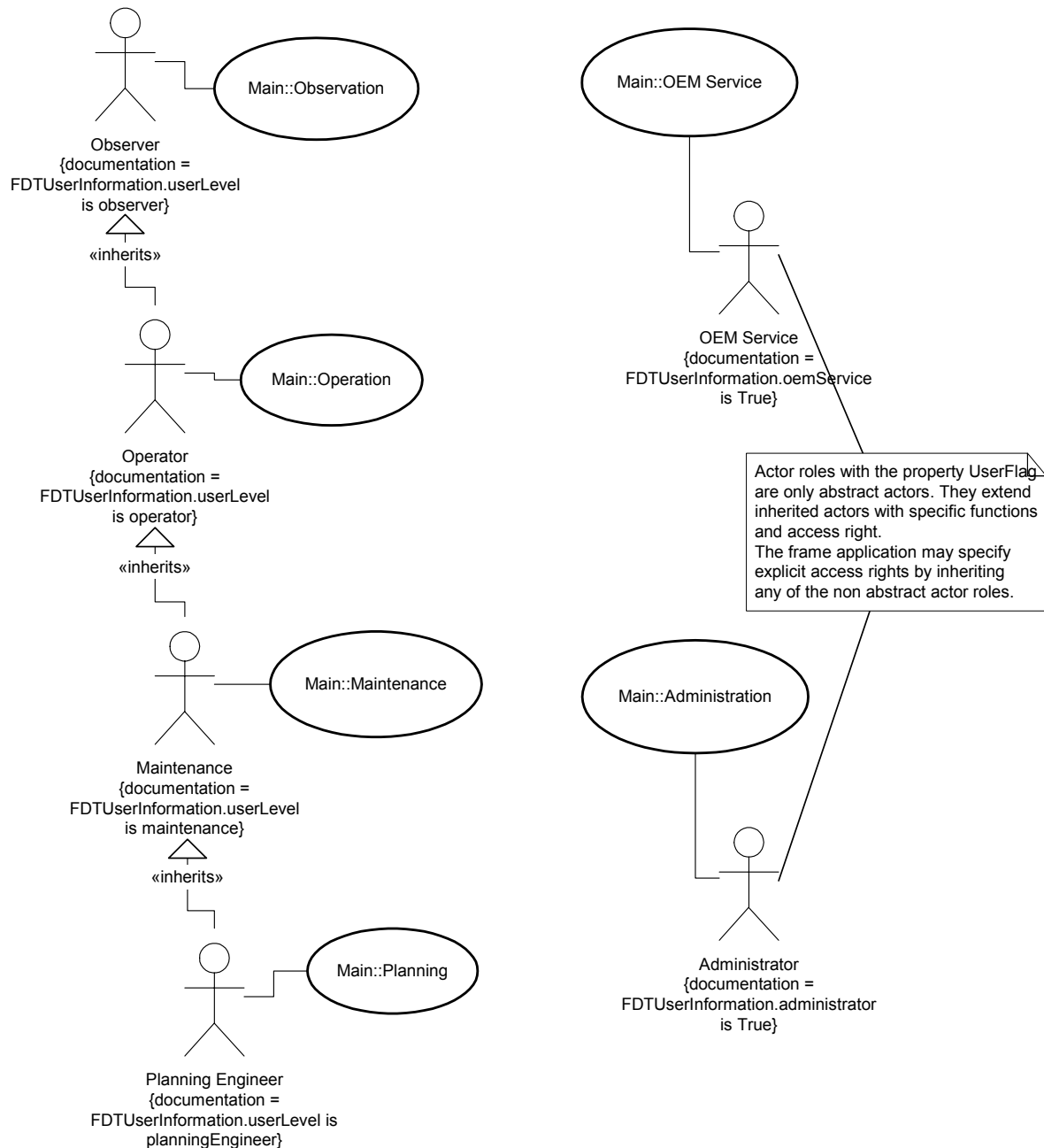


Figure 4: Use case - Main Use Case Diagram

They are specified in more detail in the following sections, including textual descriptions and optionally some necessary scenarios.

4.1 Actors

The actor types in the above displayed use case diagram are structured in a hierarchical way. The “[Maintenance](#)” actor inherits the “[Operator](#)” actor. Both the “[OEM Service](#)” and the “[Planning Engineer](#)” actors inherit the use cases of the “[Maintenance](#)” actor. The “[Administrator](#)” inherits the use cases of the “[Planning Engineer](#)”. Use cases, which are bequeathed 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:

Actor Name:	Observer
Brief Description:	This actor stands for a person that may only observe the current process.
Inherits:	None
User Level:	FDTUserInformation.userLevel = observer
Access Verification:	The access as "Observer" actor may not have a password.
Use Cases:	None

Actor Name:	Operator
Brief Description:	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.
Inherits:	Observer
User Level:	FDTUserInformation.userLevel = operator
Access Verification:	The access as "Operator" requires a password.
Use Cases:	User Login, Online View, Audit Trail, Archive, Report Generation, Asset Management

Actor Name:	Maintenance
Brief Description:	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.
Inherits:	Operator
User Level:	FDTUserInformation.userLevel = maintenance
Access Verification:	The access as "Maintenance" actor requires a password.
Use Cases:	Simulation, Online Operation, Repair, Offline Operation, Bulk Data Handling User Login*, Online View*, Audit Trail*, Archive*, Report Generation*, Asset Management* (* Inherited use cases)

Actor Name:	Planning Engineer
Brief Description:	The actor "Planning Engineer" 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.
Inherits:	Maintenance
User Level:	FDTUserInformation.userLevel = planningEngineer
Access Verification:	The access as "Planning Engineer" actor requires a password.
Use Cases:	DTM Instance Handling, Simulation*, Online Operation*, Repair*, Offline Operation*, Bulk Data Handling*, User Login*, Online View*, Audit Trail*, Archive*, Report Generation*, Asset Management* (* Inherited use cases)

Actor Name:	Administrator (abstract actor)
Brief Description:	The " Administrator " actor stands for person that has to perform administrative operations within an engineering environment. He is responsible for adding and removing of software components.
Inherits:	Depends on the frame-application
User Flag*:	FDTUserInformation.administrator = TRUE
Access Verification:	The access as " Administrator " actor requires a password.
Use Cases:	Integration and all inherited use cases

	OEM Service (abstract actor)
Brief Description:	The actor " OEM Service " may perform the complete set of DTM specific functions. OEM specific operation may be accessible to an " OEM Service " actor, like resetting of internal counters and exchanging firmware. The " OEM Service " has only inherited use cases, but the inherited use cases (especially the " Online Operation " use case) may have significant extensions.
Inherits:	Depends on the frame-application
User Flag*:	FDTUserInformation.oemService = TRUE
Access Verification:	The access verification for an " OEM Service " 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.
Use Cases:	Inherited use cases only

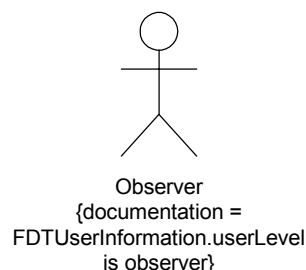
* "User Flags" may be combined with any "User Level", to specify the inherited actor role of an "Administrator" or "OEM Service" actor.

4.2 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.

4.2.1 Observation



The "[Observer](#)" stands for a person, that has the lowest access level. No use case is associated with this person.

4.2.2 Operation

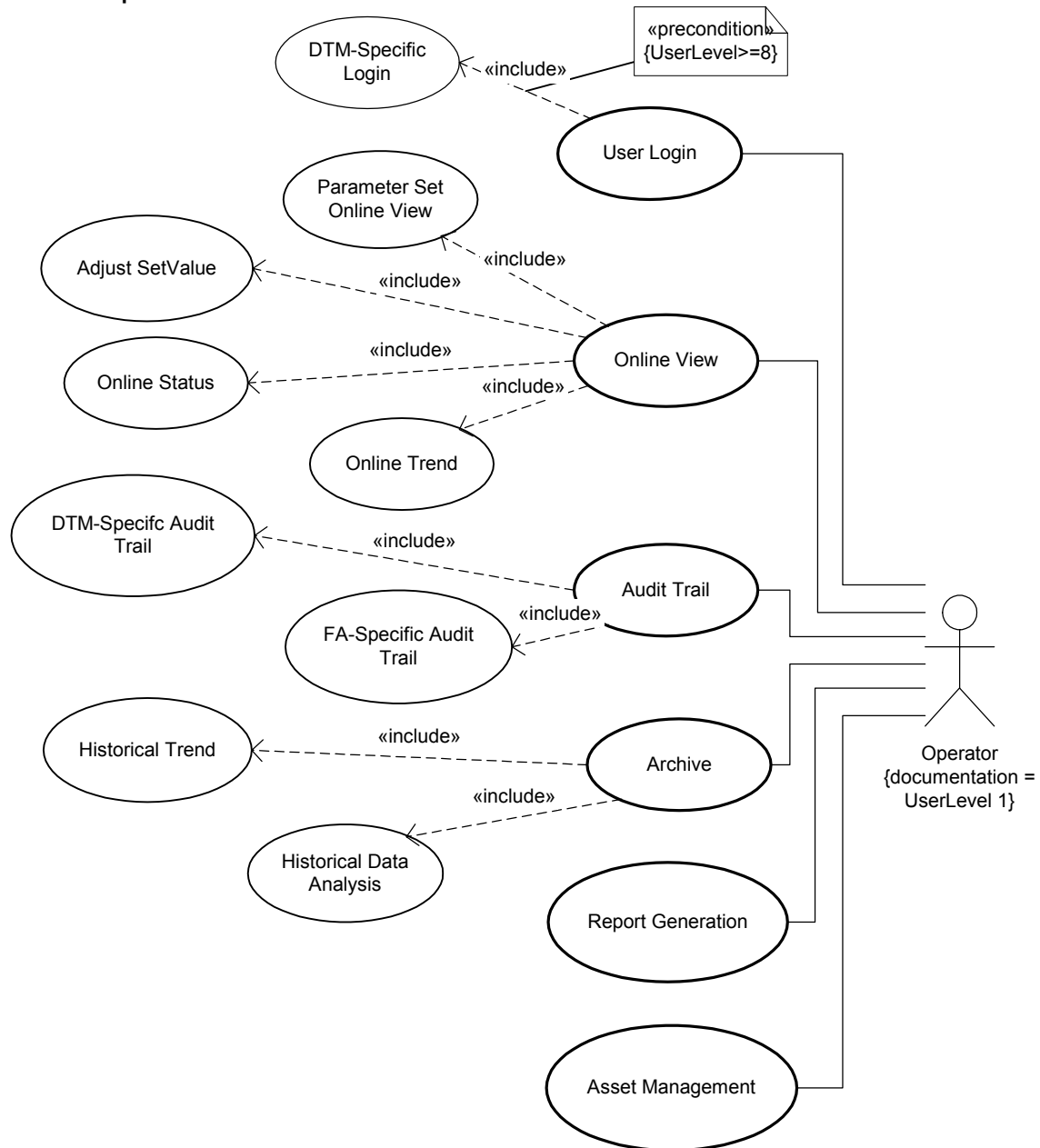


Figure 5: Use case - Operation

Use Case:		User Login
Brief Description:		A person of specified actor type logs into the frame-application. If verification fails the person may act as an "Observer" actor only.
GUI:		Part of the frame-application
Print:		No support
Device Connection:		Not established
UserLevel	Observer:	Depends on the frame-application
	Operator:	Accessible
	Maintenance:	
	Planning Eng.:	
UserFlag	Administrator:	
	OEM Service:	Accessible with extended functionality (see below)
Included Use Case:		DTM-Specific Login
Brief Description:		Access to manufacturer specific information e.g. production codes, changes log
GUI:		Part of the DTM
Print:		No support
Device Connection:		Typically needs an established connection
UserFlag	OEM Service:	Accessible only for OEM Service

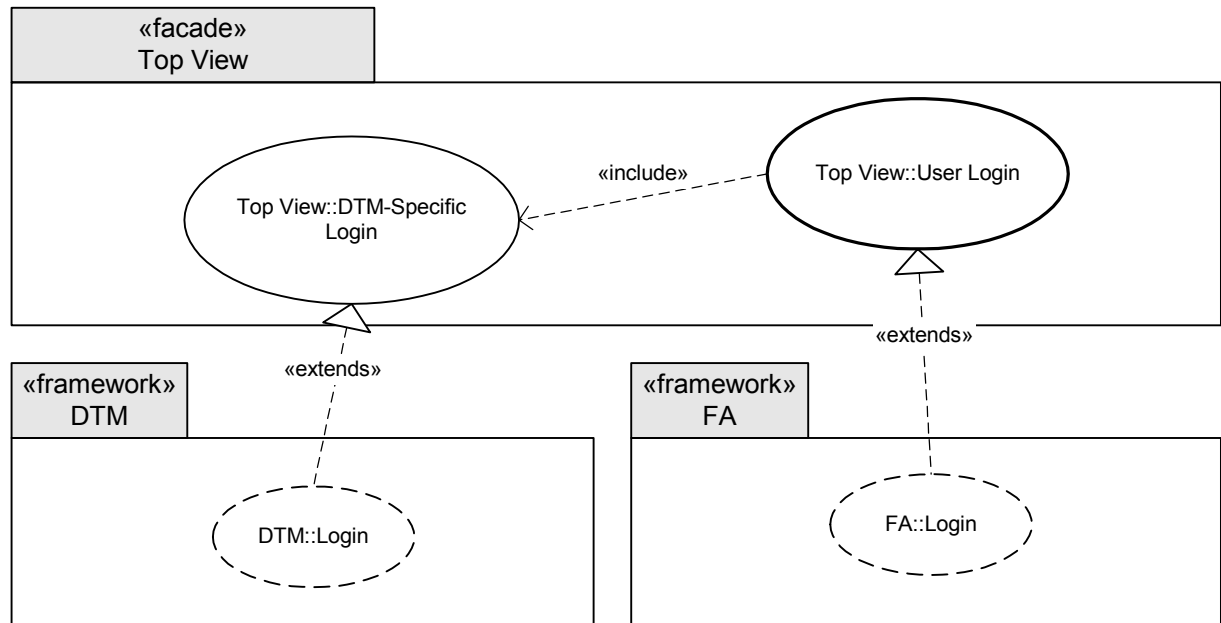


Figure 6: Realization of use case "User Login"

Use Case:		Online View
Brief Description:		This use case offers a complete set of operations for viewing of device parameters and status.
GUI:		The GUI is part of the DTM. The frame-application can offer online views for a group of devices.
Print:		Online View should always support print function to create documentation.
Device Connection:		Needs an established connection to one or more devices (Adjust SetValue may influence device data)
UserLevel	Observer:	No access
	Operator:	Accessible
	Maintenance:	
	Planning Eng.:	
UserFlags:		No changes
Included Use Case:		Parameter Set Online View
Brief Description:		Enables the actor to load parameters from the device for viewing and documenting (printing).
Included Use Case:		Adjust SetValue
Brief Description:		Enables the actor to adjust the SetValue of a device (e.g. positioner, controllers).
Included Use Case:		
Brief Description:		Use case for viewing and analyzing the current device status.
Included Use Case:		Online Trend
Brief Description:		Use case for generating and watching trends of dynamic online values.

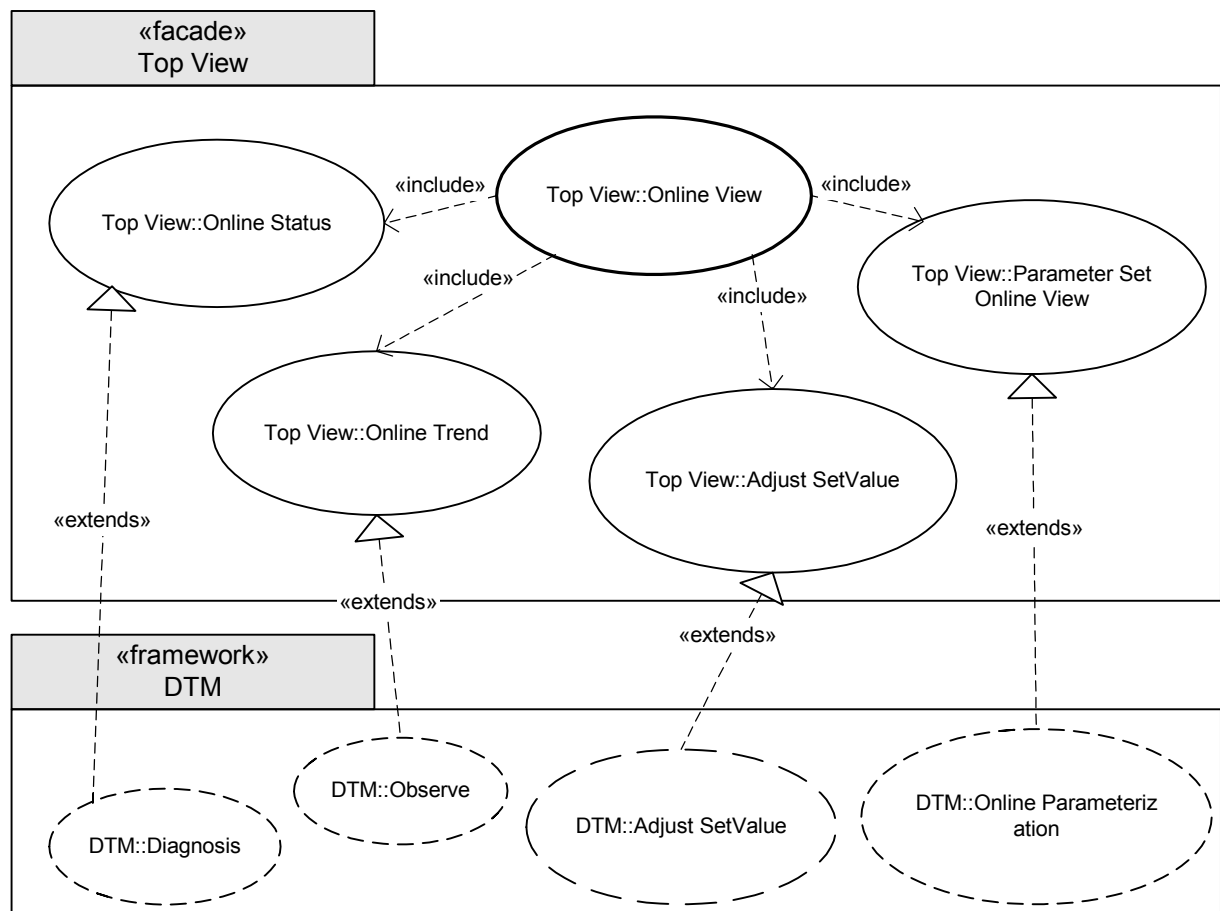


Figure 7: Realization of use case "Online View"

Audit Trail	
Brief Description:	Use case to enable the actor viewing and deleting audit trail data.
GUI:	The component, which supports audit- trail, has to provide an adequate GUI.
Print:	Printing for documentation purpose as a need for audit trails and therefore has to be supported.
Device Connection:	No connection necessary
UserLevel	Observer: No access
	Operator: Accessible for viewing only
	Maintenance:
	Planning Eng.: View, export and delete
UserFlags:	No changes
Included Use Case:	DTM-Specific Audit Trail
Brief Description:	Every DTM might support an audit trail on its own.
Included Use Case:	FA-Specific Audit Trail
Brief Description:	The frame-application may implement a system global audit trail using internal data and named DTM properties exported from the DTM via the IDTMAuditTrailEvents interface.

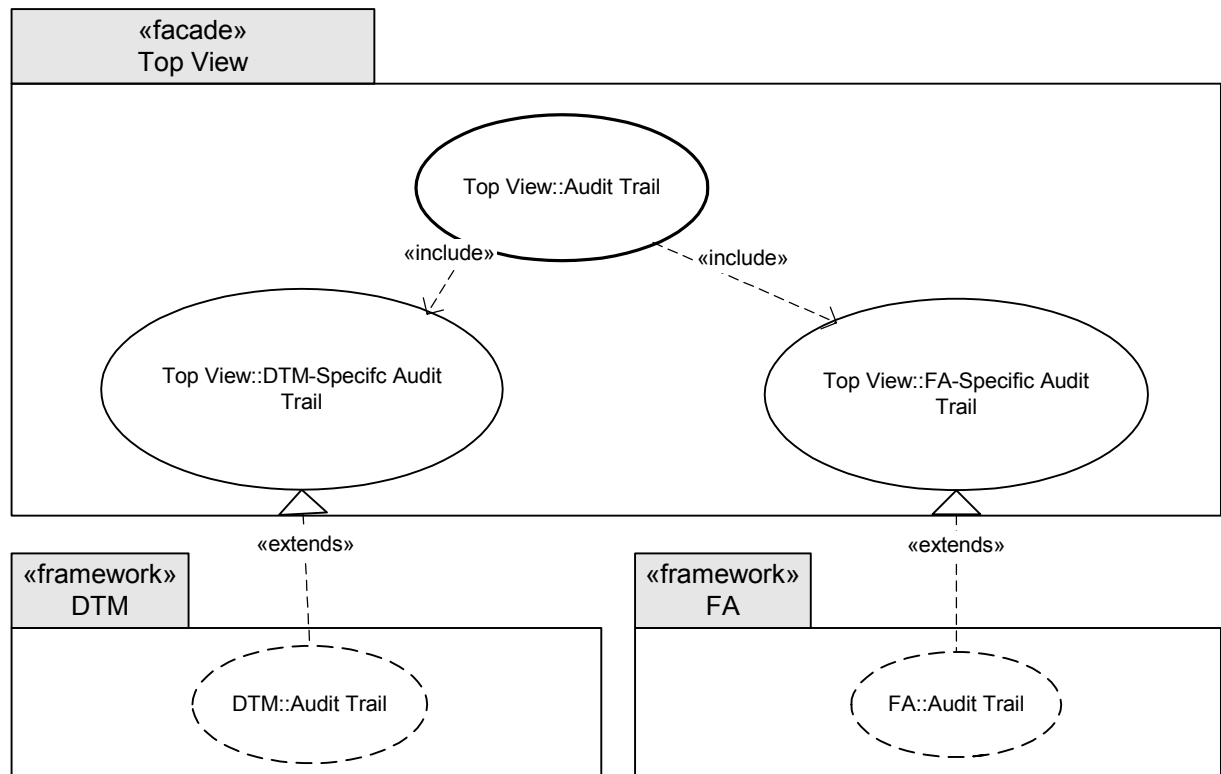


Figure 8: Realization of use case "Audit Trail"

Use Case:	Archive								
Brief Description:	The "Archive" use case describes the access to a frame-application archive for viewing and data analysis.								
GUI:	The component, which supports archive functions, has to provide an adequate GUI.								
Print:	Every archive component should support printing too.								
Device Connection:	Not necessary								
UserLevel:	<table> <tr> <td>Observer:</td><td>No access</td></tr> <tr> <td>Operator:</td><td>Accessible for viewing only</td></tr> <tr> <td>Maintenance:</td><td></td></tr> <tr> <td>Planning Eng.:</td><td>View, export and delete</td></tr> </table>	Observer:	No access	Operator:	Accessible for viewing only	Maintenance:		Planning Eng.:	View, export and delete
Observer:	No access								
Operator:	Accessible for viewing only								
Maintenance:									
Planning Eng.:	View, export and delete								
UserFlags:	No changes								
Included Use Case:	Historical Trend								
Brief Description:	The archive operations may support visualization of historical data (for example trending).								
Included Use Case:	Historical Data Analysis								
Brief Description:	Some frame-applications and DTMs may support extended operations to analyze historical data.								

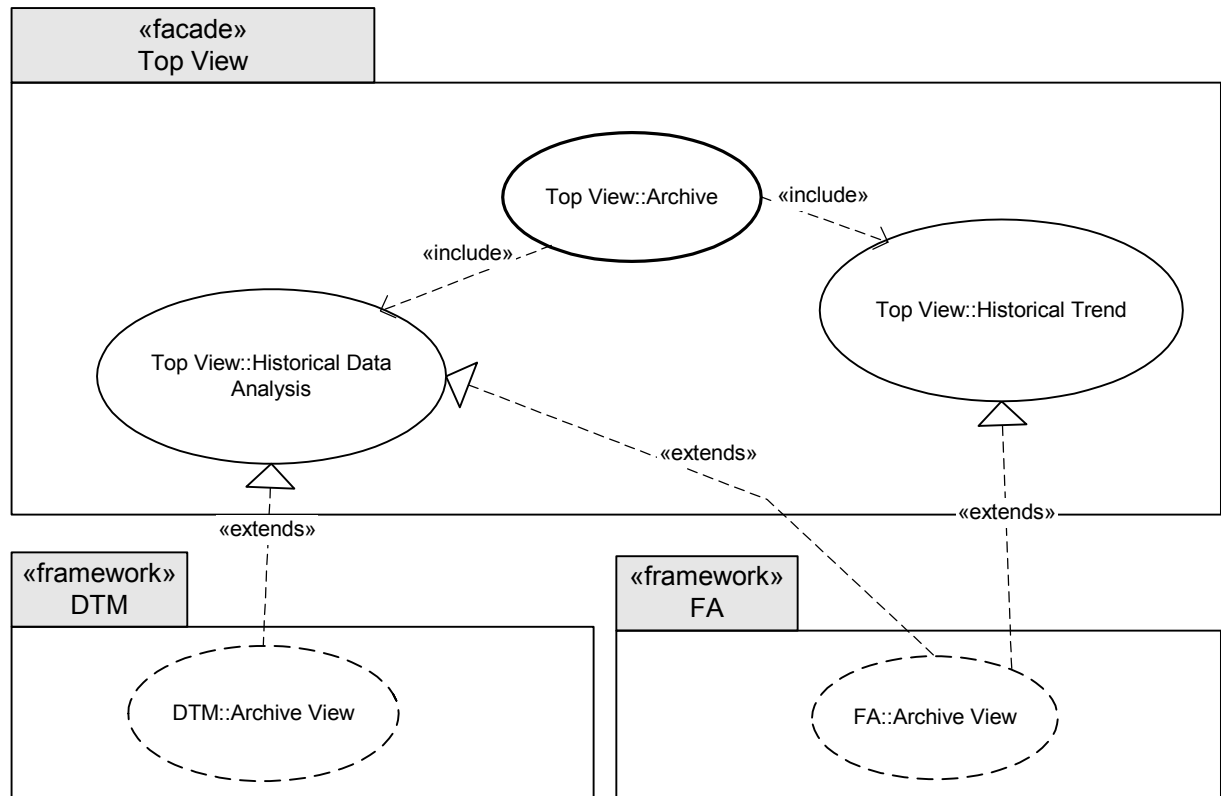


Figure 9: Realization of use case "Archive"

Use Case:	Report Generation
Brief Description:	The print function of a use case can normally be started from its GUI. In addition to printing the actual view of a GUI some frame-applications may implement a configurable documentation mechanism which enables the actor to generate reports. This report may be generated by combining the prints of several use cases or several devices. For example a report could be generated containing "Online View", "Audit Trail" and "Online Operation" printouts for one device or a report may be generated containing the configuration of a HART multiplexer and its clients.
GUI:	Frame-application
Print:	Frame-application in combination with one or more DTMs
Device Connection:	Depends on the type of report
UserLevel, UserFlag:	The printed information may depend on user level and user flags.

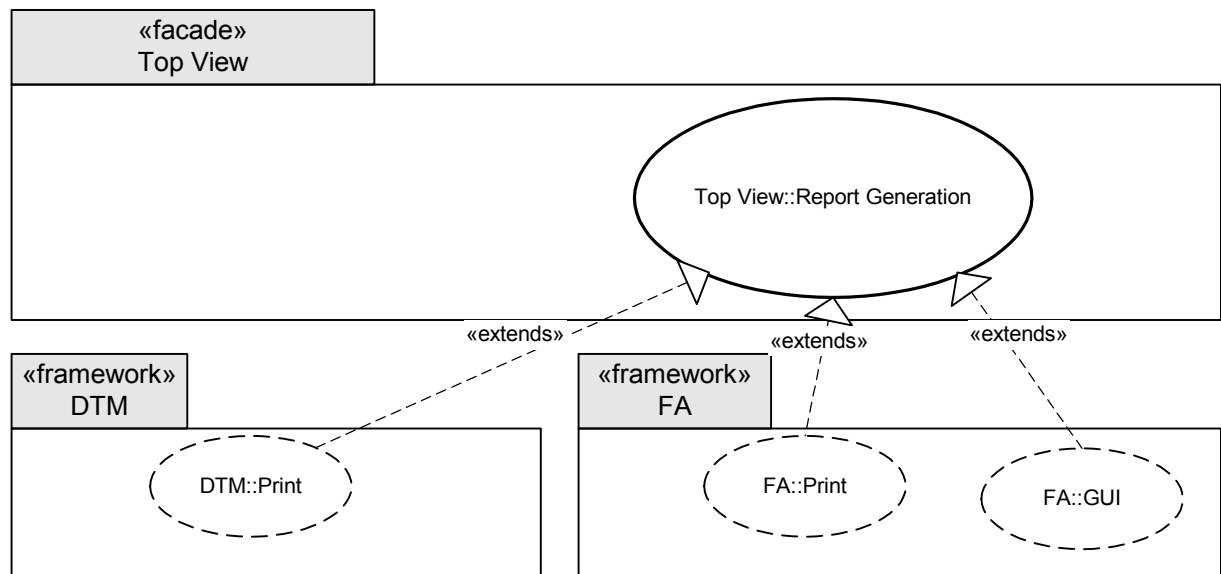


Figure 10: Realization of use case "Report Generation "

Use Case:		Asset Management
Brief Description:		This use case has not been discussed in detail up to now.
GUI:		The component, which supports archive functions, has to provide an adequate GUI.
Print:		Every component supporting this use case should support printing too.
Device Connection:		Not necessary
UserLevel	Observer:	No access
	Operator:	Accessible for viewing only
	Maintenance:	
	Planning Eng.:	View, export and delete
UserFlags:		No changes

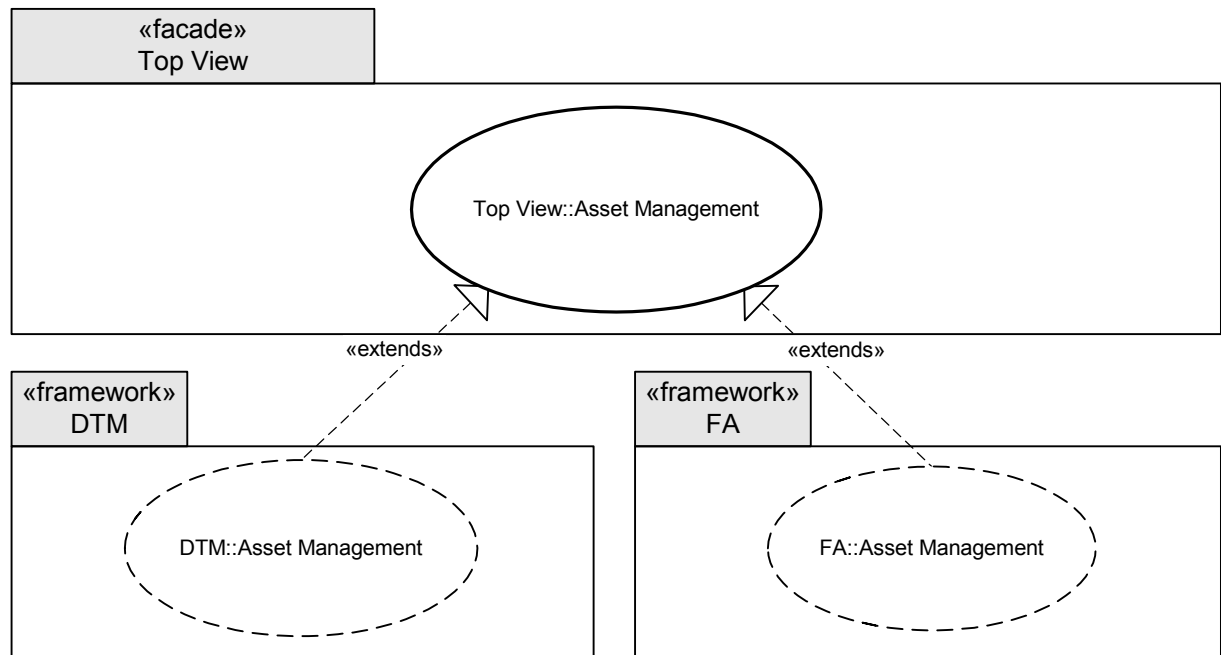


Figure 11: Realization of use case "Asset Management"

4.2.3 Maintenance

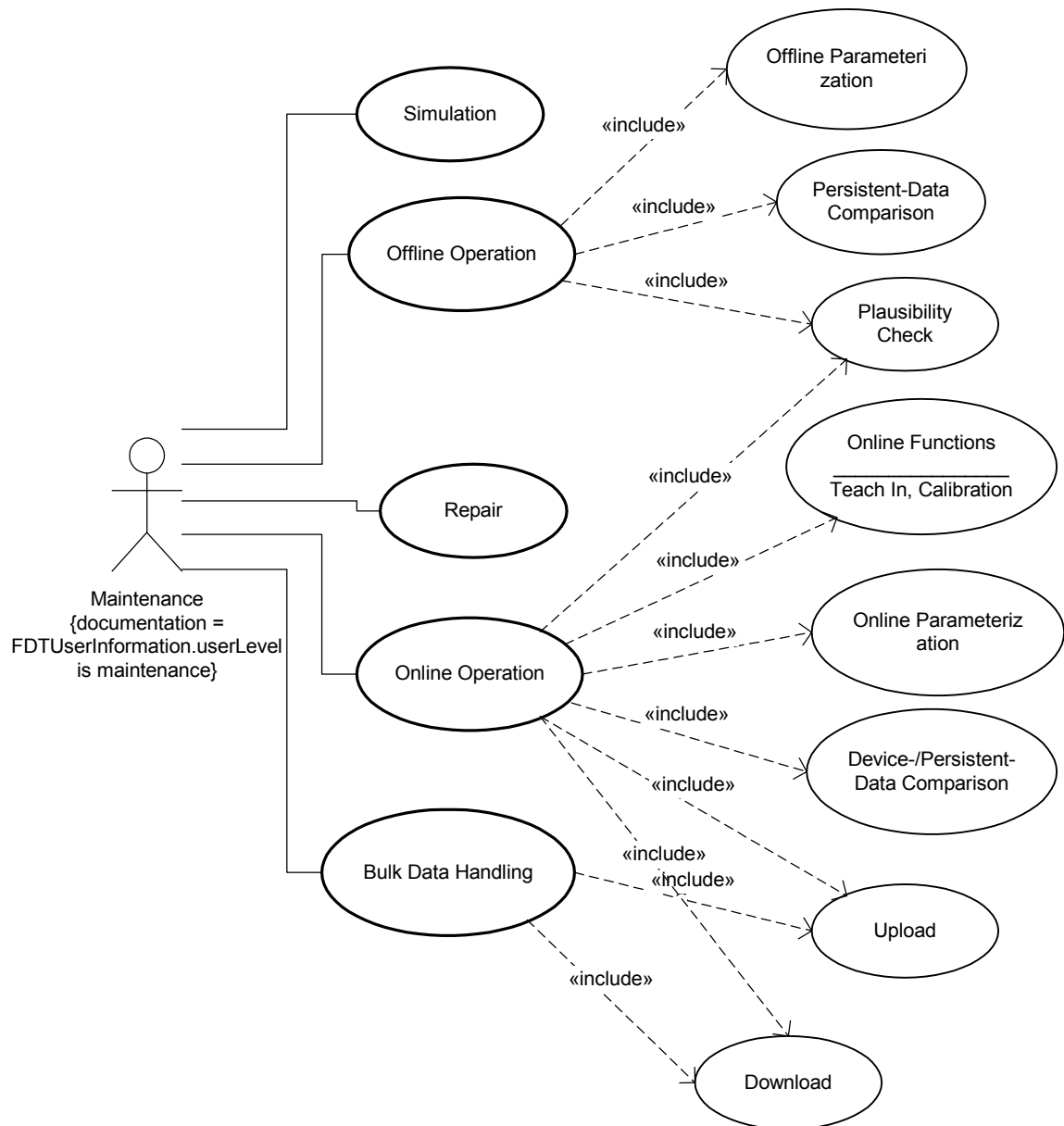


Figure 12: Use case - Maintenance

Use Case:		Simulation
Brief Description:		This use case simulates the behavior of a device. Therefore the actor is allowed to perform temporary changes within the device parameters, like forcing the device to set a fixed current analog output.
GUI:		DTM-specific
Print:		DTM-specific
Device Connection:		Must have a connection established
UserLevel	Observer:	No access
	Operator:	
	Maintenance:	Accessible
	Planning Eng.:	
UserFlags:		No changes

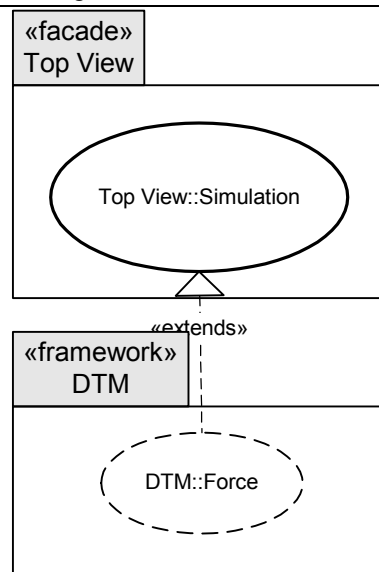


Figure 13: Realization of use case "Simulation"

Use Case:		Offline Operation
Brief Description:		This use case includes all operations which do not require an established connection to a device. Only for up- and download a connection to a device may be temporarily established.
GUI:		DTM and frame-application
Print:		DTM
Device Connection:		Depends on the included use case
UserLevel:	Observer:	No access
	Operator:	
	Maintenance:	Accessible
	Planning Eng.:	
UserFlags:		No changes
Included Use Case:		Plausibility Check
Brief Description:		Every time the parameter values are changed a plausibility check is automatically performed. As long as the plausibility check fails, the data cannot be saved to the database or written to the device. There may be two options depending on rules or application environment: <ul style="list-style-type: none"> • After display of a warning, inconsistent data may be stored • For safety reasons after display of a warning writing of data is prohibited.
Users:		The plausibility check is used from all actors. The plausibility check may be more tolerant for actors of higher level.
Included Use Case:		Offline Parameterization
Brief Description:		The user may change parameter values. The changes will only affect data in the frame-application database after stored as persistent data. Data should be signed as transient data until they are stored.
GUI:		DTM
Print:		DTM
Device Connection:		No connection necessary
Included Use Case:		Persistent Data Comparison
Brief Description:		Allows the user to compare the offline parameter set with parameter sets of other instances, of device default or of project default parameter sets. The data sets may be editable and data may be transferred from one data set to the other.
GUI:		DTM
Print:		May be supported by the DTM
Device Connection:		Not necessary

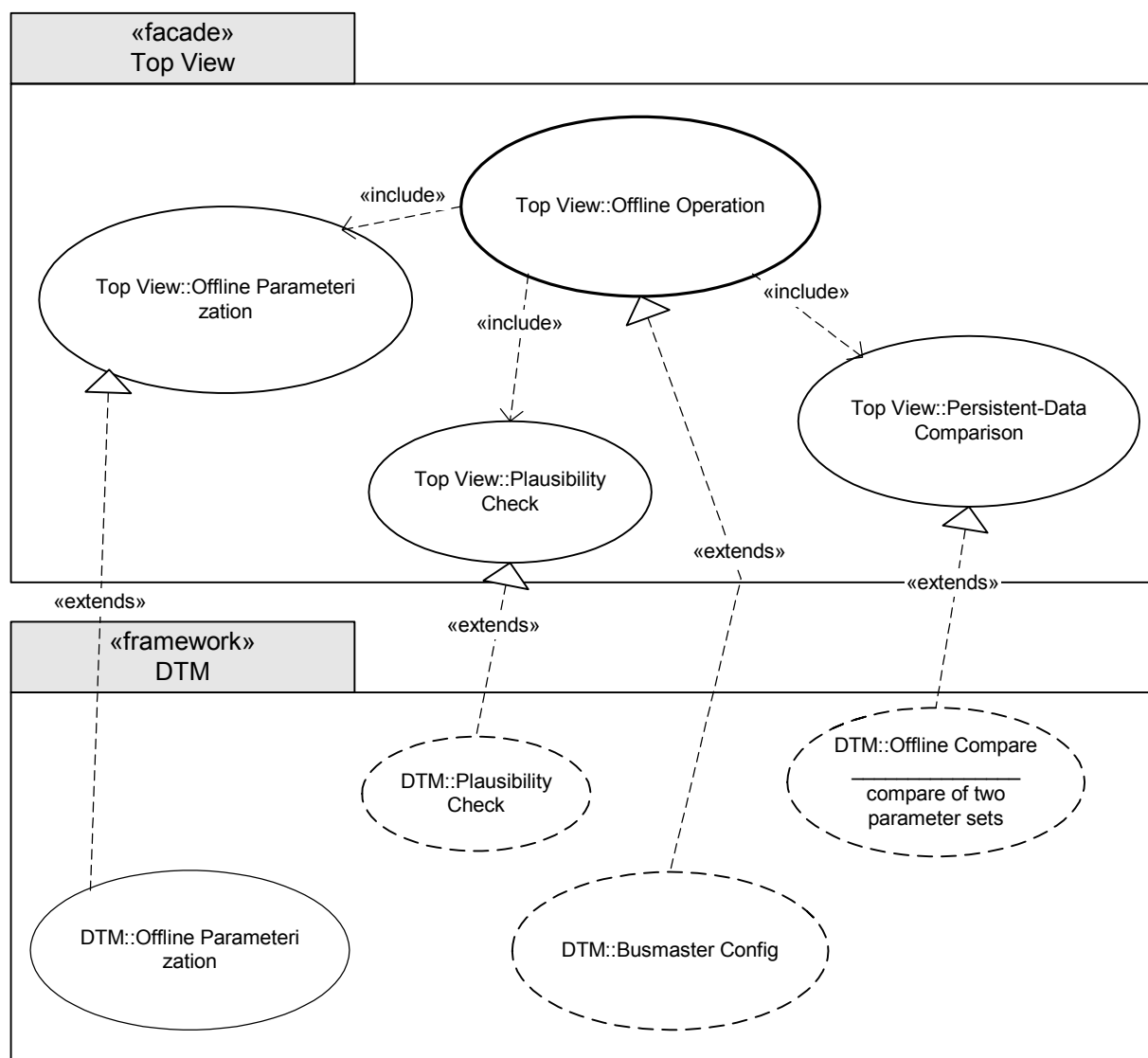


Figure 14: Realization of use case "Offline Operation"

Use Case:		Repair
Brief Description:		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.
UserLevel	Observer:	No access
	Operator:	
	Maintenance:	Accessible
	Planning Eng.:	
UserFlag	Administrator:	No changes
	OEM Service:	Accessible with extended functionality (see below)

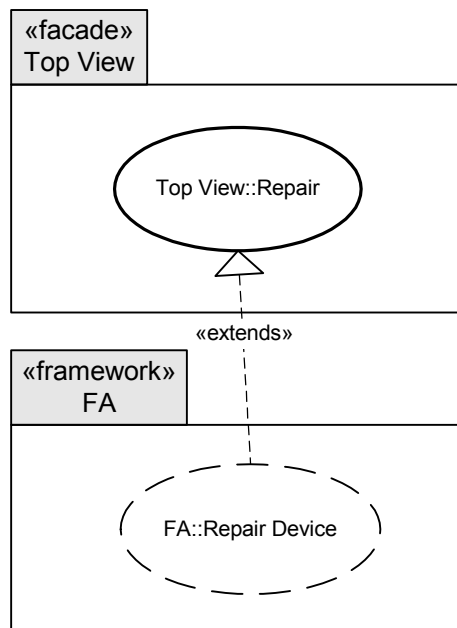


Figure 15: Realization of use case "Repair"

Use Case:		Online Operation
Brief Description:		This use case includes all operations which are performed directly with the device.
GUI:		DTM-specific
Print:		DTM-specific
Device Connection:		Online or offline
Users:	Operator:	No access
UserLevel	Observer:	No access
	Operator:	
	Maintenance:	Accessible
	Planning Eng.:	Fully accessible excluding OEM-specific parameter
UserFlag	Administrator:	No changes
	OEM Service:	Accessible with extended functionality (see below)
Included Use Case:		Online Functions
Brief Description:		The "Online Functions " use case offers all procedures which include direct communication with the device. The use case may include simple procedures like resetting but also more complex procedures with several sections like calibration or teach in.
Included Use Case:		
Brief Description:		When this use case starts all parameters are read from the device and exposed to the user within a GUI. Within the GUI the user may change parameters, depending on the user level. The device is updated immediately if the changed parameterization is in a verified state.
Included Use Case:		Device-/Persistent Data Comparison
Brief Description:		This use case gives the user the possibility to compare persistent and device data. The user may edit data and copy between these two sources.
Included Use Case:		Plausibility Check
Brief Description:		Every time the device parameters or the configuration data is changed a plausibility check is automatically performed. As long as the plausibility check fails, the data cannot be written to the device.
UserLevel, UserFlag:		All actors use the plausibility check. The plausibility check may be more tolerant for actors of higher level.
Included Use Case:		Upload
Brief Description:		Reads the whole parameter set from the device into the frame-application database. An upload started by an "OEM Service" actor may upload additional OEM parameters.
GUI:		If the frame-application supports up- and download for a group of devices, the frame-application may offer a GUI for a better control of the upload process.
Print:		No support
Device Connection:		Needs a temporary connection
Included Use Case:		Download
Brief Description:		Like upload, but in the opposite direction.
GUI:		If the frame-application supports up- and download for a group of devices, the frame-application may offer a GUI for a better control of the download process.
Print:		No support
Device Connection:		Needs a temporary connection

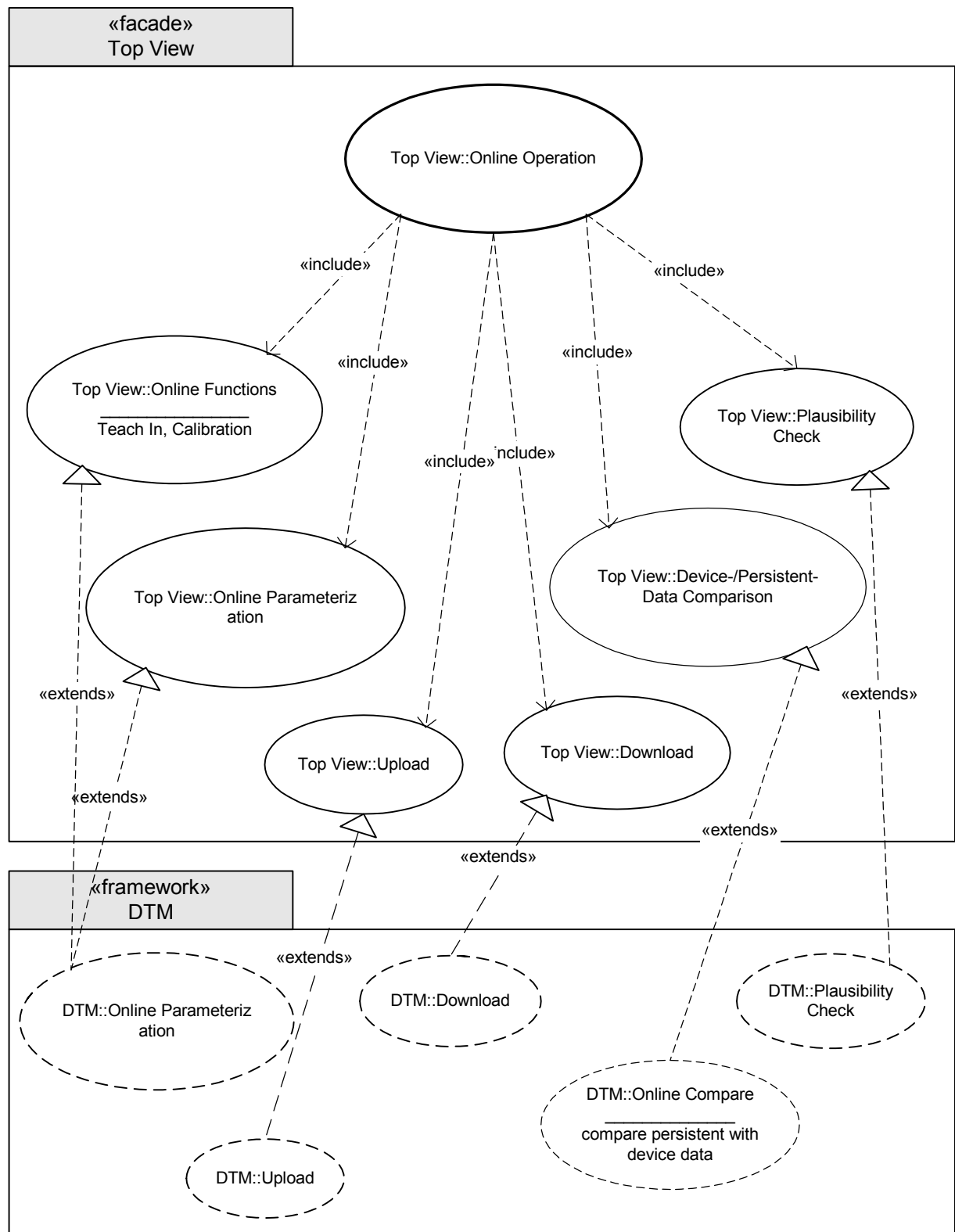


Figure 16: Realization of use case "Online Operation"

Use Case:	Bulk Data Handling	
Brief Description:	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.	
GUI:	Frame-application	
Print:	Not supported	
Device Connection:	Needs a connection	
UserLevel:	Observer:	No access
	Operator:	
	Maintenance:	Accessible
	Planning Eng.:	
UserFlags:	No changes	
Included Use Case:	Upload	
Brief Description:	Reads the whole parameterization from the devices of the group into the frame-application database.	
Included Use Case:	Download	
Brief Description:	Like upload, but in the opposite direction.	

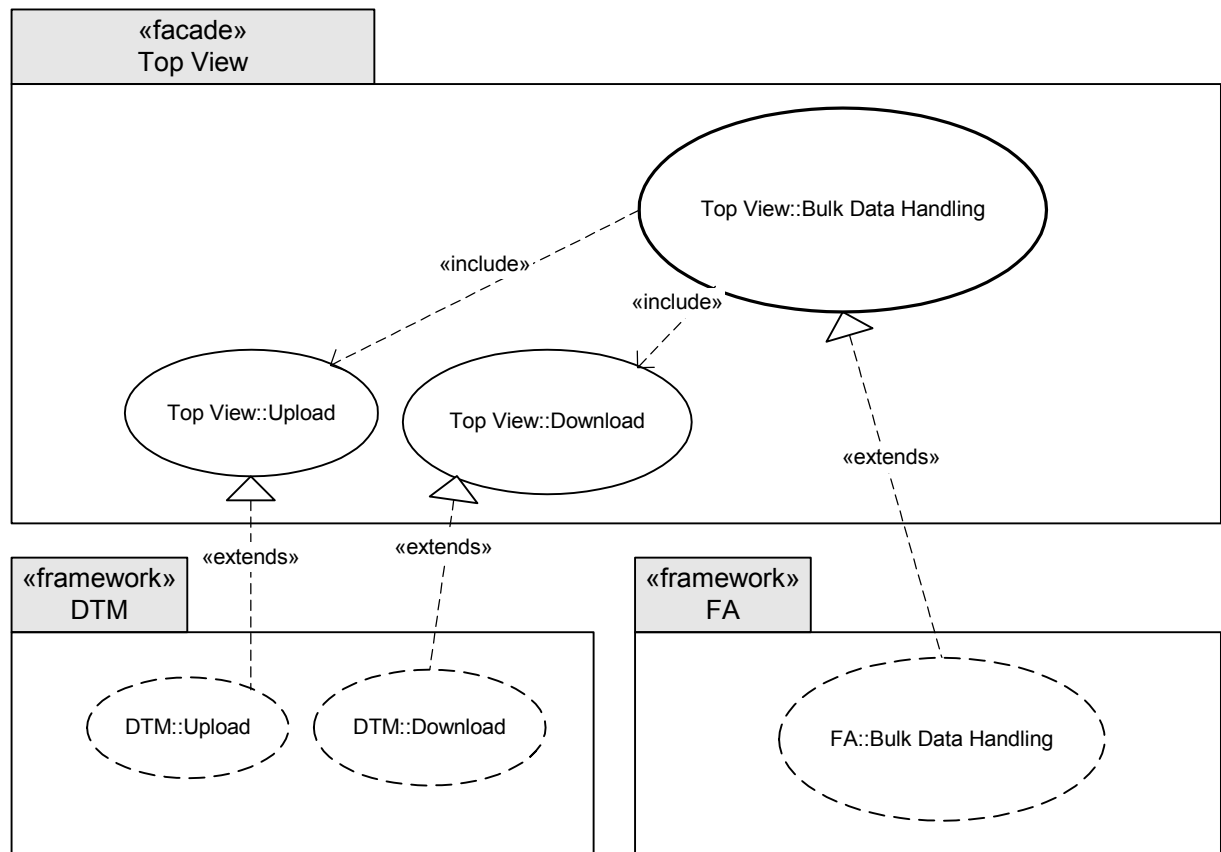


Figure 17: Realization of use case "Bulk Data Handling"

4.2.4 Planning

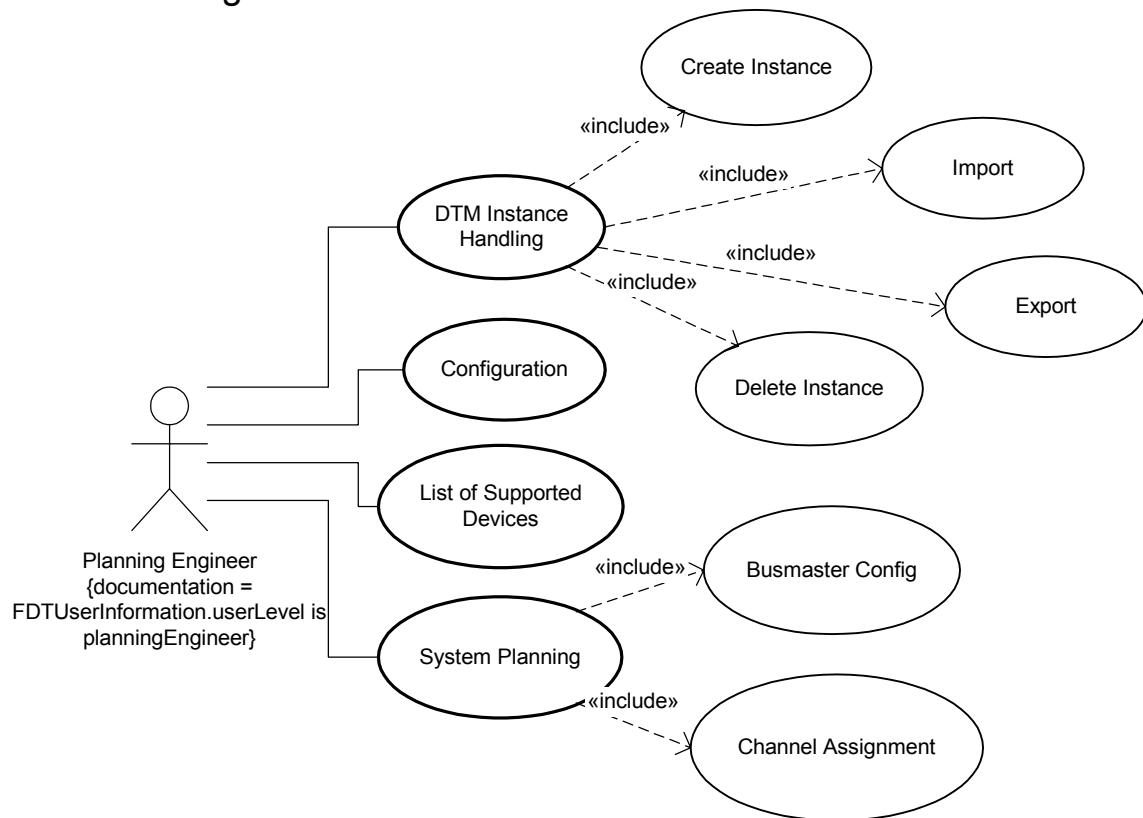


Figure 18: Use case - Planning Engineer

Use Case:		DTM Instance Handling
Brief Description:		With this use case a "Planning Engineer" actor can handle (E.g. instantiate, import, export, delete) a device instance in the frame-application.
GUI:		Frame application and DTM
Print:		Not supported
Device Connection:		Not necessary
UserLevel	Observer:	No access
	Operator:	
	Maintenance:	
	Planning Eng.:	Accessible
UserFlags:		No changes
Included Use Case:		Create Instance
Brief Description:		In this use case a new device instance can be created and placed into the project. After creation of a new device instance, the device parameter set must be instantiated. This action is performed by the DTM.
GUI:		Frame application and DTM (if it supports device default selection)
Included Use Case:		Import
Brief Description:		The parameter set may be instantiated by importing data from a database (for example a template database) or by copying the parameter set from an already instantiated and verified device.
GUI:		FA
Included Use Case:		Export
Brief Description:		To copy data from one device instance to another, the device instance data can be exported.
GUI:		FA
Included Use Case:		Delete Instance
Brief Description:		Deletion of a device instance
GUI:		FA

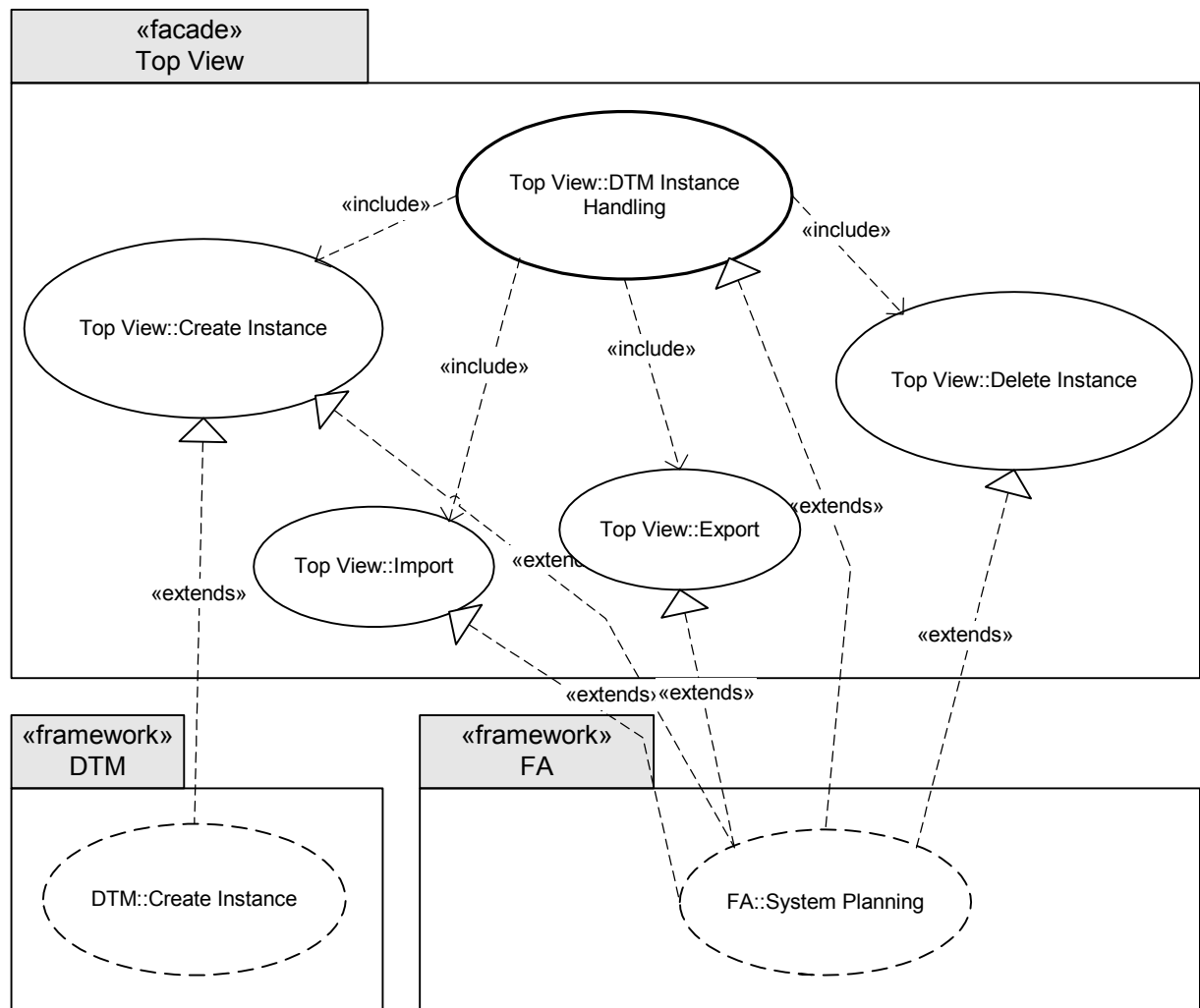


Figure 19: Realization of use case "DTM Instance Handling"

Use Case:	System Planning	
Brief Description:	Use case to perform all necessary actions for the editing of topology information.	
GUI:	Frame-application and DTM	
Print:	Frame-application and DTM	
Device Connection:	Not necessary	
UserLevel:	Observer:	No access
	Operator:	
	Maintenance:	
	Planning Eng.:	Accessible
UserFlags:	No changes	
Included Use Case:	Bus master Configuration	
Brief Description:	Use case for configuration of bus master DTMs	
Included Use Case:	Channel Assignment	
Brief Description:	Use case for editing the FDT channel assignments	

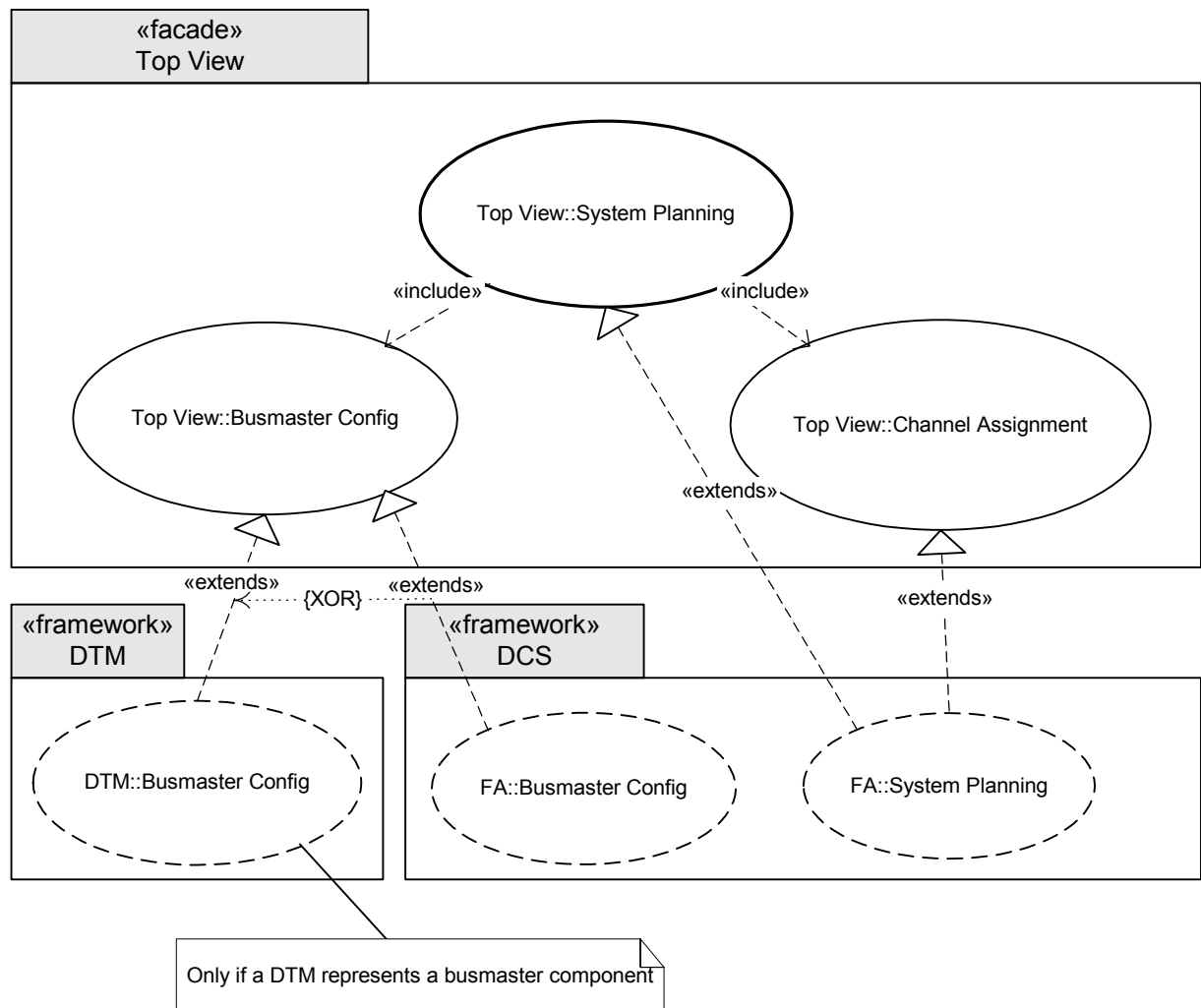


Figure 20: Realization of use case "System Planning"

Use Case:		List of Supported Devices
Brief Description:		In this use case a list of all supported devices within a project can be viewed and edited. A “ Planning Engineer ” actor can select a device from this list to create a new device instance. An actor with the “ Administrator ” actor flag set has access to change this list.
GUI:		FA
Print:		No support
Device Connection:		No connection
UserLevel	Observer:	No access
	Operator:	
	Maintenance:	
	Planning Eng.:	Accessible for viewing only
UserFlag	Administrator:	Accessible for editing
	OEM Service:	No changes

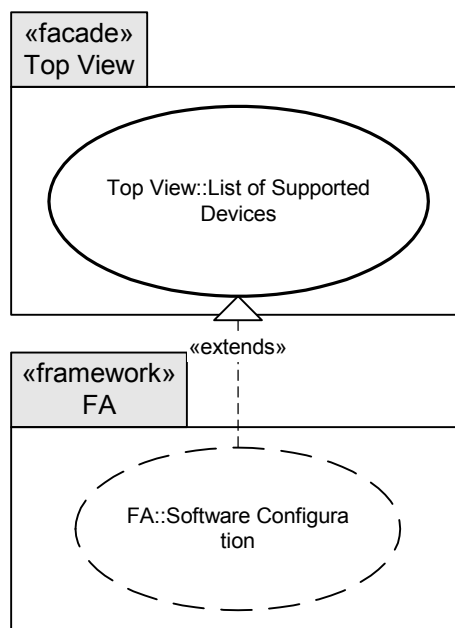


Figure 21: Realization of use case “List of Supported Devices”

Use Case:		Configuration
Brief Description:		This use case enables the “ Planning Engineer ” actor to configure a complex device.
GUI:		DTM
Print:		May be supported
Device Connection:		Must not have a connection established
UserLevel:	Observer:	No access
	Operator:	
	Maintenance:	
	Planning Eng.:	Accessible
UserFlags:		No changes

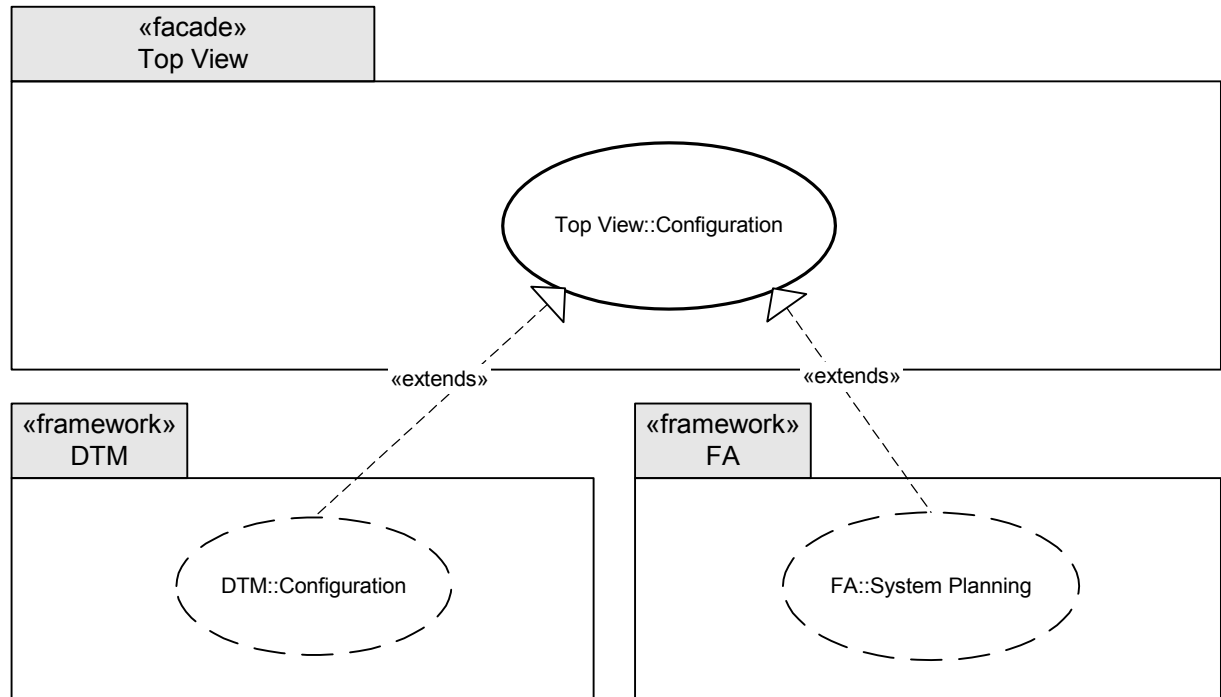


Figure 22: Realization of use case “Configuration”

4.2.5 OEM Service

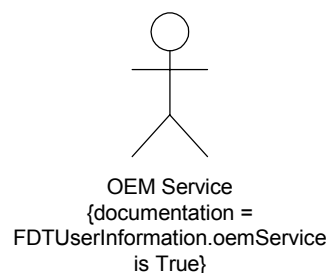


Figure 23: Use case - OEM Service

The “[OEM Service](#)” actor may only have extended access to use cases of other actor roles.

4.2.6 Administrator

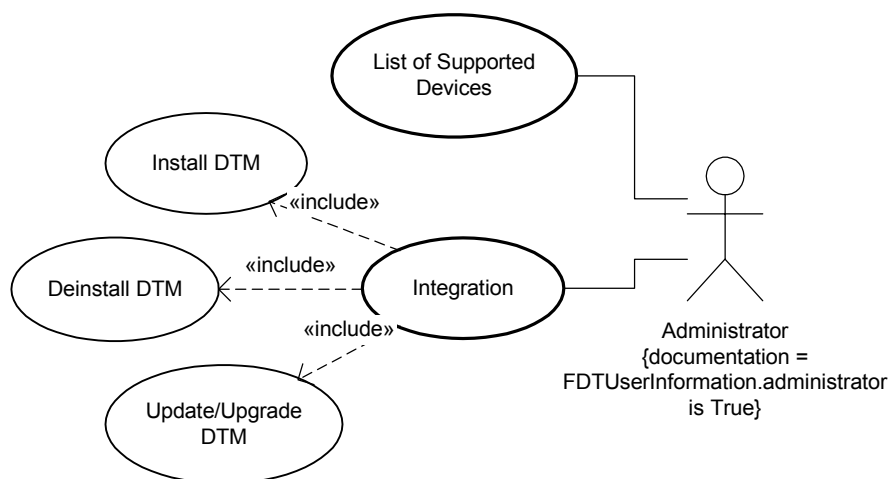


Figure 24: Use case – Administrator

Use Case:		Integration
Brief Description:		This use case enables the actor to install, deinstall or upgrade new DTMs. Installation and deinstallation are not controlled by the frame-application.
GUI:		The Frame application may support the management of DTMs.
Print:		Not supported
Device Connection:		Must not have a connection established
Users:	Operator:	Not accessible
	Maintenance:	
	Planning Eng.:	
	Administrator:	Accessible
	OEM Service:	Not accessible
Included Use Case:		Install
Brief Description:		Installation of a new DTM.
GUI:		Responsibility of the DTM
Included Use Case:		Deinstall
Brief Description:		dito
GUI:		responsibility of the DTM
Included Use Case:		Update/Upgrade
Brief Description:		dito
GUI:		responsibility of the DTM

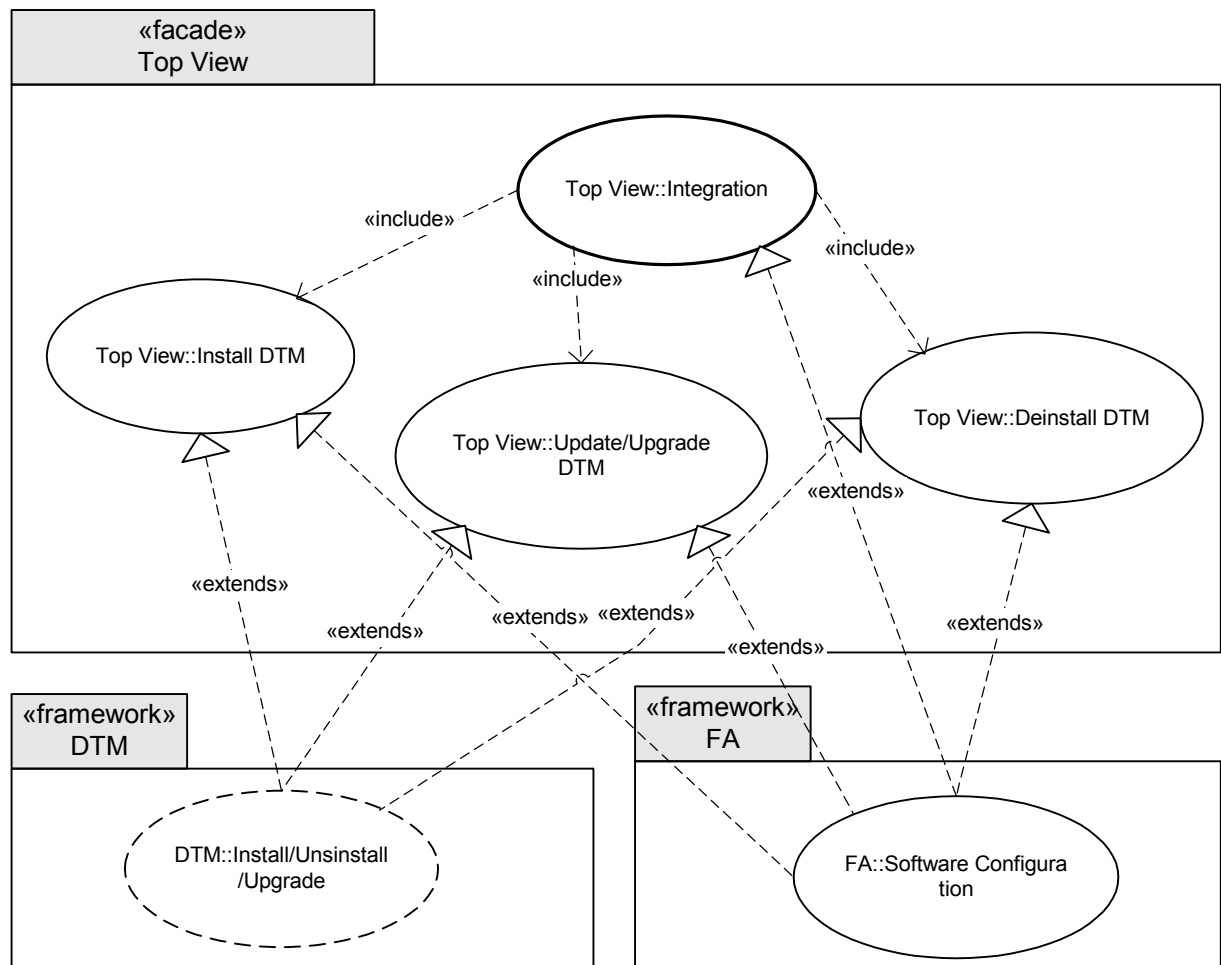


Figure 25: Realization of use case "Integration"

4.3 DTM Realization Elements

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.

These functions are then realized via the two main operations: GUI and print support.

Operations supporting or containing these functions are realized by the corresponding main operations.

The DTM-operations are shown in schematic representation in the following diagram:

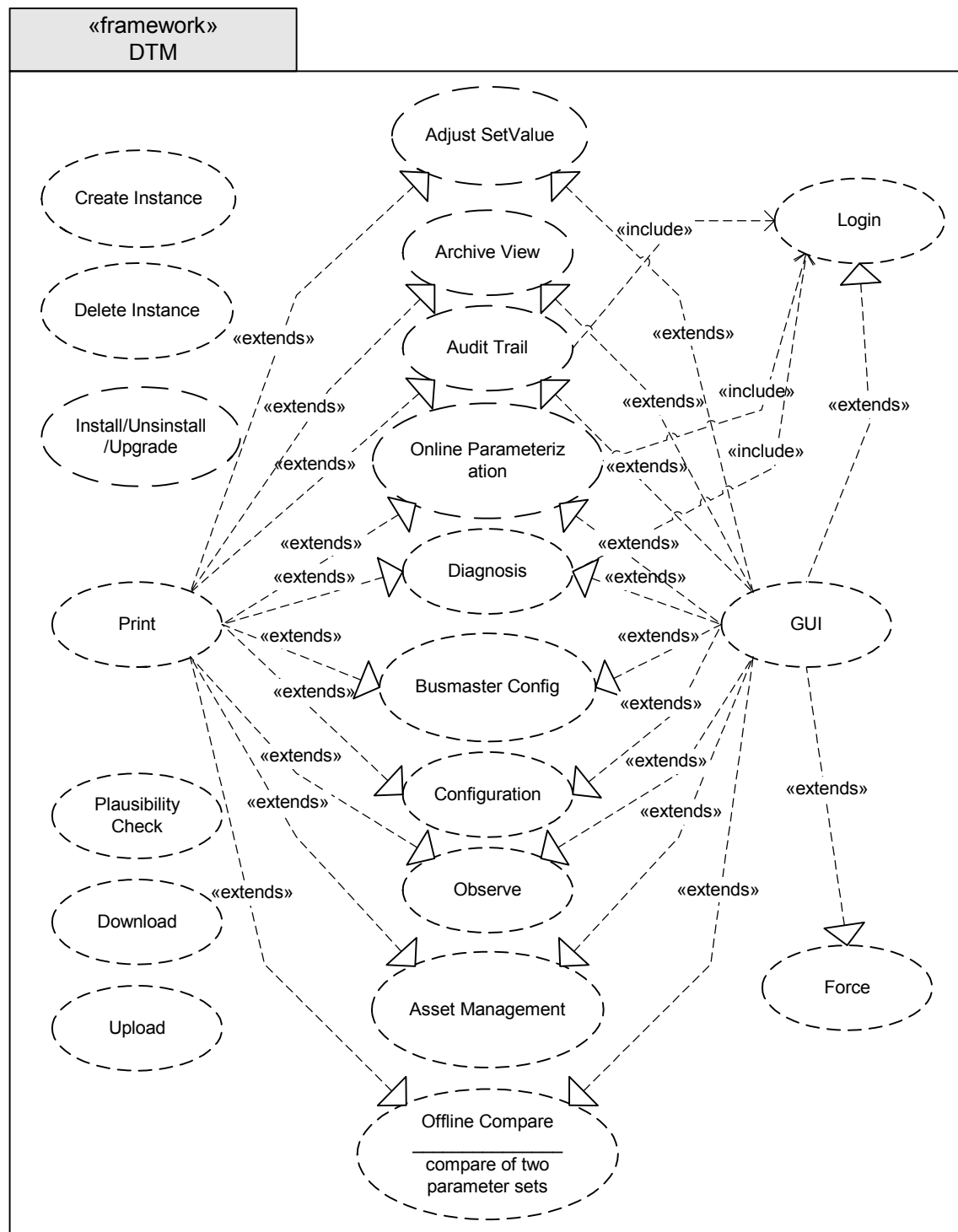


Figure 26: DTM Collaboration Architecture

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 and their DTMS-context-related method of operation.
- Extends Use Case(s): Listing of all Use Cases whose realization involves this operation.
- Invocation: Description showing how this operation is called within the FDT.
- Operation ID: Identification number for the setup of the application context. All graphical interfaces, which are directly requested by frame-application, are configured via the application context. Identification numbers are only assigned to operations which are directly started by the frame-application.

Operation:	GUI (Main Operation)
Brief Description:	<p>The GUI is a main operation of the DTM and corresponds to a summary of all graphical interfaces offered to the user by the DTM. A distinction is essentially made between two types of user interfaces as the GUI is observed:</p> <ol style="list-style-type: none"> 1. Interfaces that are directly called by the frame-application: Examples of these user interfaces are namely configuration, parameterization, io-data, or simulation interfaces. The configuration of these interfaces is performed via a context description by means of the structure applicationId. 2. Indirectly called interfaces, like e.g. dialogs that are opened during an operation for displaying and/or editing data. <p>Only the graphical interfaces of point 1 are relevant to the observation of a component model.</p>
Extends Use Case(s):	<p>The GUI Operation of the DTM only indirectly affects the realization of Use Cases through the realization of other objects.</p> <p>The GUI Operation is indirectly involved in mostly all use cases.</p>
Invocation:	The GUI is only called indirectly by other operations when it participates in their realization.
Operation ID:	-- not supported --

Operation:	Print (Main Operation)
Brief Description:	<p>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.</p>
Extends Use Case(s):	<p>As a main operation, the print operation participates in the realization of several operations of the DTM.</p> <p>Every application context a DTM supports with a GUI, should also support printing.</p>
Invocation:	<p>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.</p> <p>The frame-application starts the Print Operation of the DTM directly for the realization of the Use Cases "Report Generation". In this case the frame-application invokes the documentation interface of the DTM.</p>
Operation ID:	-- not supported --

Operation:	Adjust SetValue
Brief Description:	Enables the actor to adjust the SetValues of a device (e.g. positioner, controllers).
Extends Use Case(s):	Adjust SetValue
Invocation:	The Adjust-SetValue-Operation is called directly by the frame-application
Operation ID:	FDTApplicationIdSchema:fdtAdjustSetValue

Operation:	Asset Management
Brief Description:	The contents of the Asset Management have not been defined yet.
Extends Use Case(s):	Asset Management
Invocation:	The Asset-Management-Operation is called directly by the frame-application
Operation ID:	FDTApplicationIdSchema:fdtAssetManagement

Operation:	Audit Trail
Brief Description:	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. As long as a frame-application-wide Audit Trail has not been specified yet, the Audit Trail Operation is only effective within the DTM and a standard interface to the frame-application does not have to be offered.
Extends Use Case(s):	Device-Specific Audit Trail
Invocation:	Is called directly by the frame-application
Operation ID:	FDTApplicationIdSchema:fdtAuditTrail

Operation:	Bus Master Configuration
Brief Description:	GUI for configuration of the bus master.
Extends Use Case(s):	Bus Master Configuration
Invocation:	Is called directly by the frame-application
Operation ID:	FDTApplicationIdSchema:fdtManagement

Operation:	Configuration
Brief Description:	The configuration operation lets a user define the structure of complex devices. Configuration is used online and offline.
Extends Use Case(s):	Configuration
Invocation:	is called directly by the frame-application
Operation ID:	FDTApplicationIdSchema:fdtConfiguration

Operation:	Create Instance
Brief Description:	This operation generates the data set a DTM requires for the management of a device and feeds it with preset data.
Extends Use Case(s):	Create Instance
Invocation:	is called directly by the frame-application
Operation ID:	-- not supported --

Operation:	Delete Instance
Brief Description:	Deletes the data
Extends Use Case(s):	Delete Instance
Invocation:	is called directly by the frame-application
Operation ID:	-- not supported --

Operation:	Diagnosis
Brief Description:	This operation realizes all diagnostic methods a DTM provides for a device check.
Extends Use Case(s):	Online Status
Invocation:	is called directly by the frame-application
Operation ID:	FDTApplicationIdSchema:fdtDiagnosis

Operation:	Download
Brief Description:	The Download Operation writes the current parameterization to the field device.
Extends Use Case(s):	Download
Invocation:	is called directly by the frame-application
Operation ID:	-- not supported --

Operation:	Force
Brief Description:	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.
Extends Use Case(s):	Simulation
Invocation:	Is called directly by the frame-application
Operation ID:	FDTApplicationIdSchema:fdtForce

Operation:	
Brief Description:	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.
Extends Use Case(s):	Install DTM, Deinstall DTM, Update/Upgrade DTM
Invocation:	Can be started directly by the frame-application or by the surface of the operating system.
Operation ID:	-- not supported --

Operation:	Login
Brief Description:	When an operation with OEM-specific functions is started by an actor of the "OEM Service" user level, free accessibility to those functions can be enabled by another password inquiry. This password inquiry is realized by operation "Login". 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.
Extends Use Case(s):	DTM-Specific Login
Invocation:	is called during other operations, when an "OEM Service" actor accesses operations with OEM-specific function elements.
Operation ID:	-- not supported --

Operation:	Observe
Brief Description:	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 "Online Trend" can be named as an example of such an operation.
Extends Use Case(s):	Online Trend
Invocation:	is called directly by the frame-application
Operation ID:	FDTApplicationIdSchema:fdtObserve

Operation:	Offline Compare
Brief Description:	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 IDtmDiagnosis::Compare function
Extends Use Case(s):	Persistent Data Comparison
Invocation:	is called directly by the frame-application
Operation ID:	FDTApplicationIdSchema:fdtOfflineCompare

Operation:	Offline Parameterize
Brief Description:	This operation enables editing of the instance persistent data set.
Extends Use Case(s):	Offline Parameterization
Invocation:	is called directly by the frame-application
Operation ID:	FDTApplicationIdSchema:fdtOfflineParameterize

Operation:	Online Compare
Brief Description:	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.
Extends Use Case(s):	Device-/Persistent Data Comparison
Invocation:	is called directly by the frame-application
Operation ID:	FDTApplicationIdSchema:fdtOnlineCompare

	Online Parameterization
Brief Description:	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 "Operator" in order to realize the "Parameter Set Online View" Use Case.
Extends Use Case(s):	Online Parameterization , Online Functions , Parameter Set Online View
Invocation:	is called directly by the frame-application
Operation ID:	FDTApplicationIdSchema:fdtOnlineParameterize

Operation:	Plausibility Check
Brief Description:	Operation "Plausibility Check" 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
Extends Use Case(s):	Plausibility Check
Invocation:	is only called indirectly via other operations
Operation ID:	-- not supported --

Operation:	Upload
Brief Description:	Loads the current configuration from the device to the device instance data set of the DTM
Extends Use Case(s):	Upload
Invocation:	is called directly by the frame-application
Operation ID:	-- not supported --

4.4 Frame-Application Realization Elements

The following graphical display contains any operation the frame-application must provide for the realization of the FDT Uses Cases. In addition to this chart, the operations are again listed in tabular form and introduced in a brief description explaining their function in the context of the FDT-concept.

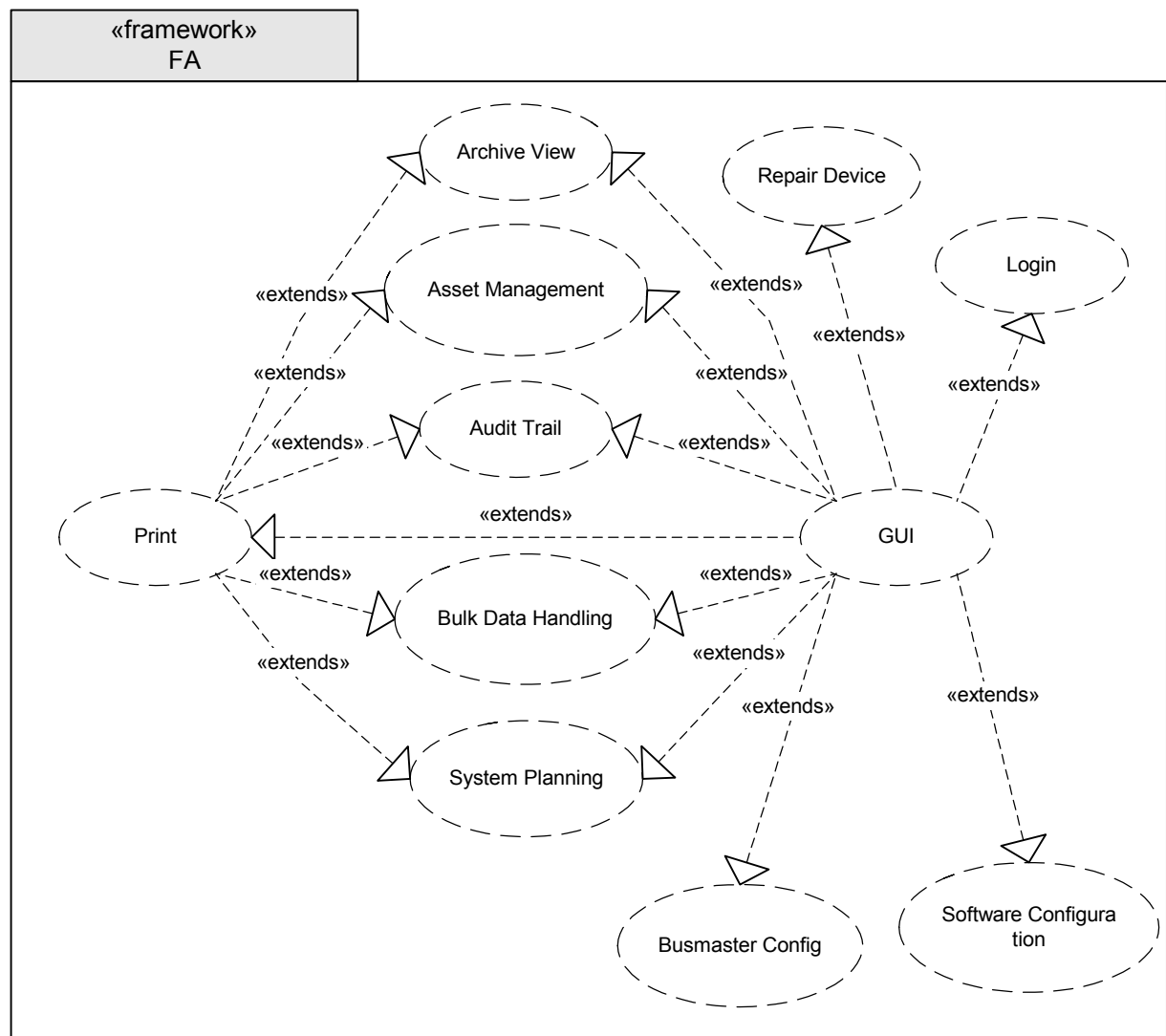


Figure 27: DTM Collaboration Architecture

Operation:	GUI (Main Operation)
Brief Description:	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.
Operation:	Print (Main Operation)
Brief Description:	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.
Operation:	Archive View
Brief Description:	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.

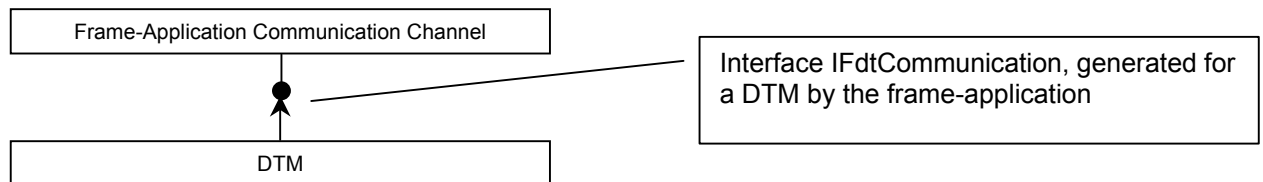
Operation:	Asset Management
Brief Description:	The Asset Management requires an extension of the DTM-interface specification of an interface which is not yet contained in the current specification. This operation is therefore a wildcard for the future realization concept.
Operation:	Audit Trail
Brief Description:	The extension of the DTM-interface for Audit Trails will be available in a later edition of the FDT-Specification. This operation is therefore used as a wildcard of the future realization concept.
Operation:	Bulk Data Handling
Brief Description:	Engineering tools 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.
Operation:	Login
Brief Description:	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 Device-Specific Login the user's role can also remain valid for several sessions.
Operation:	Repair Device
Brief Description:	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.
Operation:	Software Configuration
Brief Description:	The configuration operation lets a user define the structure of complex devices. Configuration is used online and offline.
Operation:	System Planning
Brief Description:	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.

5 FDT Sequence Charts

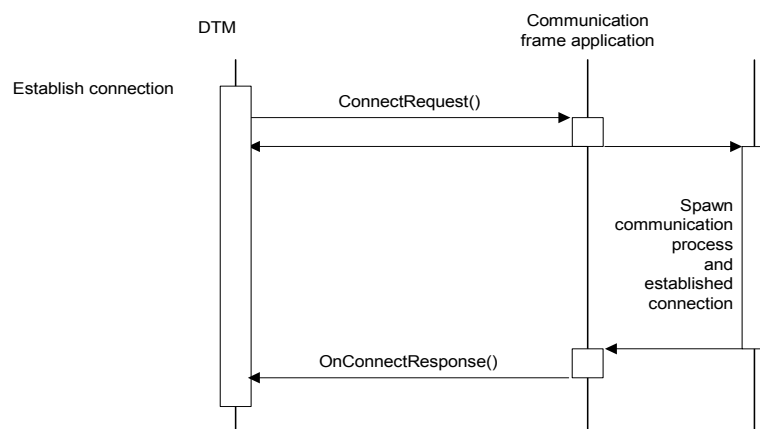
5.1 DTM Peer To Peer Communication

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.

5.1.1 Establish a Connection between DTM and Device



5.1.2 Asynchronous Connect

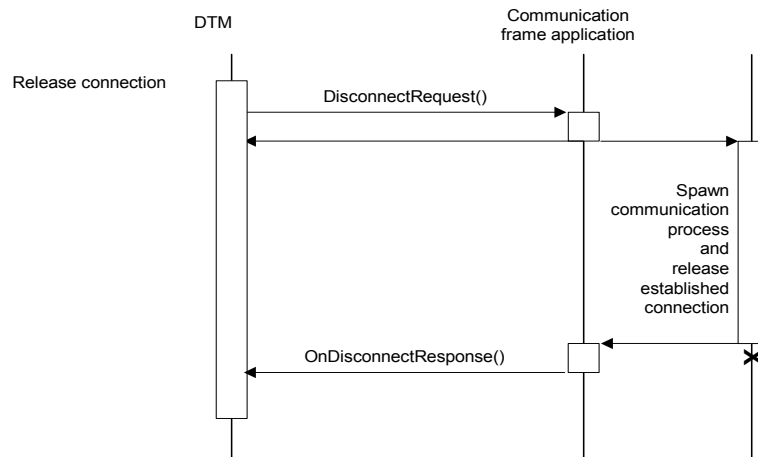


Used methods:

`IFdtCommunication::ConnectRequest()`

`IFdtCommunicationEvents::OnConnectResponse()`

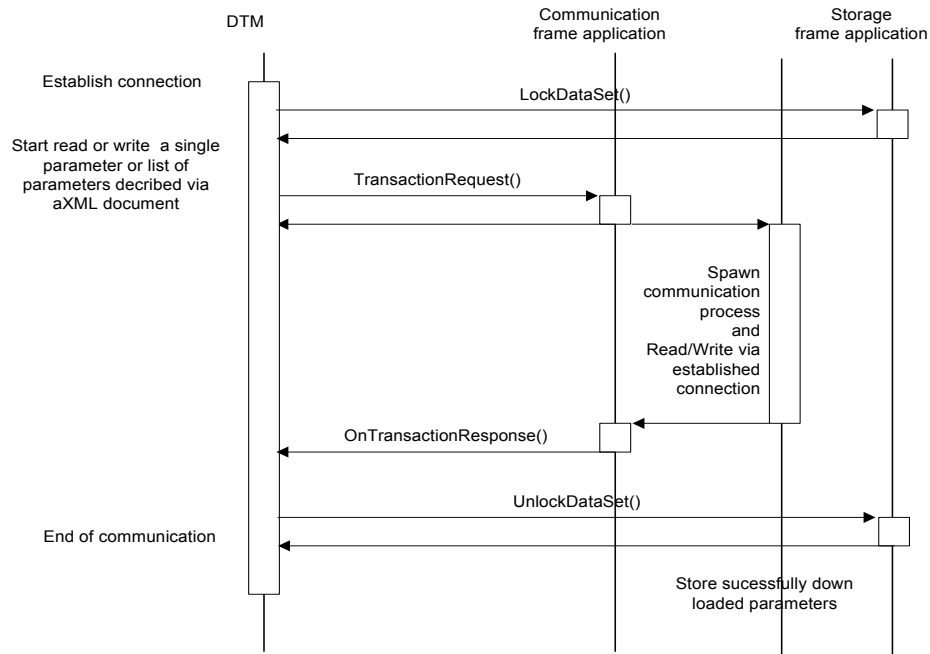
5.1.3 Asynchronous Disconnect

**Used methods:**

[IFdtCommunication::DisconnectRequest \(\)](#)

[IFdtCommunicationEvents::OnDisconnectResponse\(\)](#)

5.1.4 Asynchronous Transaction



Used methods:

`IFdtCommunication::TransactionRequest()`

`IFdtCommunicationEvents::()`

`IFdtContainer::LockDataSet()`

`IFdtContainer::UnlockDataSet()`

5.2 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.

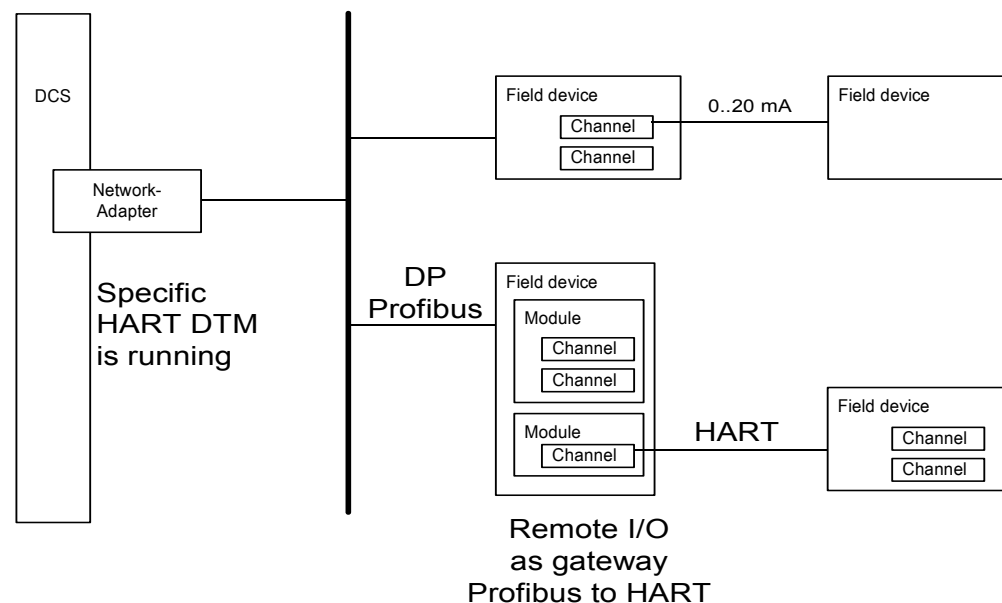
The requirement to this architecture is that a DTM must not know anything about the kind of the overlaying system. Nevertheless, the structure of the sub system is well known to frame-application and DTMs.

Due to this, there is only a reference stored via the IFdtTopology interface mirroring the system topology.

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:

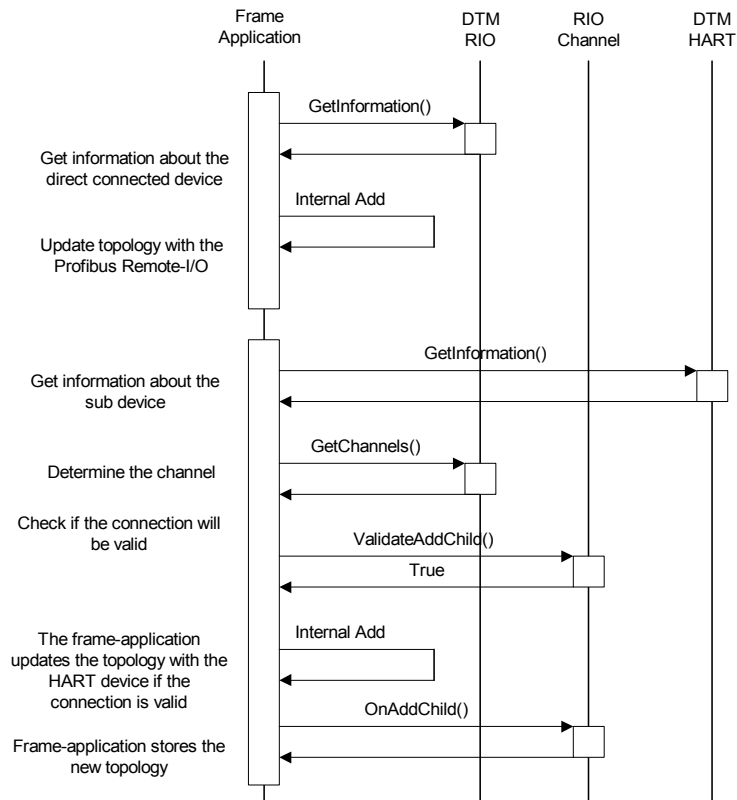


5.2.1 Generate Systemtopology

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

5.2.1.1 Frame-Applications point of view



Used methods:

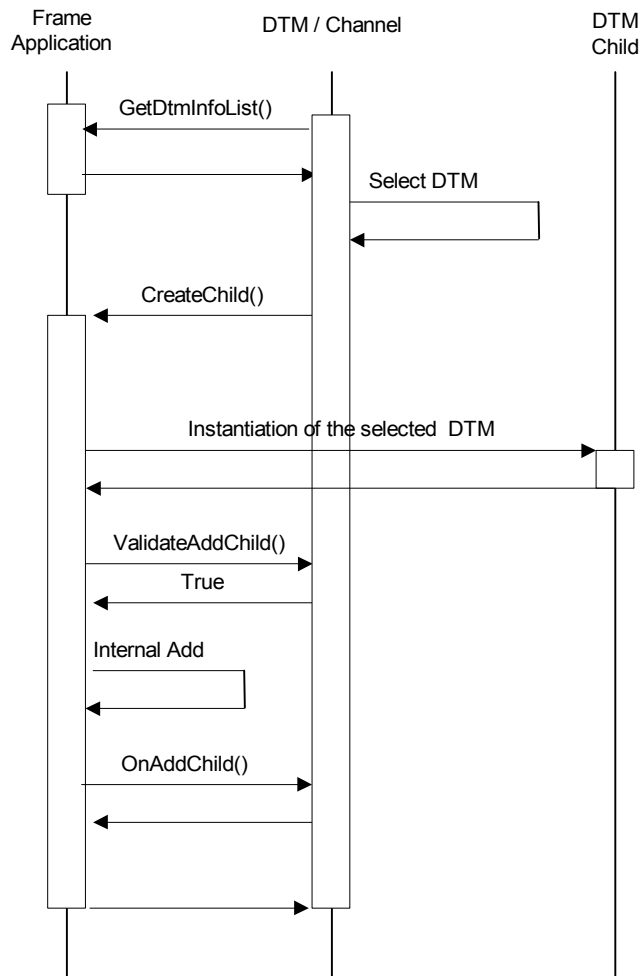
[IDtmInformation::GetInformation\(\)](#)

[IDtmChannel::GetChannels\(\)](#)

[IFdtChannelSubTopology::ValidateAddChild\(\)](#)

[IFdtChannelSubTopology::OnAddChild\(\)](#)

5.2.1.2 DTMs point of view



Used methods:

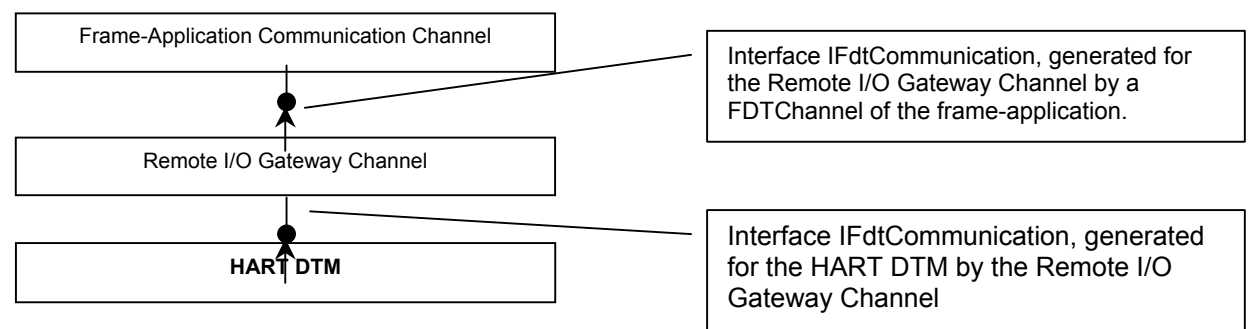
[IFdtTopology::GetDtmInfoList\(\)](#)

[IFdtTopology::CreateChild\(\)](#)

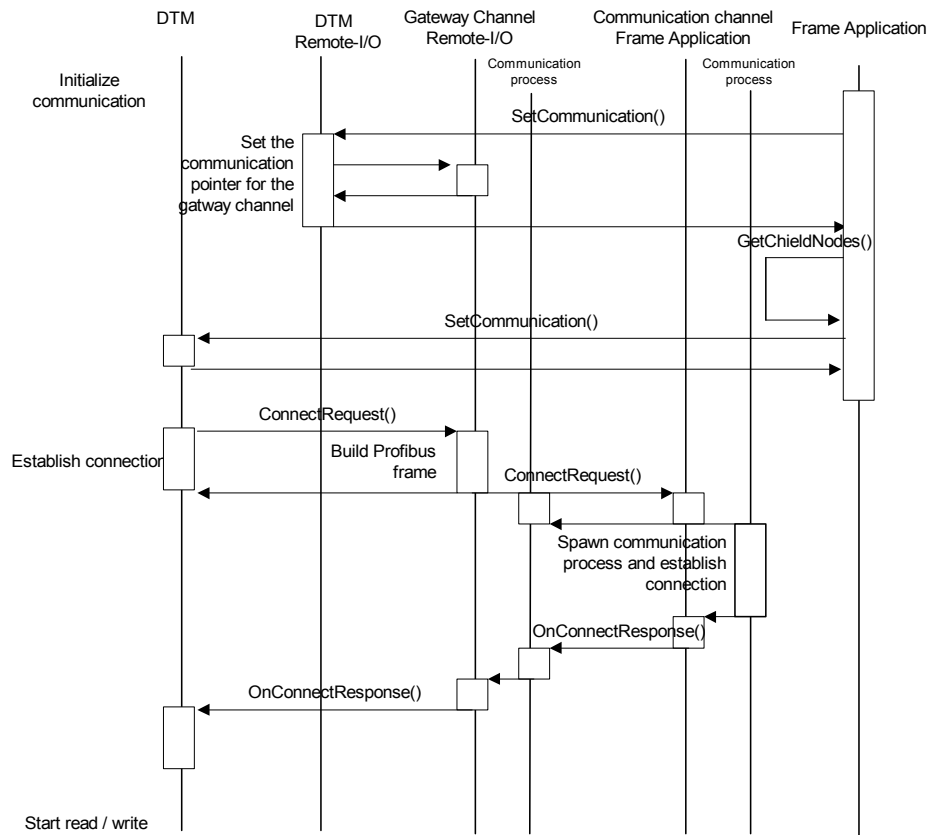
[IFdtChannelSubTopology::ValidateAddChild\(\)](#)

[IFdtChannelSubTopology::OnAddChild\(\)](#)

5.2.2 Establish a Connection between DTM and Device



The topology information must be available to create the proper communication interface hierarchy



Used methods:

[IDtm::SetCommunication\(\)](#)

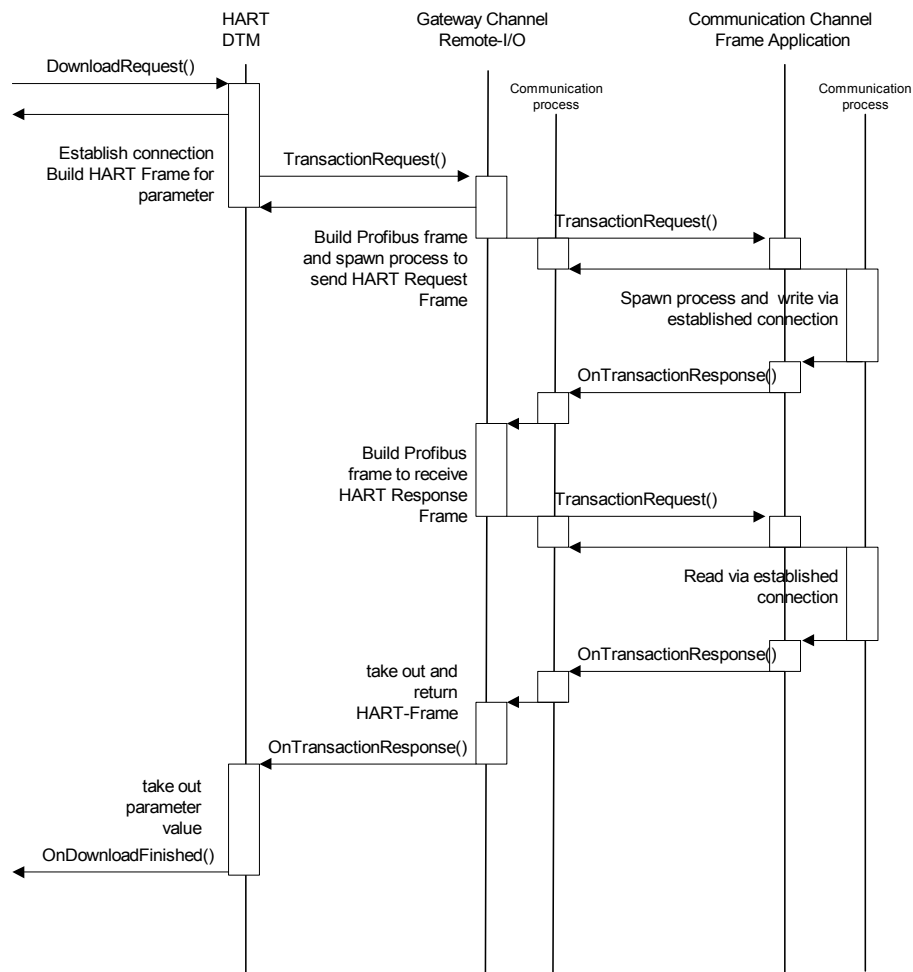
[IFdtTopology::GetChildNodes\(\)](#)

[IFdtCommunication::ConnectRequest\(\)](#)

[IFdtCommunicationEvents::\(\)](#)

5.2.3 Asynchronous Transaction

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 gateway-channel.



Used methods:

`IFdtCommunication::TransactionRequest()`

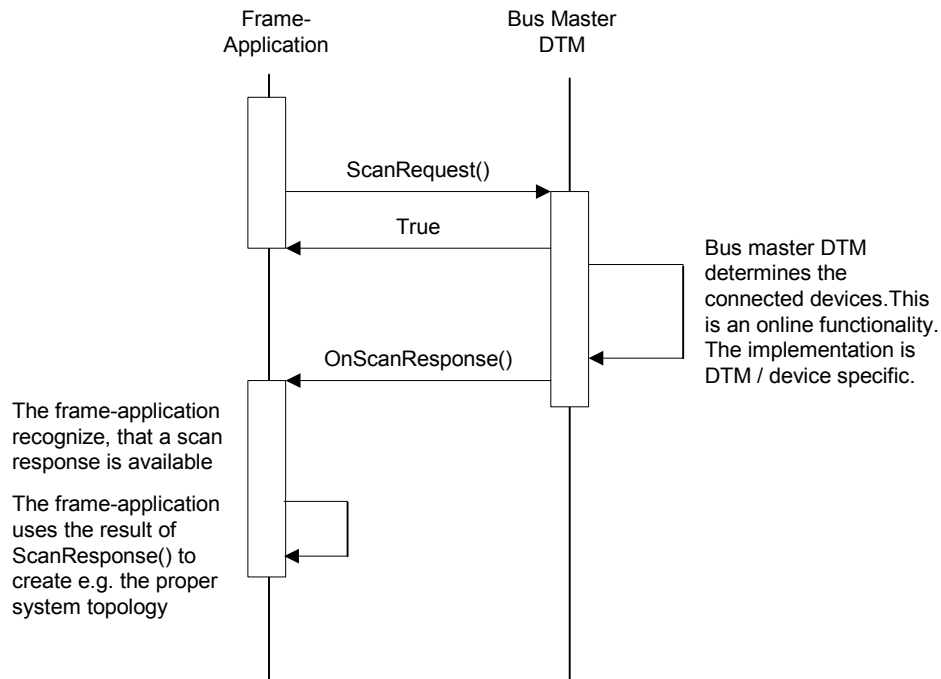
`IFdtCommunicationEvents::OnTransactionResponse()`

`IDtmOnlineParameter::DownloadRequest()`

`IDtmEvents::OnDownloadFinished()`

5.3 Topology Scan

In this scenario a frame-application creates a system topology out of an existing sub topology. This is used for a kind of 'reverse engineering'



Used methods:

`IFdtChannelSubTopology::ScanRequest()`

`IDtmEvents::OnScanResponse()`

5.4 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 DTM's 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.

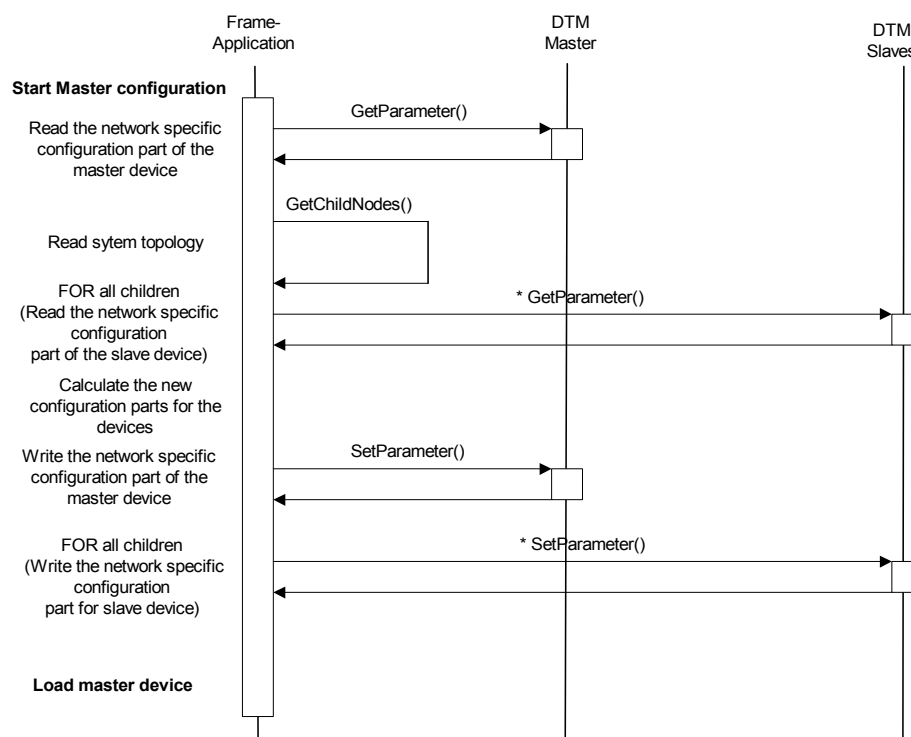
Each DTM must at least fill in the device specific parameters and all parameters which can be changed by its application.

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:

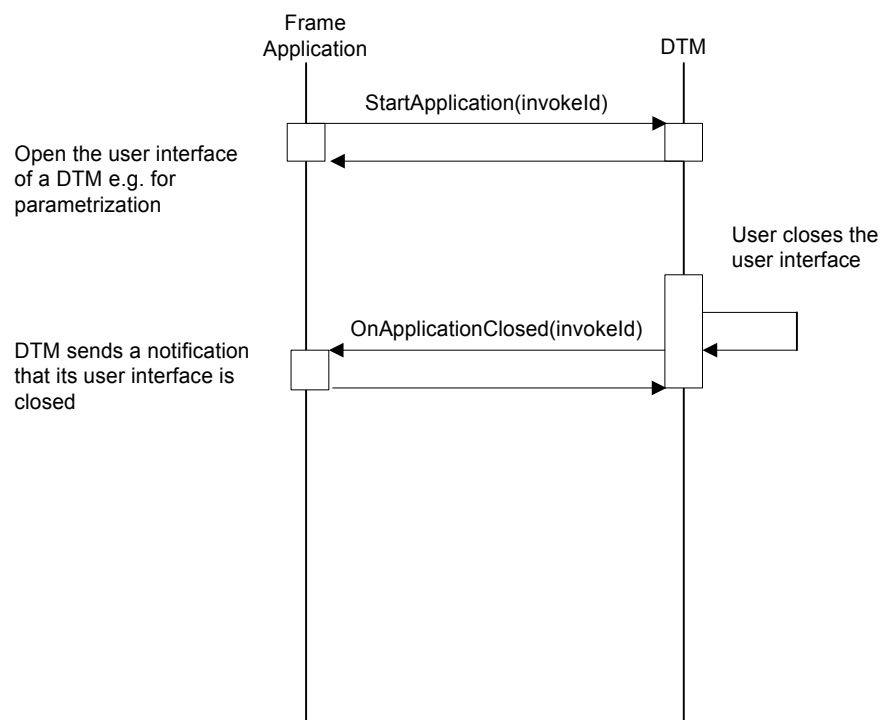
[IdtmParameter::GetParameters\(\)](#)

[IdtmParameter::SetParameters\(\)](#)

[IfdtTopology::GetChildNodes\(\)](#)

5.5 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\(\)](#)



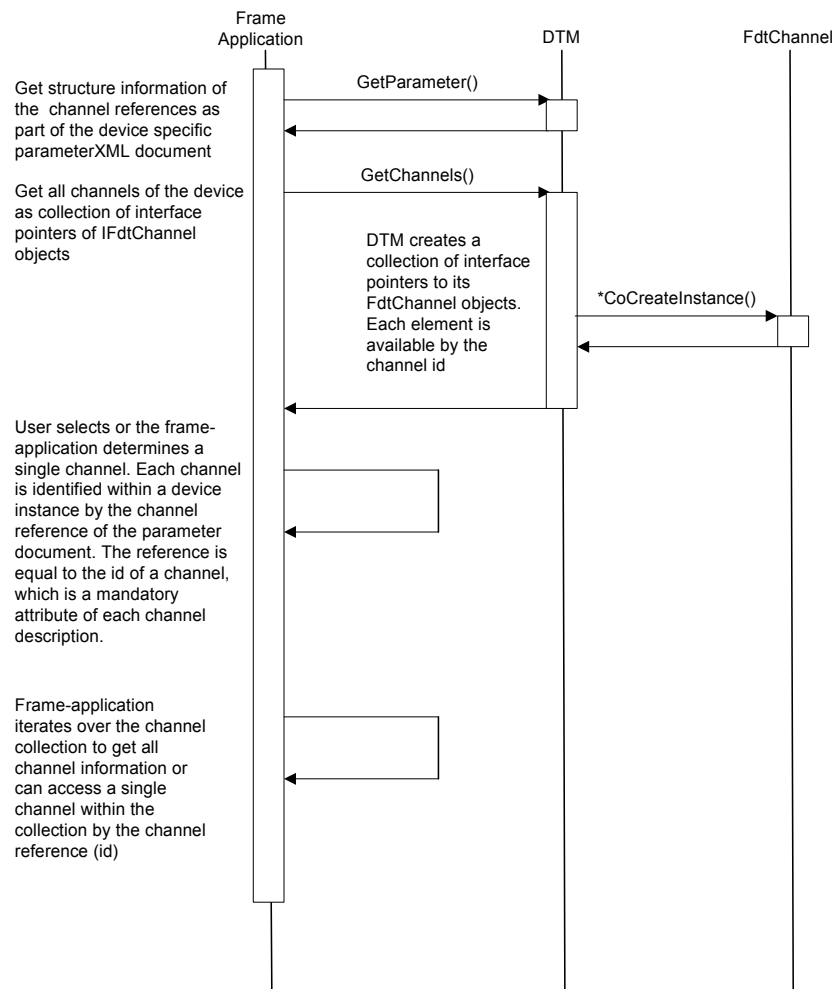
Used methods:

[IdtmApplication::StartApplication\(\)](#)

[IDtmEvents::OnApplicationClosed\(\)](#)

5.6 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\(\)](#)

5.7 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.

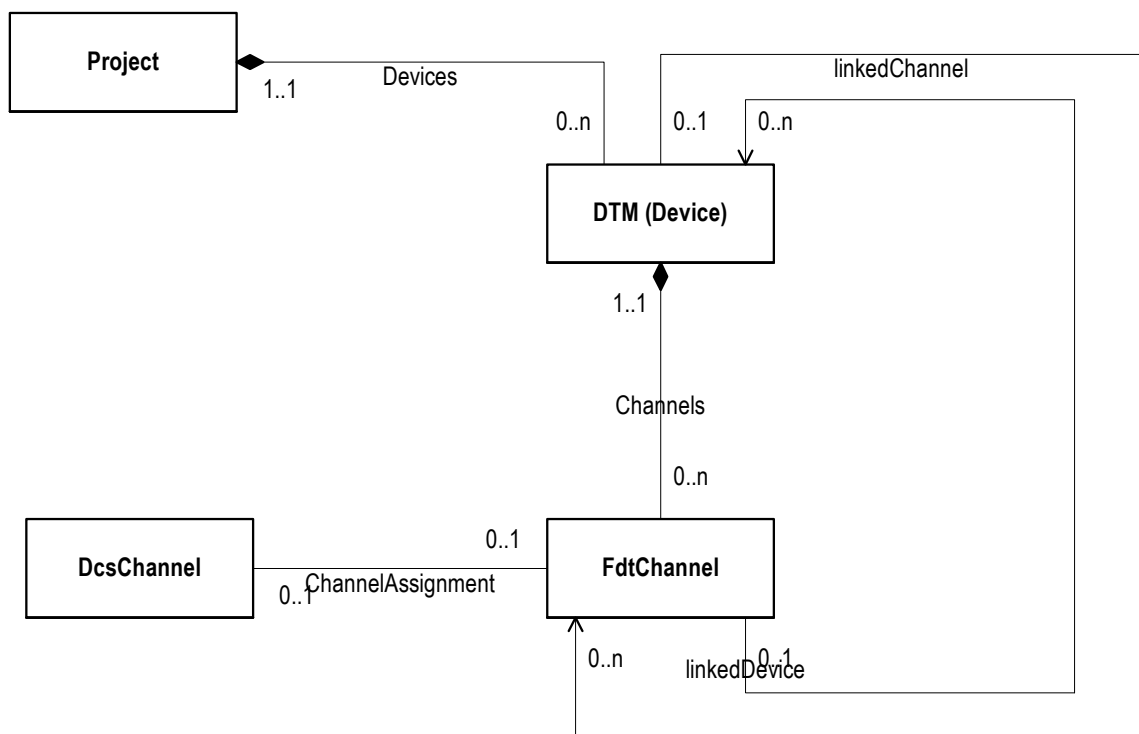
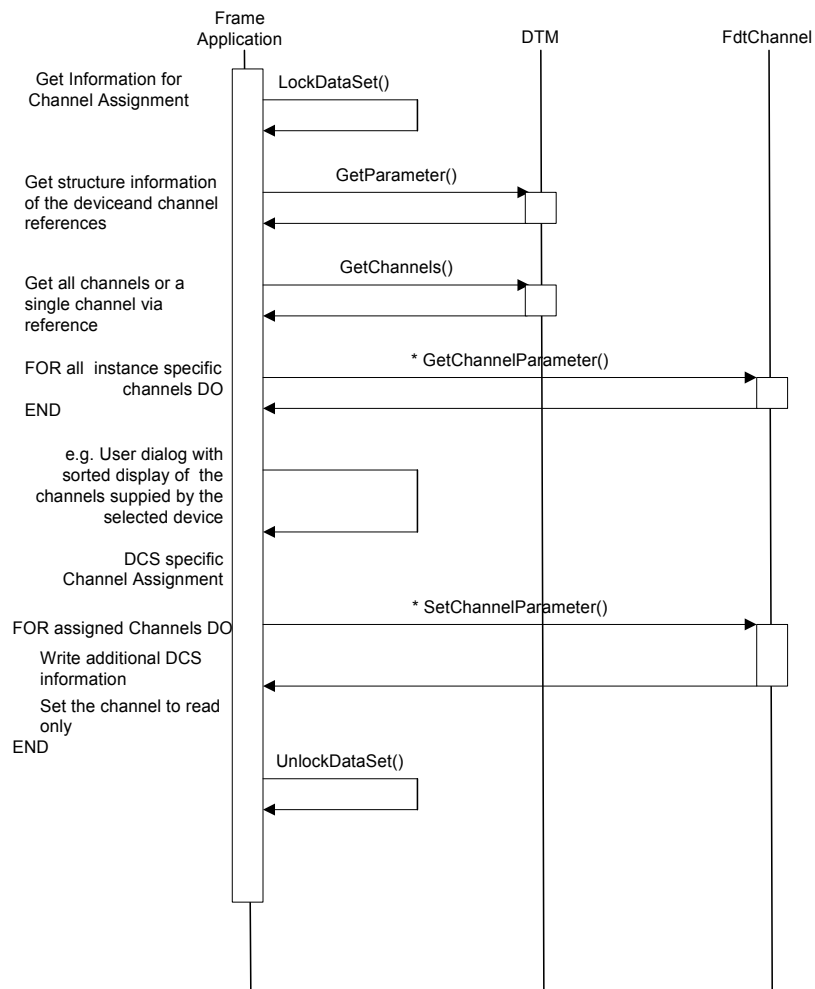


Figure 4-3 – Single DTM

**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.

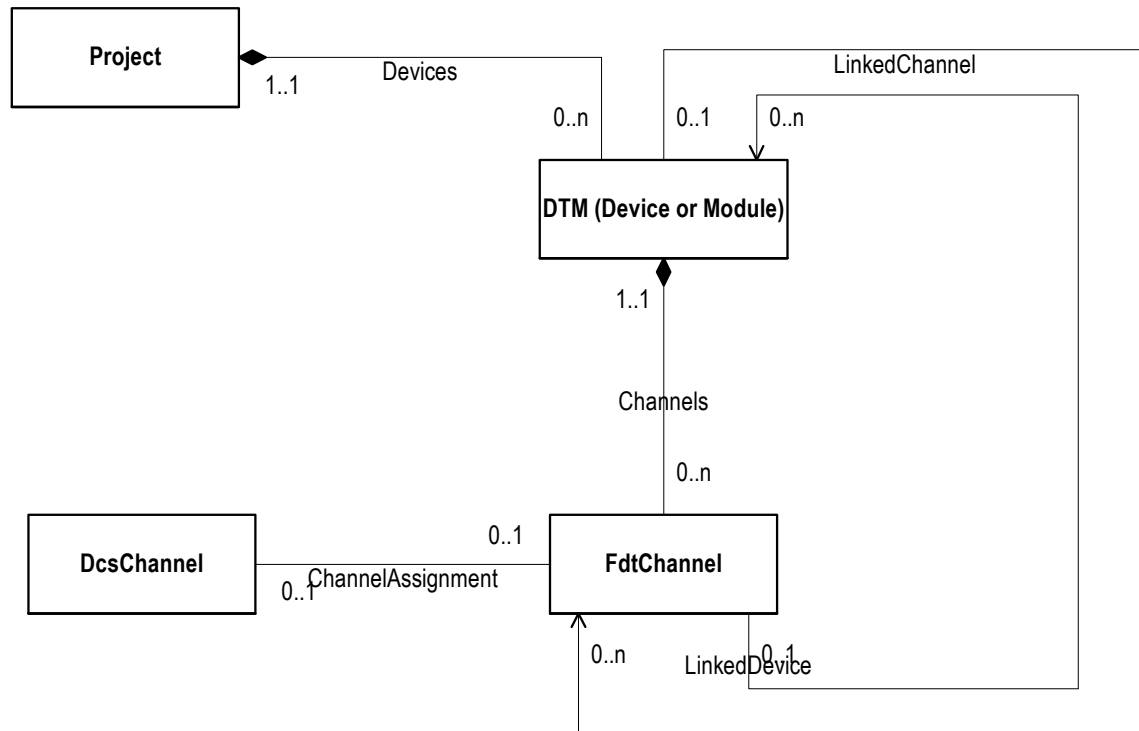
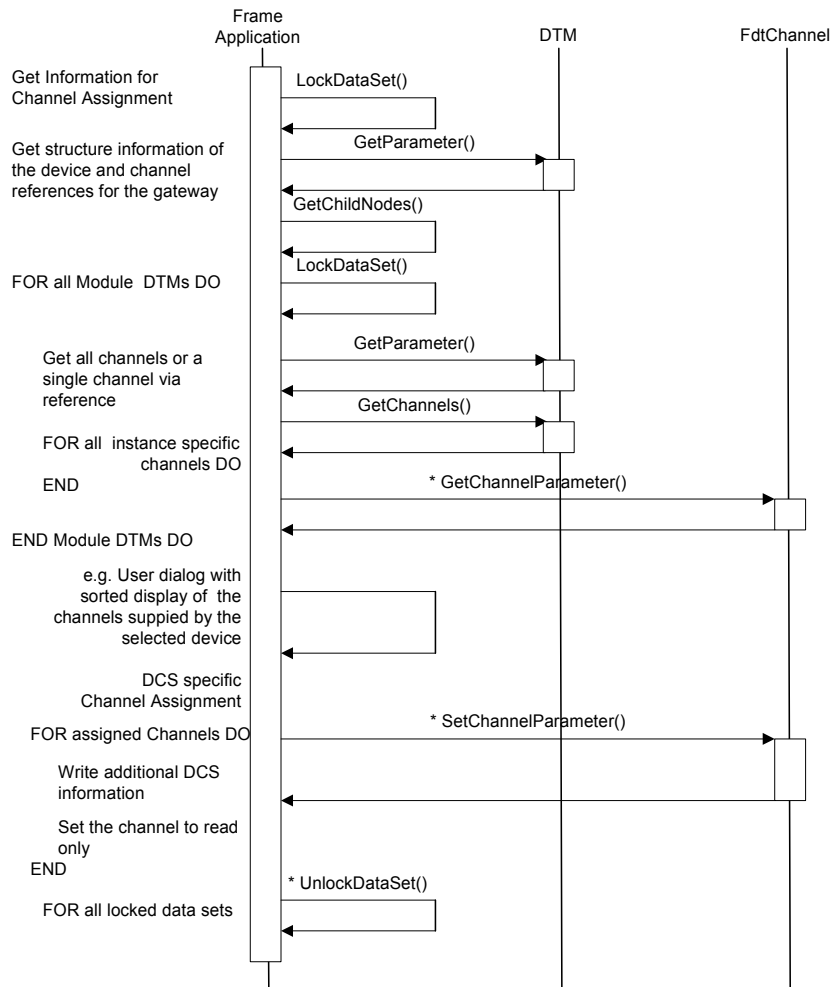


Figure 4-4 – Modular DTM Structure

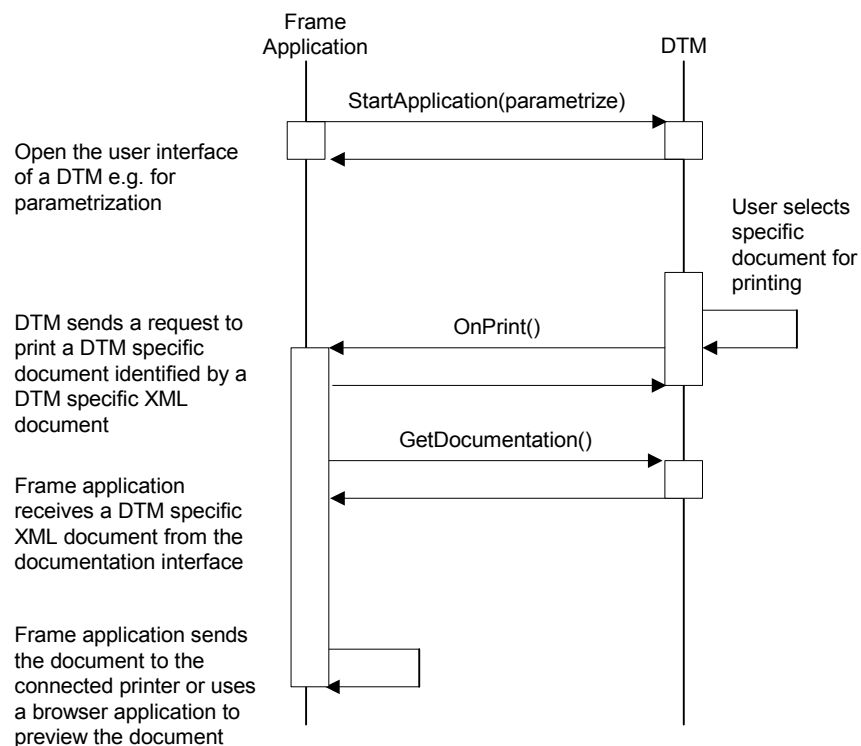
In this case 'LinkedDevice' specifies the connection between a gateway-channel of the bus coupler (DTM Device) and the modules (DTM Module) as well as the connection between a gateway-channel of a module and a connected device. Especially for a remote I/O the FdtChannels are similar to the slots at the 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()`

5.8 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:

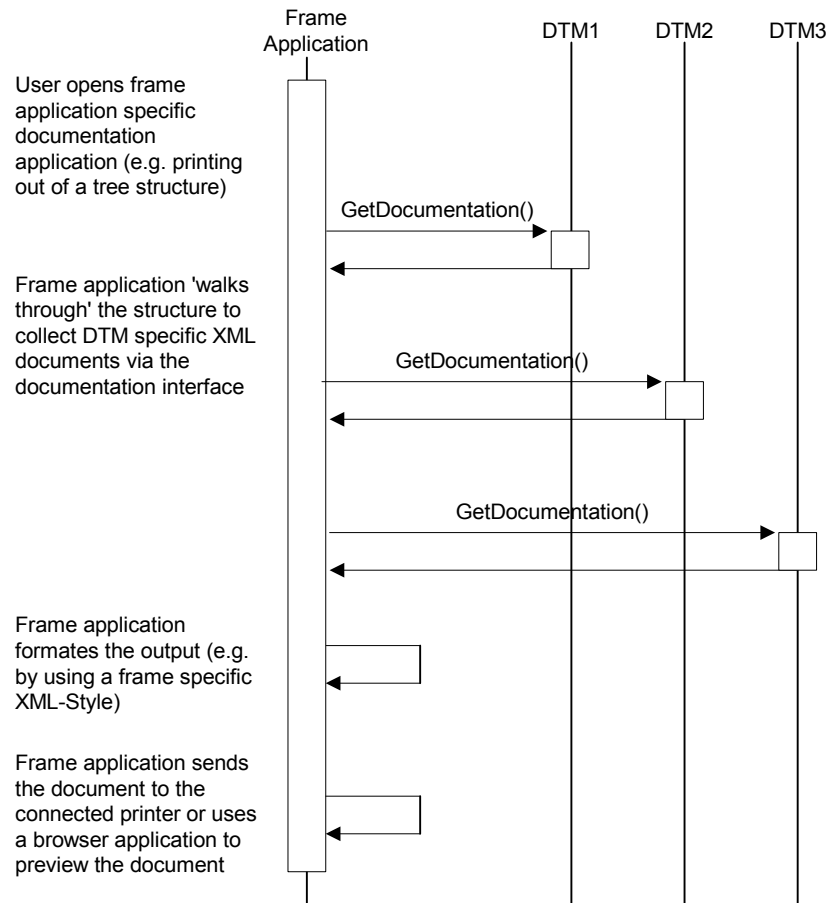
[IDtmApplication::StartApplication\(\)](#)

[IDtmEvents::OnPrint\(\)](#)

[IDtmDocumentation::GetDocumentation\(\)](#)

5.9 Printing of Frame specific Documents

[IdtmDocumentation](#) allows the frame-application to ask a DTM for context specific documents identified by the information passed via an XML document.

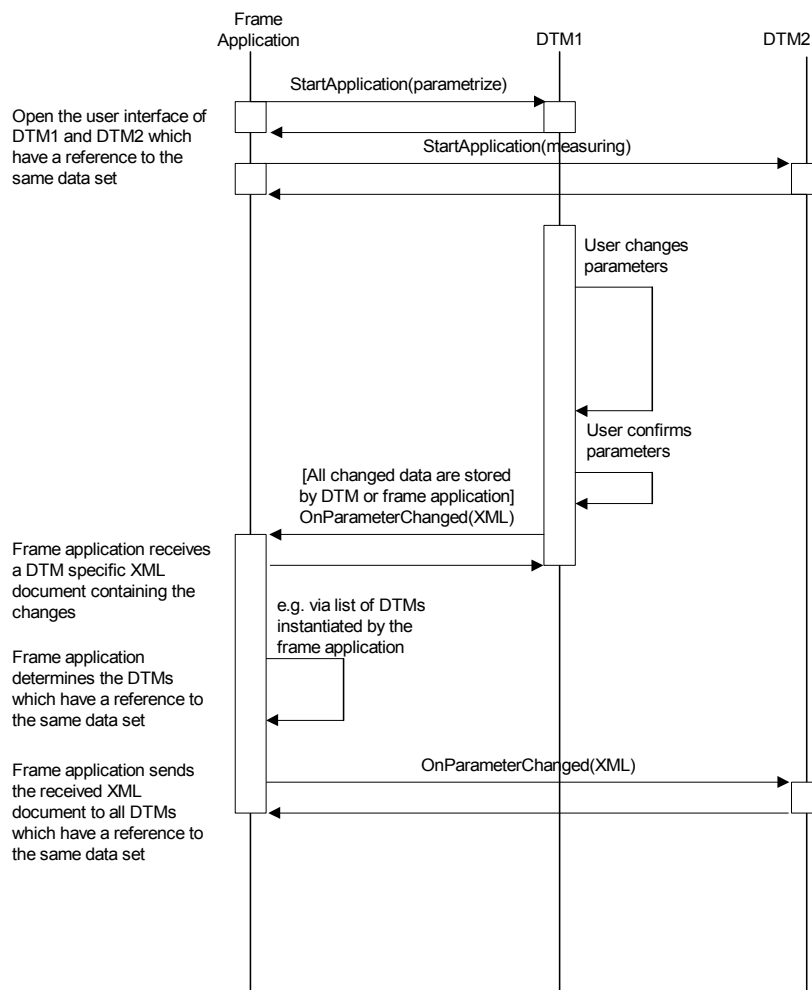


Used methods:

[IdtmDocumentation](#):: `GetDocumentation()`

5.10 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\(\)](#)

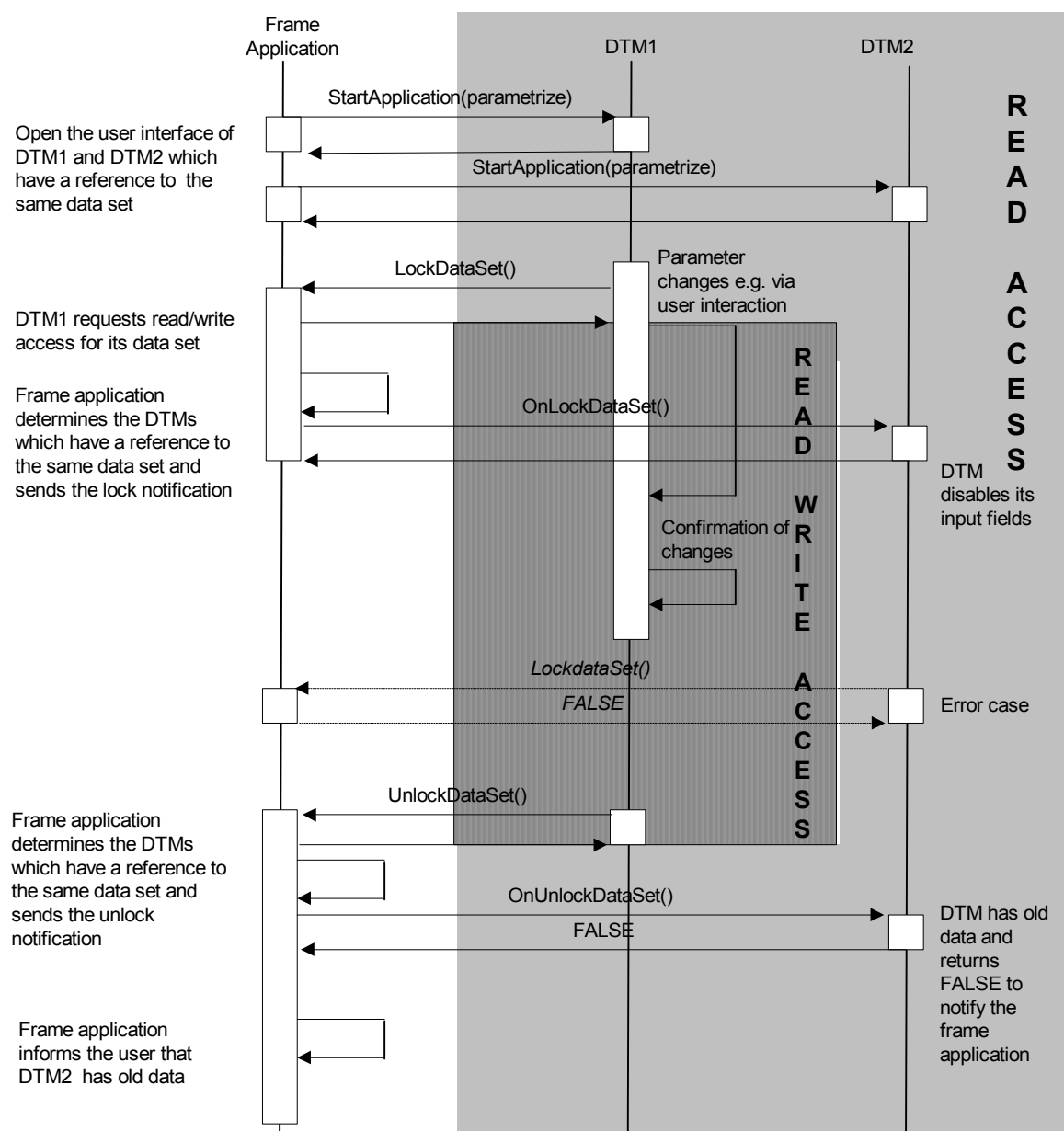
[IfdtEvents::OnParameterChanged\(\)](#)

5.11 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.

5.11.1 Locking for non synchronized DTMs

This locking mechanism is suitable for all DTMs which do not implement `lfdtEvents::OnParameterChanged()` (`lfdtEvents::OnParameterChanged()` returns `E_NOT_IMPL`). The mechanism must be implemented to support multi-user environments.



Used methods:

[IdtmApplication::StartApplication\(\)](#)

[IfdtContainer::LockDataSet\(\)](#)

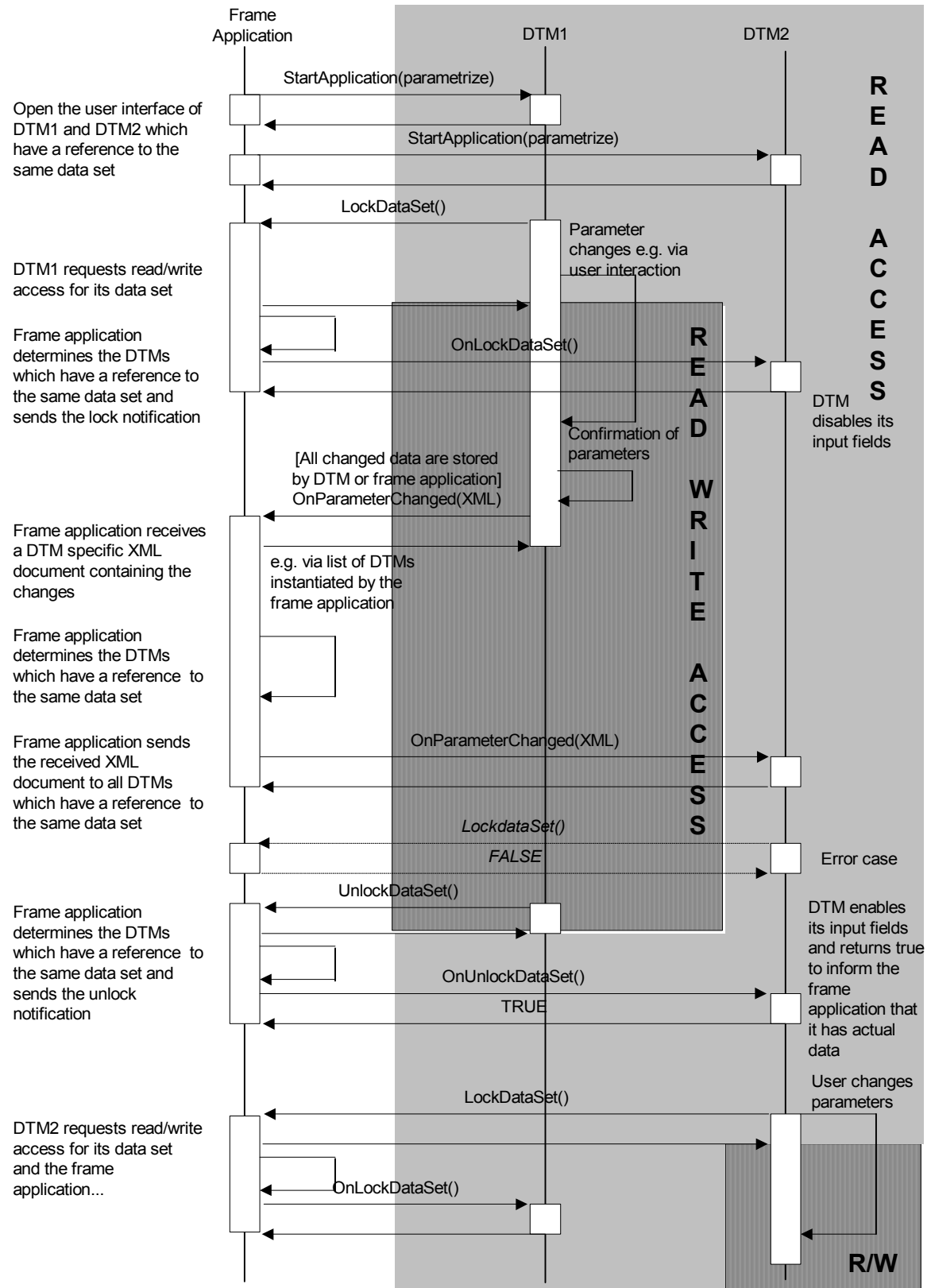
[IfdtContainer::UnlockDataSet\(\)](#)

[IfdtEvents::OnLockDataSet\(\)](#)

[IfdtEvents::OnUnlockDataSet\(\)](#)

5.11.2 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()`

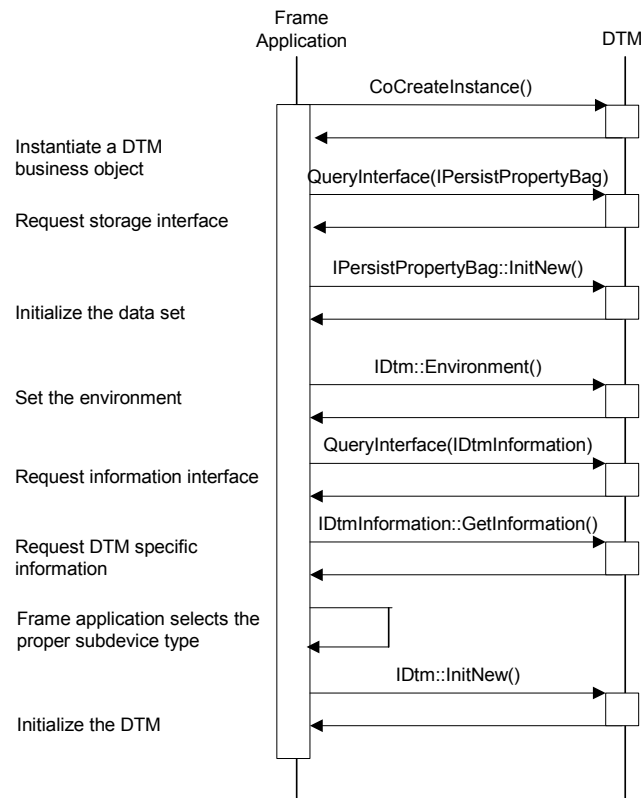
`IfdtEvents::OnLockDataSet()`

`IfdtEvents::OnUnlockDataSet()`

5.12 Instantiation and Release

5.12.1 Instantiation of a new DTM

After creation of a DTM business object) the frame 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

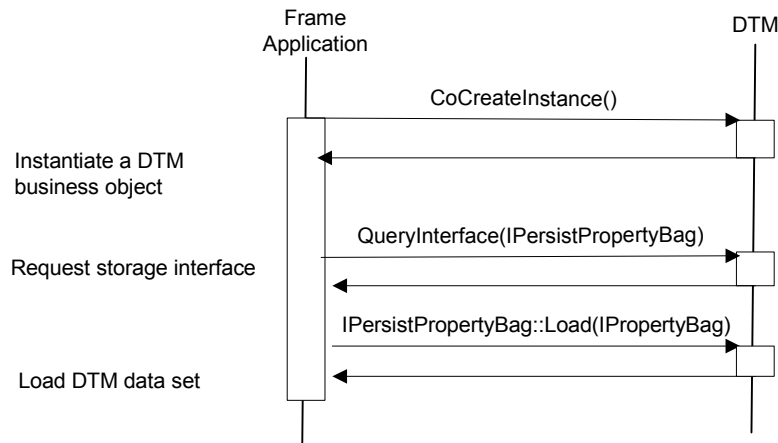
[IDtm::Environment\(\)](#)

[IDtm::InitNew\(\)](#)

[IDtmInformation::GetInformation\(\)](#)

5.12.2 Instantiation of a existing DTM

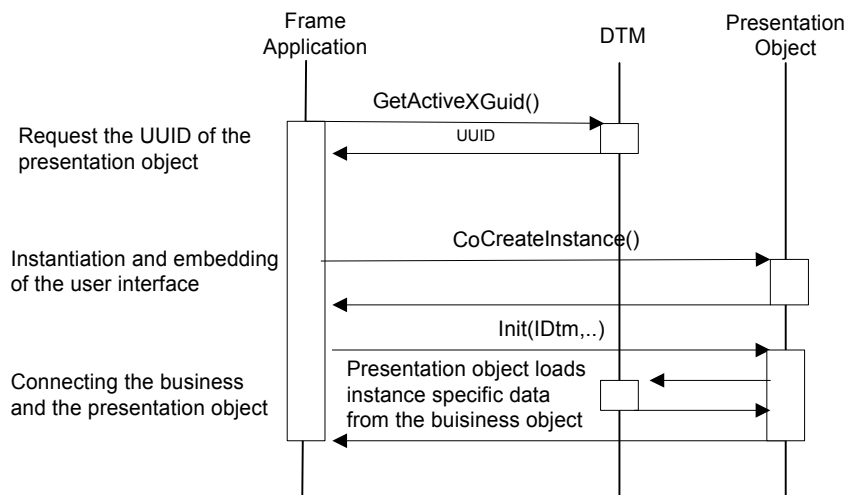
After creation of a DTM business object for an existing instance the frame must request the IPersistXXX interface pointer and call IPersistXXX::Load() with a reference to the stream object of the appropriate field device instance. Within this method the DTM business object must initialize it's device parameters based on the information of the stream object.

**Used methods:**

Standard Microsoft

5.12.3 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.

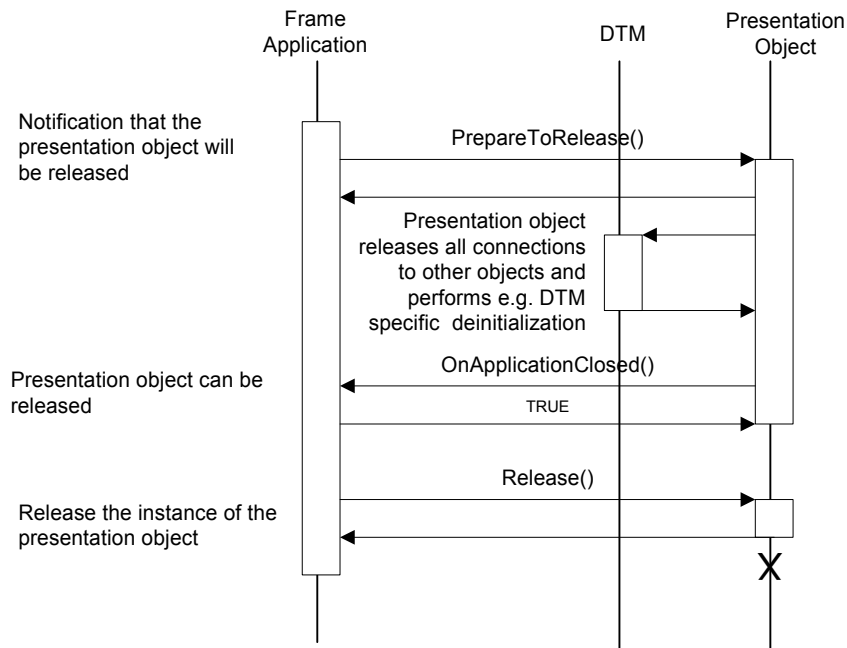
**Used methods:**

Standard Microsoft

[IDtmActiveXControl::Init\(\)](#)[IDtmActiveXInformation::GetActiveXGuid\(\)](#)

5.12.4 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. Within the `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\(\)](#)

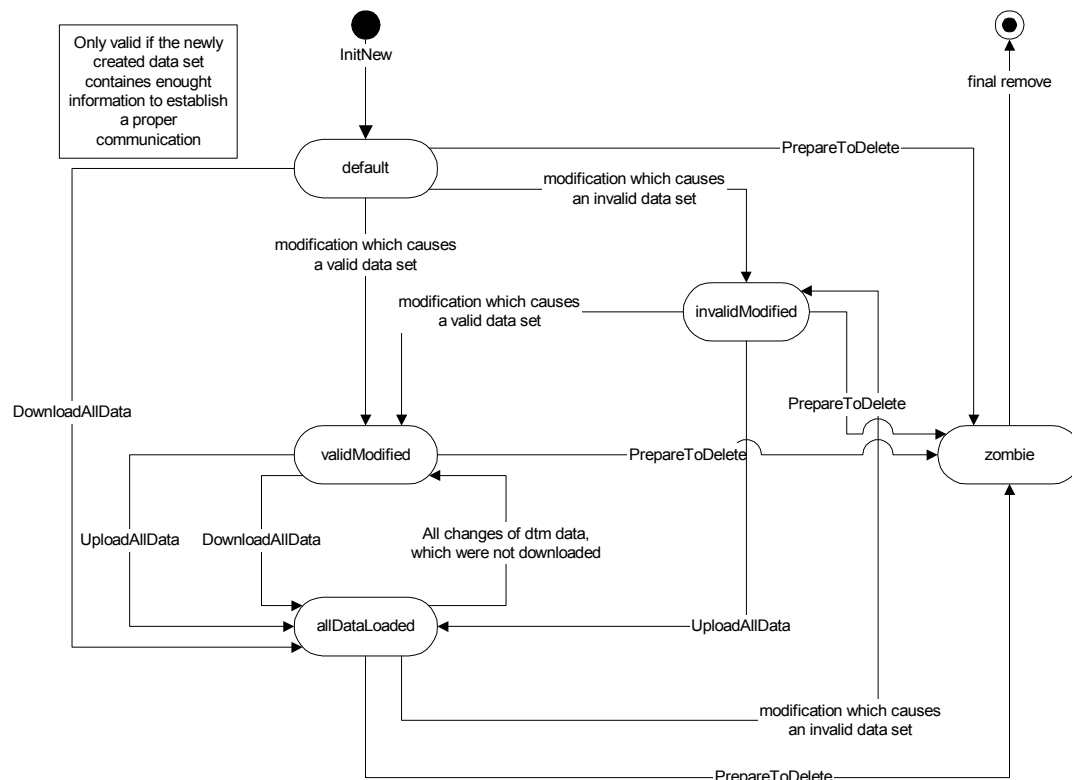
[IDtmEvents::OnApplicationClosed\(\)](#)

5.13 Persistent Storage of a DTM

5.13.1 State machine of instance data

5.13.1.1 Modifications

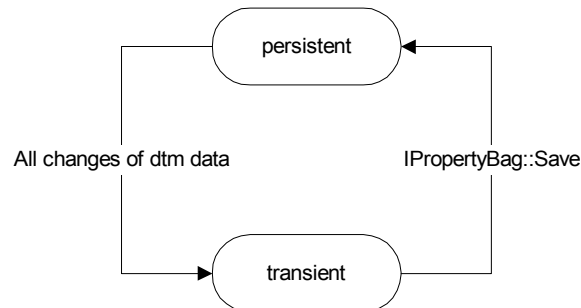
This state machine reflects the possible states of an instance data set concerning modifications.



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

5.13.1.2 Persistence

This state machine reflects the possible states of an instance data set concerning the persistence of data.

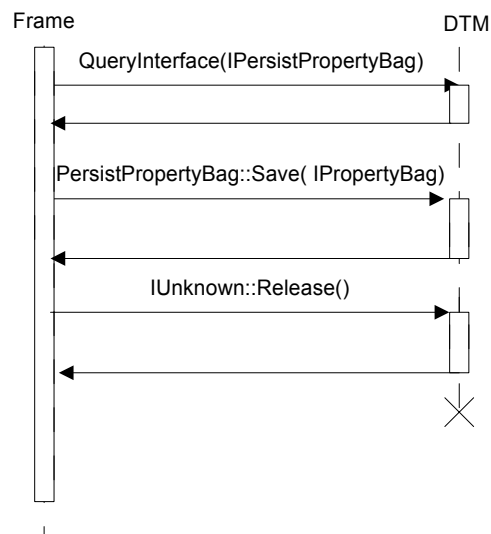


State	Meaning
persistent	The content of the instance data set is persistent. This state will typically appear after the data set was saved
transient	The data set was modified. These changes are not saved in a persistent way

5.13.2 Saving instance data of a DTM

To save the private data of a DTM object the frame 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 is able to release the DTM business object



Used methods:

Standard Microsoft

5.13.3 Reload of a DTM object for another instance

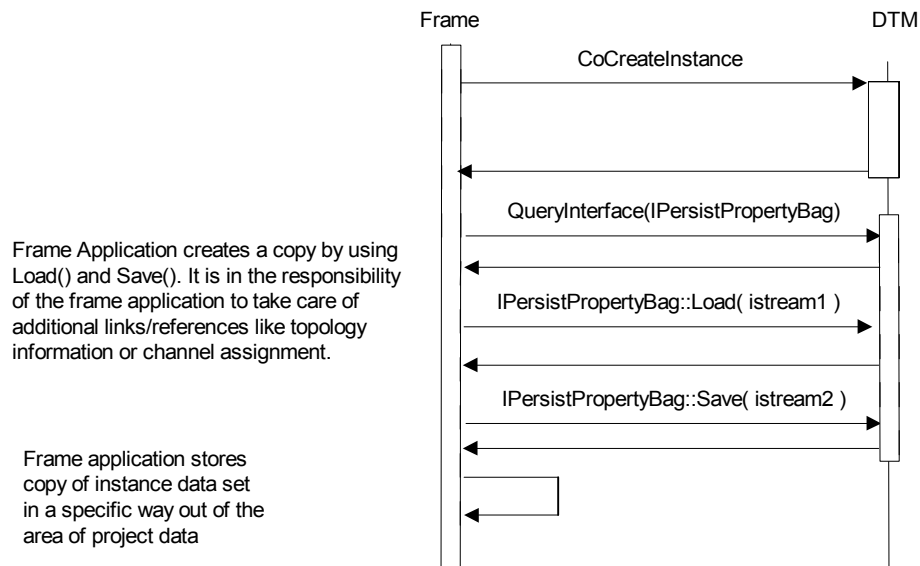
A frame-application may reuse one DTM COM object for a number of different field device instances by calling `IPersistXXX::Load()` several times and with different `Istream` / `IpropertyBag` objects as argument. But, if an `IPersistXXX::Load()` call fails, the frame-application must instantiate a new DTM COM object.

5.14 Versioning

5.14.1 Copy of a DTM

After creation of a DTM object the frame must request the IPersist 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 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 specific versioning aspects and to manage the different instance data sets for a device.



Used methods:

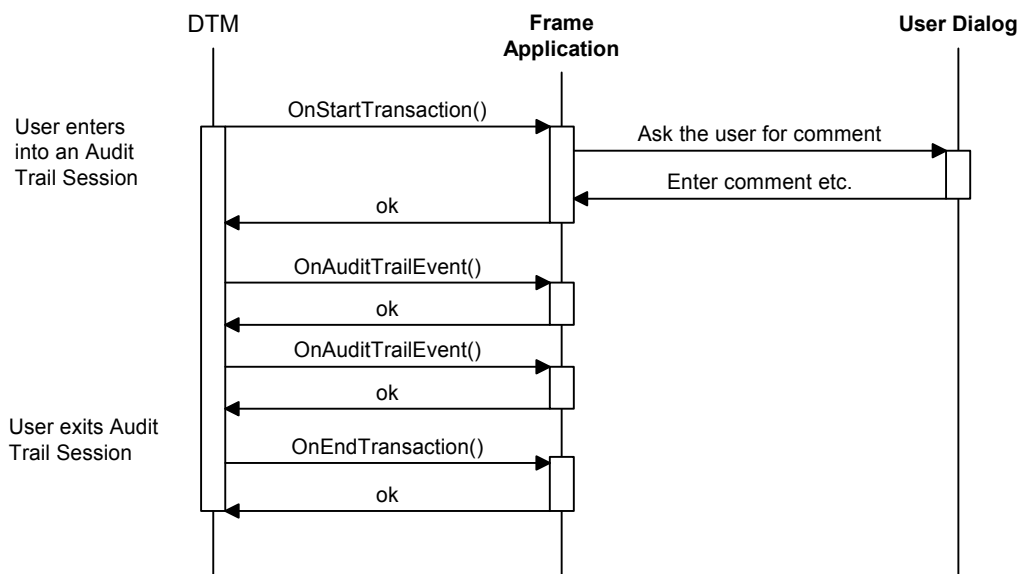
Standard Microsoft

5.15 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`. The audit trail functionality and the interface for the DTM are optional.

5.15.1 Frame-application supports Audit Trail

The frame-application supports audit trail functionality and is ready for starting a audit trail session:



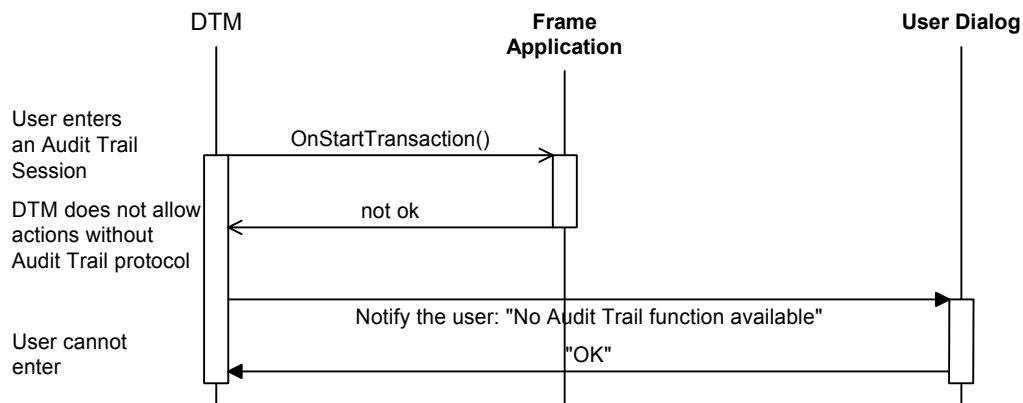
Used methods:

`IdtmAuditTrailEvents::OnStartTransaction()`
`IdtmAuditTrailEvents::OnAuditTrailEvent()`
`IdtmAuditTrailEvents::OnEndTransaction()`

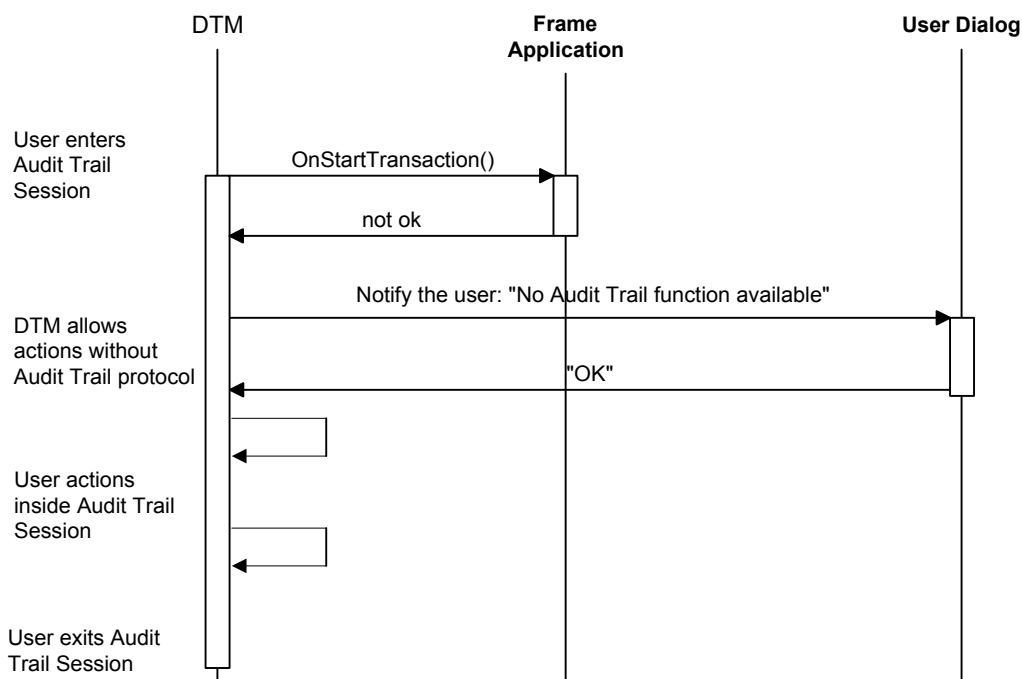
5.15.2 Frame-application doesn't support Audit Trail

The frame-application does not support audit trail functionality or is not ready for starting an audit-trail session. It's up to the DTM to allow or not to allow actions without audit trail protocol.

This DTM does not allow actions without Audit Trail protocol:



This DTM allows actions without Audit Trail protocol:



Used methods:

[IdtmAuditTrailEvents::OnStartTransaction\(\)](#)
[IdtmAuditTrailEvents::OnAuditTrailEvent\(\)](#)
[IdtmAuditTrailEvents::OnEndTransaction\(\)](#)

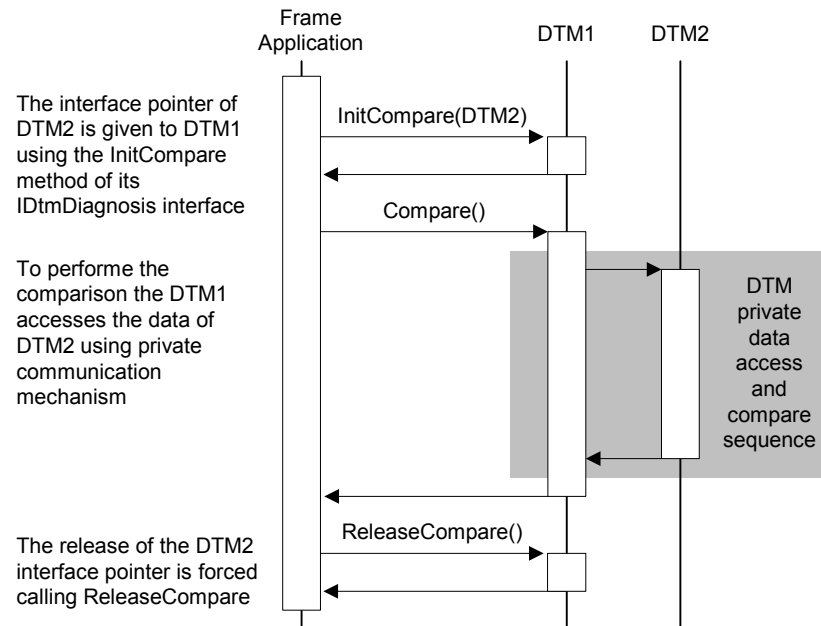
5.16 Comparison of two Instance Data Sets

This section describes sequences to compare the data sets of two different instances using the IdtmDiagnosis interface.

The two DTMs invoked in the comparison have to be of the same type.

5.16.1 Comparison without User Interface

The sequence chart for the comparison of two instance data sets is as follows:



Used methods:

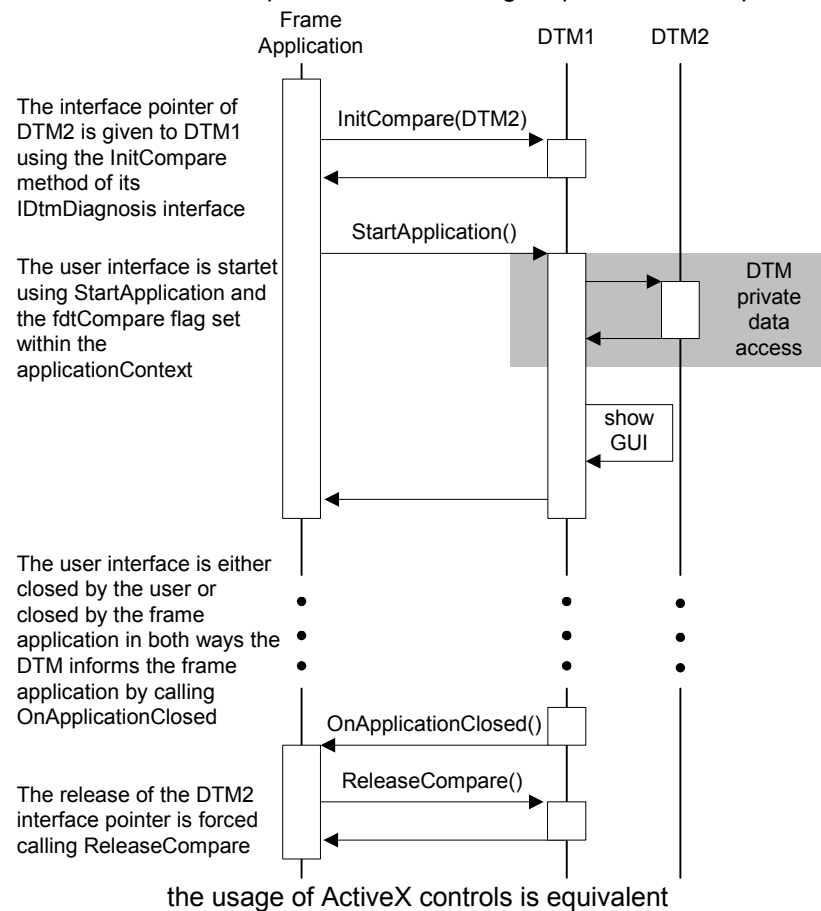
[IdtmDiagnosis::InitCompare\(\)](#)

[IdtmDiagnosis::Compare\(\)](#)

[IdtmDiagnosis::ReleaseCompare\(\)](#)

5.16.2 Comparison with User Interface

To invoke a user interface for comparison, the following sequence is to be performed.



Used methods:

[IdtmDiagnosis::InitCompare\(\)](#)

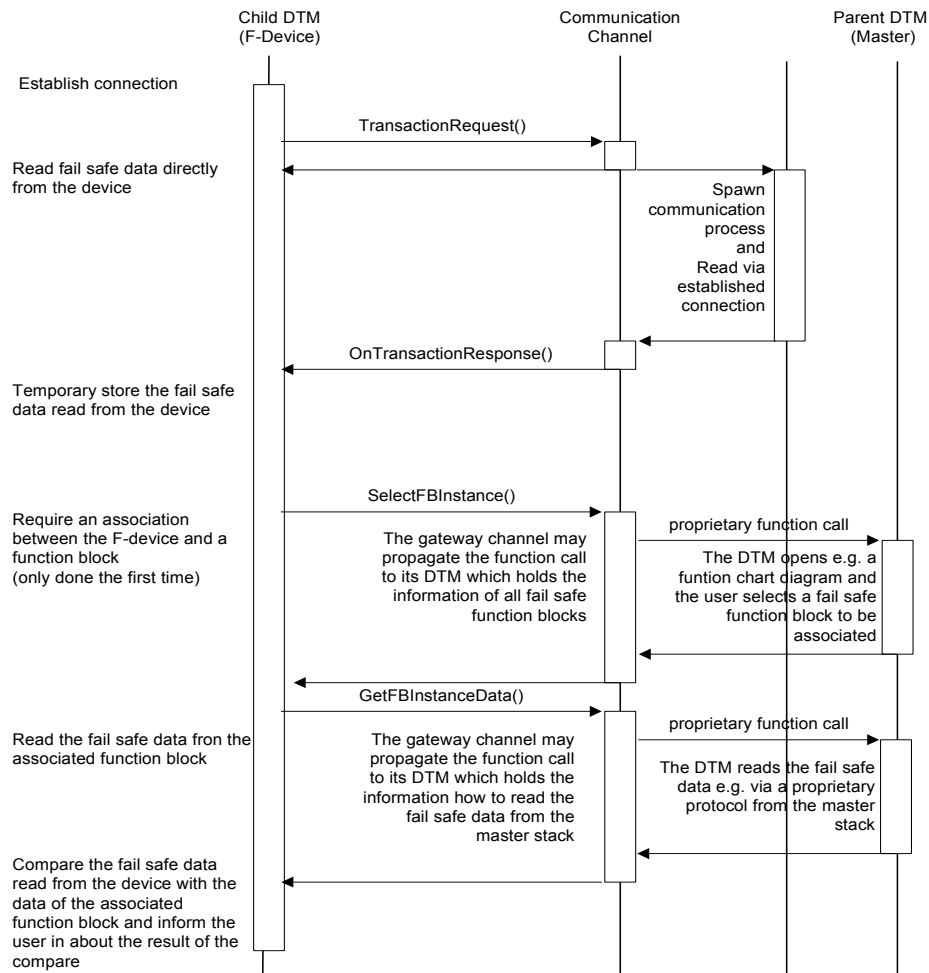
[IdtmDiagnosis::ReleaseCompare\(\)](#)

[IdtmApplication::StartApplication\(\)](#)

[IDtmEvents::OnApplicationClosed\(\)](#)

5.17 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 Installation Issues

6.1 FDT Registry and Device Information

6.1.1 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.

6.1.2 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 providing class information via IDtmInformation .
"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 DTM Channel"	CATID_FDT_CHANNEL	{036D1493-387B-11D4-86E1-00E0987270B9}	Channel object of a DTM for integration within a frame-application. Available via IDtmChannel
"FDT DTM Communication"	CATID_FDT_COMMUNICATION	{036D1494-387B-11D4-86E1-00E0987270B9}	Object providing communication interfaces
"FDT DTM Presentation ActiveXControl"	CATID_FDT_PRESENTATIONACTIVECONTROL	{036D1495-387B-11D4-86E1-00E0987270B9}	Presentation object of a DTM as ActiveX-Control available via IDtmActiveXInformation
"FDT DTM Included Presentation"	CATID_FDT_PRESENTATION	{036D1496-387B-11D4-86E1-00E0987270B9}	Presentation object are part of a DTM and are not available for embedding within the frame-application
"FDT FIELDBUS Profibus DPV0"	CATID_FDT_PROFIBUS_DPV0	{036D1497-387B-11D4-86E1-00E0987270B9}	Object supports Profibus DP V0 protocol
"FDT FIELDBUS Hart"	CATID_FDT_HART	{036D1498-387B-11D4-86E1-00E0987270B9}	Object supports HART protocol
"FDT FIELDBUS Profibus DPV1"	CATID_FDT_PROFIBUS_DPV1	{036D1499-387B-11D4-86E1-00E0987270B9}	Object supports Profibus DPV1 protocol
		{036D149A-387B-11D4-86E1-00E0987270B9}	Reserved for further releases
		...	Reserved for further releases
		{036D1590-387B-11D4-86E1-00E0987270B9}	Reserved for further releases

A CATID consist of its symbolic name and the UUID within the registry. The FDT IDL defines a symbolic name, e.g., CATID_FDT_PROFIBUS.

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_CHANNEL	CATID_FDT_COMMUNICATION	CATID_FDT_PRESENTATIONACTIVEXCONTROL	CATID_FDT_PRESENTATION	CATID_FDT_HART	CATID_FDT_PROFIBUS_DPV0	CATID_FDT_PROFIBUS_DPV1
CATID_FDT_DTM		√	√			√	√	√	√	√
CATID_FDT_DEVICE	√					√	√	√	√	√
CATID_FDT_MODULE	√					√	√	√	√	√
CATID_FDT_CHANNEL					√	√		√	√	√
CATID_FDT_COMMUNICATION				√		√		√	√	√
CATID_FDT_PRESENTATIONACTIVEXCONTROL	√	√	√	√	√					
CATID_FDT_PRESENTATION	√	√	√							
CATID_FDT_HART	√	√	√	√	√					
CATID_FDT_PROFIBUS_DPV0	√	√	√	√	√					
CATID_FDT_PROFIBUS_DPV1	√	√	√	√	√					

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 Profibus device	CATID_FDT_DTM CATID_FDT_DEVICE CATID_FDT_PROFIBUS
DTM for a module of a modular device	CATID_FDT_DTM CATID_FDT_MODULE
Channel object of a DTM	CATID_FDT_CHANNEL
Channel object of a DTM with gateway functionality for HART	CATID_FDT_CHANNEL CATID_FDT_HART CATID_FDT_COMMUNICATION
DTM with ActiveX Control as presentation object	CATID_FDT_DTM CATID_FDT_PRESENTATIONACTIVEXCONTROL
DTM with integrated user interfaces	CATID_FDT_DTM CATID_FDT_DEVICE CATID_FDT_PRESENTATION

Additionally, at least one of the “FDT FIELDBUS ...” categories must be registered for each class object.

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 IcatRegister documentation for additional information.

6.1.3 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.

6.1.4 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.

6.1.5 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.

6.1.6 Building a Frame-Application-Database of available DTMs and Supported Devices

Using component categories a frame-application is able to detect all installed DTMs together with the different types of business and presentation objects available within a DTM. Additional information, e.g. vendor information, list of supported devices etc., can be determined by instantiating a DTM and using the [ldtmInformation](#) interface.

By using this information, a frame-application is able to build a database with

- All available DTMs
- DTMs supporting a specific fieldbus
- Available presentation controls
- 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 out of scope of the FDT specification.

7 Description of Data Types, Parameters and Structures

7.1 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.

	Data type	
invokeld	FdtUUIDString	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.
CommunicationReference	FdtUUIDString	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.

7.2 Data Type Definitions

Helper objects for documentation

Name	Data type	
FdtUUIDString	BSTR	String containing a unique identifier according to the Microsoft standard UUID. The format must be e.g. "C2137DD1-7842-11d4-A3C9-005004DC410F" (without bracket).
FdtXMLDocument	BSTR	String containing a XML document
FdtXPath	BSTR	String containing a Xpath to an element of a XML document For FDT 1.2 the Xpath has to be the root tag "FDT" <ul style="list-style-type: none"> • Data exchange via complete documents until the specification of data fragment interchange is released by the W3C • XML documents are accepted as complete document. Otherwise the DTM informs the frame-application via the event interface about the occurred of errors
All boolean parameters	VARINAT_BOOL	TRUE and FALSE according to the definition of VARIANT_TRUE and VARIANT_FALSE

8 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
Gateway Channel	A gateway-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 the PROFIBUS device with a predetermined syntax. It is delivered with 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 gateway-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

9 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 < http://www.w3.org >		
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

10 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 1.20 Interfaces

[
    uuid(036D1471-387B-11D4-86E1-00E0987270B9),
    version(1.0)
]
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 invokeId,
            [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 invokeId,
            [in] FdtXmlDocument functionCall,
            [in] BSTR windowTitle,
            [out, retval] VARIANT_BOOL* result);

        [id(0x2)]
        HRESULT ExitApplication(
            [in] FdtUUIDString invokeId,
            [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 invokeId,
        [out,retval] VARIANT_BOOL* result);

    [id(0x2)]
    HRESULT DownloadRequest(
        [in] FdtUUIDString invokeId,
        [in] FdtXPath parameterPath,
        [out,retval] VARIANT_BOOL* result);

    [id(0x3)]
    HRESULT UploadRequest(
        [in] FdtUUIDString invokeId,
        [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 invokeId,
            [in] FdtXmlDocument response);
        [id(0x3)]
        HRESULT OnDisconnectResponse(
            [in] FdtUUIDString invokeId,
            [in] FdtXmlDocument response);
        [id(0x4)]
        HRESULT OnTransactionResponse(
            [in] FdtUUIDString invokeId,
            [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 invokeId,
            [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 childsSystemTag,
        [out, retval] VARIANT_BOOL* result);

    [id(0x4)]
    HRESULT ValidateRemoveChild(
        [in] BSTR childsSystemTag,
        [out, retval] VARIANT_BOOL* result);

    [id(0x5)]
    HRESULT OnAddChild(
        [in] BSTR childsSystemTag);

    [id(0x6)]
    HRESULT OnRemoveChild(
        [in] BSTR childsSystemTag);
};

// 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);
};

//
// FDT Container 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 invokeId,
        [in] VARIANT_BOOL success);

    [id(0x5)]
    HRESULT OnDownloadFinished(
        [in] FdtUUIDString invokeId,
        [in] VARIANT_BOOL success);

    [id(0x6)]
    HRESULT OnApplicationClosed(
        [in] FdtUUIDString invokeId);

    [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 invokeId,
        [in] VARIANT_BOOL success);

    [id(0x10)]
    HRESULT OnScanResponse(
        [in] FdtUUIDString invokeId,
        [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 invokeId,
            [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);
};

};

```

““““““ ““““““

11 FDT Quick Reference

11.1 Standard COM Interfaces used by FDT

Following standard COM-interfaces are used by FDT:

DTM:

- IPersistStreamInit
- IPersistPropertyBag
-

FDT-Container:

- IStream
- IPropertyBag

These interfaces are mandatory for implementation of FDT components.

For detail information about these interfaces and their methods, refer to the corresponding Microsoft documentation (MSDN).

11.2 Custom Interfaces of FDT

This section includes a quick reference for the methods on the Custom Interfaces. These interfaces, their parameters and behavior are described in more detail at the chapters of [FDT Interface](#).

Mandatory interfaces for the functional interaction within the FDT architecture

- Idtm
- IdtmInformation
- IDtmEvents
- IfdtCommunication
- IfdtCommunicationEvents
- IfdtEvents

All other interfaces are optional. The implementation depends on the capability of a DTM and its according device.

11.3 Device Type Manager

11.3.1 Interface IDtm

Is called by the frame-application for initialization concerning the current user.

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

Is called by the frame-application to set the systemTag and the back pointer to the frame-application


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

Returns a XML document containing information about standard (defined by applicationID) or additional functionality of a DTM

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

Is called by the frame-application to initialize a newly created instance data set for a specific device type.

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

Starts a function of a DTM.

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

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.

```
HRESULT PrepareToDelete(  
    [out, retval] VARIANT_BOOL* result);
```

Used to inform the DTM that it has to release its links to other components. The DTM will be released by the frame-application.

```
HRESULT PrepareToRelease(  
    [out, retval] VARIANT_BOOL* result);
```

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

```
HRESULT PrepareToReleaseCommunication(  
    [out, retval] VARIANT_BOOL* result);
```

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

HRESULT [PrivateDialogEnabled](#)(
 [in] VARIANT_BOOL enabled,
 [out, retval] VARIANT_BOOL* result);

Used to inform the DTM that the communication pointer is released by the frame-application.

HRESULT [ReleaseCommunication](#)(
 [out, retval] VARIANT_BOOL* result);

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

HRESULT [SetCommunication](#)(
 [in] IfdtCommunication* communication,
 [out, retval] VARIANT_BOOL* result);

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

HRESULT [SetLanguage](#)(
 [in] long languageld,
 [out, retval] VARIANT_BOOL* result);

11.3.2 Interface IDtmActiveXInformation

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

HRESULT [GetActiveXGuid](#)(
 [in] [FdtXMLDocument functionCall](#),
 [out, retval] FdtUUIDString* result);

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

HRESULT [GetActiveXProgId](#)(
 [in] [FdtXMLDocument functionCall](#),
 [out, retval] BSTR* result);

11.3.3 Interface IDtmApplication

Notification to a DTM to close all user interfaces or just a single user interface identified by the invoke id.

HRESULT [ExitApplication](#)(
 [in] FdtUUIDString [invokeld](#),
 [out, retval] VARIANT_BOOL* result);

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

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

11.3.4 Interface IDtmChannel

Returns the channel objects of a DTM.

```
HRESULT GetChannels(  
    [out, retval] IfdtChannelCollection** result);
```

11.3.5 Interface IDtmDocumentation

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

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

11.3.6 Interface IDtmDiagnosis

Returns TRUE if the complete data sets are equal.

```
HRESULT Compare(  
    [out, retval] VARIANT_BOOL* result);
```

Initializes a DTM for comparison of two device instances

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

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

```
HRESULT ReleaseCompare(  
    [out, retval] VARIANT_BOOL* result);
```

Returns TRUE if the complete data set is valid.

HRESULT [Verify](#)(
[out, retval] VARIANT_BOOL* result);

11.3.7 Interface IDtmImportExport

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

HRESULT [Export](#)(
[in] Istream* stream
[out, retval] VARIANT_BOOL* result);

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

HRESULT [Import](#)(
[in] Istream* stream,
[out, retval] VARIANT_BOOL* result);

11.3.8 Interface IDtmInformation

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

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

11.3.9 Interface IDtmOnlineDiagnosis

Returns a XML document containing the result of the compare.

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

Returns XML document which describes the status of the device.

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

11.3.10 Interface IDtmOnlineParameter

Cancels an active parameter-upload or download.

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

Sends the request to write online data to the device.

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

Sends the request to read online data from a device.

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

11.3.11 Interface IDtmParameter

Returns a XML document with the device specific parameters.

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

Sets changes of device specific parameters

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

11.3.12 Interface IFdtCommunicationEvents

Notification that the connection identified by the invoke id has been aborted.

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

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

```
HRESULT OnConnectResponse(  
    [in] FdtUUIDString invokeId,  
    [in] FdtXMLDocument response);
```

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

```
HRESULT OnDisconnectResponse(  
    [in] FdtUUIDString invokeId,  
    [in] FdtXMLDocument response);
```

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

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

11.3.13 Interface IFdtEvents

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

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

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.

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

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

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

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.

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

11.4 DTM ActiveXControl

11.4.1 Interface IdtmActiveXControl

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

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

Used to inform the DTM control that it has to release its links to other components. The frame-application will release the control.

```
HRESULT PrepareToRelease(  
    [out, retval] VARIANT_BOOL* result);
```

11.5 FDT Channel

11.5.1 Interface IFdtChannel

Returns a XML document with fieldbus dependent channel specific parameters.

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

Returns an identifier for a channel.

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

Sets changes of channel specific parameters.

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

11.5.2 Interface IFdtChannelActiveXInformation

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

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

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

```
HRESULT GetChannelActiveXProgId(  
    [in] FdtXMLDocument functionCall,  
    [out, retval] BSTR* result);
```

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

```
HRESULT GetChannelFunctions(  
    [in] FdtXMLDocument operationState,  
    [out, retval] FdtXMLDocument* result);
```

11.5.3 Interface IFdtCommunication

Aborts a communication link to a device without any response. For more than one connection to the same device, the link is identified by the [communicationReference](#).

```
HRESULT Abort([in] FdtXMLDocument fieldbusFrame);
```

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.

```
HRESULT ConnectRequest(  
    [in] IfdtCommunicationEvents* callBack,  
    [in] FdtUUIDString invokeld,  
    [in] FdtUUIDString protocolId,  
    [in] FdtXMLDocument fieldbusFrame,  
    [out, retval] VARIANT_BOOL* result);
```

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.

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

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

```
HRESULT GetSupportedProtocols(  
    [out, retval] FdtXMLDocument * result);
```


The communication component has to observe that the 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.

```
HRESULT SequenceBegin(  
    [in] FdtXMLDocument fieldbusFrame,  
    [out, retval] VARIANT_BOOL* result);
```

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

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

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.

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

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

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

11.5.4 Interface IFdtCommunicationEvents

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

It is identical to [IfdtCommunicationEvents](#) of a DTM. The event interface of a channel is especially used for nested communication and must be provided by gateway-channels only.

11.5.5 Interface IFdtChannelSubTopology

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

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

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

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

Requests the asynchronously scan of the sub topology.

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

Validates if a given device, specified by its systemTag, can be added to the sub-topology.

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

Validates if a given device, specified by its systemTag, can be removed from the sub-topology.

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

11.5.6 Interface IFdtFunctionBlockData

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 Device-FB, located in the bus master.

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

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

Before a DTM of a failsafe device can verify the consistency of the device parameters, the user has to assign the Device-FB in the host (bus master) to the DTM which contains the parameters of the failsafe device.

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

11.6 FDT Channel ActiveXControl

11.6.1 Interface IfdtChannelActiveXControl

Sets the callback pointer of an ActiveX control to the according FdtChannel.

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

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

```
HRESULT PrepareToRelease(  
    [out, retval] VARIANT_BOOL* result);
```

11.7 FDT Container

11.7.1 Interface IDtmEvents

Notification by a DTM, that its user interface identified by the invoke id is closed.

```
HRESULT OnApplicationClosed(  
    [in] FdtUUIDString invokeld);
```

Notification by a DTM, that the asynchronous download function call identified by the invoke id is finished.

```
HRESULT OnDownloadFinished(  
    [in] FdtUUIDString invokeld,  
    [in] VARIANT_BOOL success);
```

A DTM sends a notification about errors during an asynchronously function call.

```
HRESULT OnErrorMessage(  
    [in] BSTR systemTag,  
    [in] BSTR errorMessage);
```

Notification of a DTM that the information about its current available additional functionality has changed

```
HRESULT OnFunctionChanged(  
    [in] BSTR systemTag);
```

Notification of a DTM that the information about channel related functionality has changed.

```
HRESULT OnChannelFunctionChanged(  
    [in] BSTR systemTag,  
    [in] FdtXPath channelPath);
```

Notification by a DTM, that the asynchronously invoked function call identified by the invoke id is finished.

```
HRESULT OnInvokedFunctionFinished(  
    [in] FdtUUIDString invokeId,  
    [in] VARIANT_BOOL success);
```

A DTM sends an notification to navigate to a frame-specific application.

```
HRESULT OnNavigation(  
    [in] BSTR systemTag);
```

A DTM sends a notification about its online state.

```
HRESULT OnOnlineStateChanged(  
    [in] BSTR systemTag,  
    [in] VARIANT_BOOL onlineState);
```

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

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

A DTM sends a notification that it can be released.

```
HRESULT OnPreparedToRelease(  
    [in] BSTR systemTag,);
```

A DTM sends a notification that its communication pointer can be released.

```
HRESULT OnPreparedToReleaseCommunication(  
    [in] BSTR systemTag);
```

A DTM sends a notification that it wants to print a DTM specific document.

```
HRESULT OnPrint(  
    [in] BSTR systemTag,  
    [in] FdtXMLDocument functionCall);
```

A DTM sends a notification about the progress on handling of an asynchronously function call.

```
HRESULT OnProgress(  
    [in] BSTR systemTag,  
    [in] BSTR title,  
    [in] short percent,  
    [in] VARIANT_BOOL show);
```

Returns a list of fieldbus related information to identify the connected devices.

```
HRESULT OnScanResponse(  
    [in] FdtUUIDString invokeId,  
    [in] FdtXMLDocument response);
```

Notification by a DTM, that the asynchronous upload function call identified by the invoke id is finished.

```
HRESULT OnUploadFinished(  
    [in] FdtUUIDString invokeId,  
    [in] VARIANT_BOOL success);
```

11.7.2 Interface IDtmAuditTrailEvents

Notification by a DTM about changed data to be recorded by the frame-application's audit trail tool.

```
HRESULT OnAuditTrailEvent(  
    [in] BSTR systemTag,  
    [in] FdtXMLDocument logEntry);
```

A DTM sends a notification to close the audit trail sequence.

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

Notification by a DTM that the following changes should be recorded by the frame-application's audit trail tool.

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

11.7.3 Interface IFdtActiveX

A DTM sends a request to close one of its ActiveX controls.

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

Request of a DTM to start a DTM functionality defined by the function call id

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

11.7.4 Interface IFdtBulkData

Returns the instance related path for bulk data.

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

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).

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

11.7.5 Interface IFdtContainer

Returns a path where the default schemas are stored

```
HRESULT GetXmlSchemaPath  
    [out, retval] BSTR* result);
```

A DTM sends a notification to the frame-application that it wants to have unique read/write access for the currently loaded data set.

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

Informs the frame-application that it should store the changed data.

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

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.

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

11.7.6 Interface IFdtDialog

Call the Container to display a message

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

11.7.7 Interface IFdtTopology

A DTM send a request to the frame-application to create a new instance data set of the specified device type.

```
HRESULT CreateChild(  
    [in] FdtXMLDocument deviceType,  
    [in] FdtXPath channelPath,  
    [out, retval] BSTR* result);
```

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

```
HRESULT DeleteChild(  
    [in] BSTR systemTag,  
    [in] FdtXPath channelPath,  
    [out, retval] VARIANT_BOOL* result);
```

Returns a XML document containing the systemTags of all child DTMs of the DTM identified by its system tag and the channel path.

```
HRESULT GetChildNodes(  
    [in] BSTR systemTag,  
    [in] FdtXPath channelPath,  
    [out, retval] FdtXMLDocument* result);
```

Return the associated DTM according the given system tag

```
HRESULT GetDtmForSystemTag(  
    [in] BSTR systemTag,  
    [out, retval] Idtm** result);
```

Returns an XML-document containing a list of DTM related information. This information is provided via the DtmInfo-structure defined within the [DTMInformationSchema](#).

```
HRESULT GetDtmInfoList(  
    [out, retval] FdtXMLDocument * result);
```

Returns an XML-document containing al list of system tags of all parent DTMs of the DTM identified by its system tag.

```
HRESULT GetParentNodes(  
    [in] BSTR systemTag,  
    [out, retval] FdtXMLDocument* result);
```

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.

```
HRESULT MoveChild(  
    [in] BSTR systemTag,  
    [in] FdtXPath destinationChannelPath,  
    [out, retval] VARIANT_BOOL* result);
```

Release the associated DTM according the given system tag

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


12 Appendix — Implementation Hints

12.1 Initialization of a new DTM instance by calling IpersistXX::InitNew

The IpersistXX::InitNew method is to be called by the frame-application for a DTM object instantiated for a new fieldbus device instance. A DTM must initialize default parameter settings within this method.

12.2 Implementation of Ipersist Interfaces using Visual Basic

DTMs implemented using Visual Basic provide the IpersistPropertyBag and IpersistStreamInit interfaces automatically if the “Presentable” property of the appropriate source class object is enabled. But the implementation provided by the Visual Basic runtime has the following side effects:

- Either the IPersistXXX::Load or IPersistXXX::InitNew can only be called once by a client to load or initialize the appropriate COM object.
- If Load or InitNew is called again a COM error is raised.

12.3 Instantiation of DTMs using Visual Basic

A Visual Basic New() or CreateObject() automatically calls the IPersistXXX::InitNew() method after instantiation of a COM object. A frame-application implemented using Visual Basic must therefore use a factory implemented in Visual C++ to instantiate a DTM object and load it with its instance data via IPersistXXX.

Alternatively, a frame-application may use the Visual Basic PropertyBag object to save the private data of a DTM together with the COM object itself [MSDN “Persisting a Component's Data”].

12.4 Access to Ipersist Interfaces using Visual Basic

A frame-application written in Visual Basic must use wrapper classes to access Istream and IpropertyBag.

12.5 Access to Import/Export Interface using Visual Basic

To access the Istream interface using Visual Basic a C++ wrapper class is required. Such a wrapper class must copy Visual Basic byte safe arrays to and from the Istream interface.

```

// read contents of istream into an Safearray of bytes
STDMETHODIMP CVBIStream::Read(IStream *ip, SAFEARRAY **buffer)
{
    // build safearray: one dimension, size of elements is 1 (byte)
    unsigned int ndim = 1;
    HRESULT hresult = SafeArrayAllocDescriptor(ndim, buffer);
    if( FAILED(hresult))
        return E_FAIL;
    (*buffer)->cbElements = 1;           // contains bytes
    (*buffer)->rgsabound[0].cElements = 0;
    (*buffer)->rgsabound[0].lbound = 0;

    // read data from stream and copy it to safearray
    if ( ip )
    {
        ULARGE_INTEGER available;
        available.QuadPart = 0;
        unsigned long read = 0;
        unsigned long nbytes;
        // seek to start of stream
        LARGE_INTEGER seekpos;
        seekpos.QuadPart = 0;
        ip->Seek( seekpos, STREAM_SEEK_SET, &available);
        // get size of data in stream
        ip->Seek( seekpos, STREAM_SEEK_END, &available);
        nbytes = available.QuadPart;
        // seek to start of stream
        ip->Seek( seekpos, STREAM_SEEK_SET, &available);
        if ( nbytes > 0 )
        {
            // alloc data: number of bytes is size of istream
            (*buffer)->rgsabound[0].cElements = nbytes;
            (*buffer)->rgsabound[0].lbound = 0;
            if ( SUCCEEDED(SafeArrayAllocData(*buffer)) )
                // copy it from stream
                ip->Read( (*buffer)->pvData, nbytes, &read);
            else
                return E_FAIL;
        }
    }
    return S_OK;
}

// write contents of a safearray of bytes to istream
STDMETHODIMP CVBIStream::Write(IStream *ip, SAFEARRAY **buffer)
{
    // must be one dimensional array of bytes
    long cbelements = (*buffer)->cbElements;           // must be 1 == sizeof(byte)
    long dims = (*buffer)->cDims;                       // must be 1
    if ( dims != 1 || cbelements != 1 )
        return E_FAIL;

    // write number of bytes in safearray to stream
    long celements = (*buffer)->rgsabound[0].cElements;
    if ( ip )
    {
        unsigned long written;
        ip->Write( (*buffer)->pvData, celements, &written);
    }
    return S_OK;
}

```

Typically a Visual Basic PropertyBag object can be used to save and restore export and import data within the DTM VB code. The PropertyBag::contents information can then be saved or restored via the IdtmImportExport interface using such a wrapper class.

```

' Im/Export interface of DTM
' use a standard propertybag object to save and load information
' use this propertybag and copy contents to istream interface using
' the VBISStream wrapper

Private Sub IdtmImportExport_Import(ByVal ip As IStream)
    ' use a propertybag. Is filled via the istream interface
    Dim exbag As New PropertyBag
    ' Read it from stream
    Dim istream As New VBISStream ' wrapper to access istream
    ' copy info of istream to propertybag object
    exbag.Contents = istream.Read(ip)
    ' use propertybag to read import data
    Class_ReadProperties exbag

```

```

    isImport = False
End Sub
Private Sub IdtmImportExport_Export(ByVal ip As Istream)
    Dim exbag As New PropertyBag
    ' Fill Propertybag
    Class_WriteProperties exbag
    ' now use istream wrapper to copy data from propertybag to
    ' istream
    Dim istream As New VBStream
    istream.Write ip, exbag.Contents
End Sub

```

12.6 Implementation of Stream objects using Visual Basic

Using Visual Basic the Istream interface cannot be implemented directly. Instead a C++ factory must be used to create such stream objects, e.g. by calling `::CreateStreamOnHGlobal()`. Then the contents of the stream object can be accessed by using the C++ wrapper class (see above).

12.7 Usage of Microsoft XML-parser

For parsing a XML document the MSXML2 library (Microsoft XML, V3.0 / July 2000 MSXML Beta Release) or higher versions should be used

12.8 DTM Documentation with Graphical Elements

The following example shows how graphical elements like bitmaps can be converted to the MIME format and vice versa. Such an algorithm is used for the embedding of graphical elements within a XML document transferred at the [IdtmDocumentation](#) interface.

FDTImageSchema-Example

```

<Schema name="FDTImageSchema" 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="mime-type" dt:type="string"/>
  <AttributeType name="data" dt:type="bin.base64"/>
  <AttributeType name="height" dt:type="ui1"/>
  <AttributeType name="width" dt:type="ui1"/>
  <!--Definition of Elements-->
  <ElementType name="Image" content="mixed" model="closed">
    <attribute type="mime-type" required="yes"/>
    <attribute type="data" required="yes"/>
    <attribute type="height" required="yes"/>
    <attribute type="width" required="yes"/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <group order="one">
      <element type="Image"/>
    </group>
  </ElementType>
</Schema>

```

FDTImageTemplate-Example

```

<?xml version="1.0" encoding="UTF-8"?>
<FDT xmlns="x-schema:FDTImageSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <Image mime-type="image/gif" data="" height="0" width="0"/>
</FDT>

```

Sending a XML-Image

```
Private Sub SendXMLImage()  
    Dim bytBuffer() As Byte  
    Dim objXMLDom As MSXML2.DOMDocument  
    Dim objXMLNode As MSXML2.IXMLDOMNode  
  
    'load image file  
    Open "MS.jpg" For Binary As #1  
    ReDim bytBuffer(LOF(1))  
    Get #1, , bytBuffer  
    Close #1  
  
    Set objXMLDom = New MSXML2.DOMDocument  
    objXMLDom.Load App.Path & "\FDTImageTemplate.xml"  
  
    Set objXMLNode = objXMLDom.getElementsByTagName("Image").Item(0)  
    'convert image bytes to bin.base64 byte array  
    objXMLNode.Attributes.getNamedItem("data").nodeTypedValue = bytBuffer  
    'set XML file attributes  
    objXMLNode.Attributes.getNamedItem("mime-type").nodeTypedValue = "image/jpeg"  
    objXMLNode.Attributes.getNamedItem("height").nodeTypedValue = 100  
    objXMLNode.Attributes.getNamedItem("width").nodeTypedValue = 100  
  
    objXMLDom.save App.Path & "\Image.xml"  
    Set objXMLDom = Nothing  
  
    ReceiveXMLImage App.Path & "\Image.xml"  
End Sub
```

Receiving a XML-Image

```

Private Sub ReceiveXMLImage(strXMLFile As String)
    Dim bytBuffer() As Byte
    Dim objXMLDom As MSXML2.DOMDocument
    Dim objXMLNode As IXMLDOMNode
    Dim objXMLNodes As IXMLDOMNodeList

    Set objXMLDom = New MSXML2.DOMDocument

    'load XML document
    objXMLDom.Load strXMLFile

    'get Image XML node
    Set objXMLNode = objXMLDom.getElementsByTagName("FDT/Image").Item(0)

    'convert bin.base64 array to image byte array
    bytBuffer = objXMLNode.Attributes.getNamedItem("data").nodeValue
    'show image attributes
    Debug.Print "mime-type= " & objXMLNode.Attributes.getNamedItem("mime-type").nodeValue
    Debug.Print "height = " & objXMLNode.Attributes.getNamedItem("height").nodeValue
    Debug.Print "width = " & objXMLNode.Attributes.getNamedItem("width").nodeValue

    'write image bytes to temp file
    Open App.Path & "\Temp.jpg" For Binary As #1
    Put #1, , bytBuffer
    Close #1
    'load temp image file and show it to user
    imgDest = LoadPicture(App.Path & "\Temp.jpg")
End Sub

```

12.9 Microsoft Category Manager access

The following example shows how to access the Microsoft Category Manager using a C++ wrapper. Such a wrapper is necessary for programming languages which do not support a direct access.

```

#ifndef __FDTCategoryManager_H
#define __FDTCategoryManager_H
#include "resource.h" // main symbols
#include <comcat.h>

// FDT COM Categories definition

static const GUID CATID_FDT_DTM = {0x36D1490, 0x387B, 0x11D4, {0x86, 0xE1, 0x0, 0xE0, 0x98, 0x72, 0x70, 0xB9}};

// ... define all categories according to the specification

// The FDTCategoryManager can be used to searches of supported FDT Categories of COM Classes.

```

```

// The result of a search is a Collection of the corresponding ProgIDs of these classes.
//

class ATL_NO_VTABLE FDTCategoryManager :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<FDTCategoryManager, &CLSID_FDTCategoryManager>,
    public IDispatchImpl<IFDTCategoryManager, &IID_IFDTCategoryManager,
    &LIBID_FDTUTILITIESFORVBLib>
{
// -----
// data:

protected :
    // Standart Microsoft Component Category Manager
    cComPtr<ICatInformation> m_oCatManager;

// -----
// methods:

protected :

    // Constructor
    FDTCategoryManager();

    // Destructor
    virtual ~FDTCategoryManager();

public :
    // Overriden ATL FinalConstruct methode.
    // Method creates the internal used Standart Microsoft Category Manager object.
    //
    // PARAMETER:
    //
    // RETURN:
    //      S_OK, if object successful created
    //
    HRESULT FinalConstruct();

// -----
// interfaces:

public:
// IFDTCategoryManager

    // Returns the ProgIDs of the COM classes which supports the specified
    // FDT COM Category
    //
    // PARAMETER:
    //      Category : FDT Category to search for
    //      ProgIDs      : Collection of ProgIDs (strings) of COM Classes
    //
    // RETURN:
    //      S_OK, if no error
    //      E_INVALIDARG, if eCategory parameter is not a valid EFdtCategories value
    //      E_POINTER
    //
    STDMETHOD(EnumClassesOfCategory)(EFdtCategories eCategory, IVariantDictionary** ProgIDs);

    // Tests COM class (corresponding ProgID) wheter the given FDT COM Category is
    // supported or not.
    //
    // PARAMETER:
    //      bstrProgID:      ProgID of the COM class
    //      eCategory:      FDT COM category to test for
    //      pbResult: Result of the test (TRUE = Class supports FDT COM Category)
    //
    // RETURN:
    //      S_OK, if no error
    //      E_INVALIDARG, if eCategory parameter is not a valid EFdtCategories value

```

```

        //      E_POINTER
        //
        STDMETHOD(IsClassOfCategory)(BSTR bstrprogID, EFdtCategories eCategory, VARIANT_BOOL
*pbResult);

// -----
// ATL implementations

pub lic:
    DECLARE_REGISTRY_RESOURCEID(IDR_FDTCATEGORYMANAGER)
    DECLARE_PROTECT_FINAL_CONSTRUCT()

    BEGIN_COM_MAP(FDTCategoryManager)
        COM_INTERFACE_ENTRY(IFDTCategoryManager)
        COM_INTERFACE_ENTRY(IDispatch)
    END_COM_MAP()

}; // End FDTCategoryManager

#endif // __FDTCategoryManager_H

// EOF FDTCategoryManager.H

#include "stdafx.h"
#include "FDTUtilitiesForVB.h"
#include "FDTCategoryManager.h"

FDTCategoryManager::FDTCategoryManager()
{
}

HRESULT FDTCategoryManager::FinalConstruct()
{
    // create Category Manager object
    return m_oCatManager.CoCreateInstance(CLSID_StdComponentCategoriesMgr);
}

FDTCategoryManager::~FDTCategoryManager()
{
    if (m_oCatManager.p)
        m_oCatManager.Release();
}

////////////////////////////////////
// IFDTCategoryManager interface

STDMETHODIMP FDTCategoryManager::EnumClassesOfCategory(eFdtCategories eCategory, IVariantDictionary**
ProgIDs)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState())

    if (ProgIDs == NULL)
        return E_POINTER;

    //-----
    // search ClassID of COM classes which implements CATID

    HRESULT hr;
    CComPtr<IEnumCLSID> oEnumCLSID;

    // select FDT COM Category
    GUID oCategory;
    switch(eCategory)
    {
        case eCATID_FDT_DTM:    oCategory = CATID_FDT_DTM; break;
        // case for all categories according to the specification
    }

```

```

        default: // Parameter value unknown ==> return error
                return E_INVALIDARG;
    }

    // search COM classes which support FDT COM Category
    hr = m_oCatManager->EnumClassesOfCategories(1, &oCategory, -1, NULL, &oEnumCLSID);
    if (FAILED(hr))
        return hr;

    //-----
    // convert ClassIDs to ProgIDs and add them to the result collection

    GUID oCLSID;
    CComBSTR oProgID;
    LPOLESTR poOLEProgID;
    CComVariant oVariant;
    unsigned long ulFetched;

    // create progID result dictionary
    CComPtr<IVariantDictionary> oEnumProgIDs;
    hr = oEnumProgIDs.CoCreateInstance(CLSID_VariantDictionary);
    if (FAILED( hr ))
        return hr;

    // convert all founded ClassID to ProgIDs and add them to the result collection
    VERIFY( SUCCEEDED( oEnumCLSID->Next(1, &oCLSID, &ulFetched) ));
    while ( ulFetched )
    {
        // convert the ClassId to a ProgID
        poOLEProgID = NULL;
        ProgIDFromCLSID(oCLSID, &poOLEProgID);

        // Convert to BSTR
        oProgID = poOLEProgID;
        CoTaskMemFree(poOLEProgID);

        // insert ProgID to result collection
        oVariant.Clear();
        oVariant = (BSTR)oProgID;
        VERIFY(SUCCEEDED( oEnumProgIDs->Add( &oVariant) ));

        // go to next ClassID
        VERIFY(SUCCEEDED( oEnumCLSID->Next(1, &oCLSID, &ulFetched) ));
    }

    // return ProgID collection
    *ProgIDs = oEnumProgIDs.Detach();
    return S_OK;
}

STDMETHODIMP FDTCategoryManager::IsClassOfcategory(BSTR bstrProgID, EFdtCategories eCategory,
VARIANT_BOOL *pbResult)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState())

    if (pbResult == NULL)
        return E_POINTER;

    //-----
    // search ClassID of COM classes which implements CATID

    HRESULT hr;
    CComPtr<IEnumCLSID> oEnumCLSID;

    // select FDT COM Category
    GUID oCategory;
    switch(eCategory)
    {
        case eCATID_FDT_DTM: oCategory = CATID_FDT_DTM; break;
        // case for all categories according to the specification

        default: // Parameter value unknown ==> return error
                return E_INVALIDARG;
    }
}

```



```

// try to convert ProgID to a ClassID
GUID oCLSID;
hr = CLSIDFromProgID(bstrProgID, &oCLSID);
if (FAILED( hr ))
    return hr;

// if COM class supports specified FDT COM Category ==> return result TRUE
if (S_OK == m_oCatManager->IsClassOfCategories(oCLSID, 1, &oCategory, -1, NULL))
    *pbResult = VARIANT_TRUE;
else
    *pbResult = VARIANT_FALSE;

return S_OK;
}

// EOF FDTCategoryManager.CPP

```

12.10 DTMs running in Single Threaded Apartment

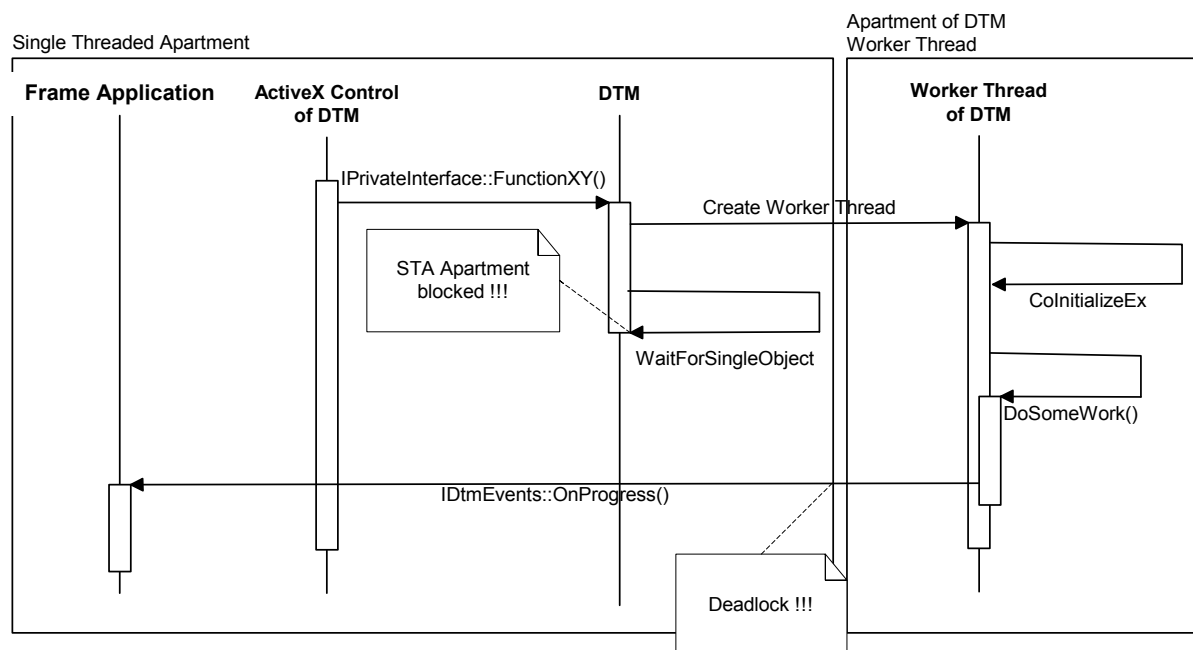
DTMs are COM objects instantiated and running in a COM Apartment.

A DTM running in a Single Threaded Apartment (STA) is not allowed to block its thread, because a blocked STA thread can't dispatch client messages to other COM objects living in the same Apartment. A thread that doesn't dispatch messages will leak small amounts of memory and also can cause deadlocks.

For a detailed description of the OLE Threading Models and the reason why a STA thread should not be blocked look to MSDN article "Q150777" and "Q136885".

Example for a caused deadlock if a DTM blocks a STA thread:

The following scenario shows a DTM that creates an internal working thread to perform some work, if an ActiveX control of the DTM starts a private function. The DTM waits for the end of the worker thread by calling WIN API method call *WaitForSingleObject()* and therefore blocks its STA. If the worker thread tries to call a method of an object living in the STA, it causes a deadlock because the STA doesn't dispatch the call to this object.



Source Code of DTM method blocking the STA:

```
// Some private function of the DTM
STDMETHODIMP IPrivateInterface::FunctionXY()
{
    // create worker thread
    CWorkerThread *poWorkerThread;
    poThread = (WorkerThread*)AfxBeginThread(RUNTIME_CLASS(WorkerThread));

    // wait for worker thread end
    WaitForSingleObject(poThread->m_hThread, INFINITE);

    return S_OK;
}
```

The problem can be fixed by replacing the WIN API method *WaitForSingleObject()* with *MsgWaitForSingleObjects()* and a message loop function as in next example.

```
// Some private function of the DTM
STDMETHODIMP IPrivateInterface::FunctionXY()
{
    DWORD dwRet;
    MSG msg;

    // create worker thread
    CWorkerThread *poWorkerThread;
    poThread = (WorkerThread*)AfxBeginThread(RUNTIME_CLASS(WorkerThread));

    while(1) {
        dwRet = MsgWaitForMultipleObjects( 1, // One event to wait for
                                           &hEvent, // The array of events
                                           FALSE, // Wait for 1 event
                                           INFINITE, // Timeout value
                                           QS_ALLINPUT); // Any message wakes up

        if(dwRet == poThread->m_hThread)
        {
            // The end of the worker thread has been signaled
            return S_OK;
        }
        else if (dwRet == WAIT_OBJECT_0 + 1)
        {
            // There is a window message available. Dispatch it.
            while(PeekMessage(&msg, NULL, NULL, NULL, PM_REMOVE))
            {
                else
                {
                    TranslateMessage(&msg);
                    DispatchMessage(&msg);
                }
            }
        }

        return S_OK;
    }
}
```

13 Appendix – FDT XML Schemas

13.1 FDT Data Types

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.

	Description
alarmType	Identifier for the alarm type to show the association between high- and low alarm and high-high- and low-low-alarm
binData	Variable containing binary data
bitLength	Length of a binary variable as bit count
busCategory	Unique identifier for a supported bus type like Profibus or HART according to the FDT specific CATID
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
classificationId	Unique identifier for classification of a channel or device like flow or temperature
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
dataSetState	State of an instance data set concerning modifications (refer to chapter 5.13.1.1 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
deviceTypeId	Unique identifier for a device type within the name space of the fieldbus specification
deviceTypeInfo	Additional device type information supplied with a device. A Profibus device has to provide its GSD information as human readable string at this attribute
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)
errorCode	Status information according to the Profibus specification
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
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
readAccess	Specifies whether the value can be read from a device
languageId	Identifier for a supported language or the language a DTM should

Attribute	Description
	interact in according to the Microsoft standard. See also IDtm::SetLanguage ()
manufacturerId	Unique identifier for a manufacturer within the name space of the fieldbus specification
number	Number variable like float, integer or other numeric data types
path	Path to the icon for a device
reference	Reference to a variable of a structure
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 5.13.1.2 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 IDtm::Init() 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
tag	Unique identifier for a device, module, or channel
time	Time variable within an element
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
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
BusCategory	Description of busCategory
BusCategories	Collection of BusCategory
ChannelReference	Reference to an object identified by its id
ChannelReferences	Collection of references
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 gateway-channel's device and optional error description
DeviceIcon	Icon for a device
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
Languageld	Contains the languageld (refer to languageld)
LowerRange	Defintion of a lower range element
NumberData	Number variable like float, integer or other numeric data types
Range	Describes the valid range of a process variable
Ranges	Collection of ranges

Tag	Description
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
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
Value	Contains the display string for a variable or the variable itself
Variable	Selected element of an enumeration
Variant	Variable with data type and display format
VersionInformation	Description of the version of a component

```
<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"/>
```

```
  <AttributeType name="communicationError" dt:type="enumeration" dt:values="abort busy invalidCommunicationReference noConnection noParallelServices noPendingRequest unknownError timeout dtmSpecific"/>
```

```
  <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="uuid"/>
<AttributeType name="substituteType" dt:type="enumeration" dt:values="lastValue lastValidValue upperRange
lowerRange"/>
<AttributeType name="path" dt:type="uri"/>
<!-- Definition of Elements -->
<ElementType name="DeviceIcon" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <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>
<!-- Definition of Buscategories -->
<ElementType name="BusCategory" content="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="busCategory" required="yes"/>
</ElementType>
<ElementType name="BusCategories" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="BusCategory" minOccurs="1" maxOccurs="*/>
</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="classificationId" 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"/>
</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="*/"/>
</ElementType>
<ElementType name="BitEnumeratorEntries" content="eltOnly" model="closed">
  <attribute type="nodeId" required="no"/>
  <element type="EnumeratorEntry" minOccurs="1" maxOccurs="*/"/>
</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="*/"/>
</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="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="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"/>
</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="empty" model="closed">
  <attribute type="nodeId" required="no"/>
  <attribute type="idref" required="yes"/>
</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"/>
            <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"/>
    <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"/>
    <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>

```

13.2 FDT Application Id

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"/>
  <!-- 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](#)

13.3 FDT User Information

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.

	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"/>
    <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>
```

13.4 DTM Information

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 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 it must be set to '2'

Tag	Description
DtmDeviceTypes	Collection of device types
DtmInfo	Describes the DTM itself
FDTVersion	Definition of the element which specifies the version information on which the implementation of a DTM is based on
FDTRoot 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"/>
  <!--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"/>
  </ElementType>
  <ElementType name="DtmDeviceTypes" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="fdt:DtmDeviceType" minOccurs="1" maxOccurs="*/>
  </ElementType>
  <ElementType name="DtmInfo" content="eltOnly" model="closed" order="seq">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="FDTVersion" minOccurs="1" maxOccurs="1"/>
    <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1"/>
    <element type="DtmDeviceTypes" minOccurs="1" maxOccurs="*/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="DtmInfo" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>
```

Example:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMInformationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
xmlns:fdtappid="x-schema:FDTApplicationIdSchema.xml">
  <DtmInfo>
    <FDTVersion major="1" minor="2"/>
    <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
    <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:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
        </fdt:BusCategories>
      </fdt:DtmDeviceType>
    </DtmDeviceTypes>
  </DtmInfo>
</FDT>
```

13.5 DTM Functions

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>
```

13.6 Parameter access

Used at: [IDtmParameter::GetParameters\(\)](#)

[IDtmParameter::SetParameters\(\)](#)

The XML document provides all instance specific information about a device.

	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	Description of bus information of a specific device
DtmDevice	Description of a device instance
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

Tag	Description
	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

```

<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"/>
    <group order="one" minOccurs="0" maxOccurs="1">
      <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"/>
        <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 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:Languageld languageld="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:Languageld languageld="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:ChannelReference idref="AI4.C3"/>
      <fdt:ChannelReference idref="AI4.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:DtmVariable name="Integer2">
          <fdt:Value>
            <fdt:Variant dataType="int">
              <fdt:NumberData number="3"/>
            </fdt:Variant>
          </fdt:Value>
        </fdt:DtmVariable>
      </fdt:DtmVariables>
    </ExportedVariables>
  </DtmDevice>
</FDT>

```

```
</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>
```

13.7 Documentation

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="*" />
    </group>
  </ElementType>

```



```

        <element type="DocumentVariable" minOccurs="0" maxOccurs="*" />
        <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:nodel" 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" />
    <element type="fdt:VersionInformation" minOccurs="1" 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/xml" href="DTMDocumentationStyle.xml"?>
<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"/>
    </DocumentVariables>
</FDT>

```



```
<GraphicReference title="GraphicReferenceTitle" fdt:descriptor="Descriptor GraphicReference"
path="device.gif"/>
</DocumentVariables>
<DTMStyleForCompleteDocument>...</DTMStyleForCompleteDocument>
<DTMSpecificXMLData>...</DTMSpecificXMLData>
</FDT>
```

13.8 Supported Fieldbus Protocols

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>
```

13.9 Topology

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 systemTag

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"/>
    <group order="one" minOccurs="1" maxOccurs="1">
      <element type="DtmParentList"/>
      <element type="DtmChildList"/>
    </group>
  </ElementType>
</Schema>
```

```

    </ElementType>
</Schema>

```

IFdtTopology::GetParentNodes():

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMSysTagListSchema.xml">
  <DtmParentList>
    <DtmPrimaryParentSysTag systemTag = "DTM-1"/>
    <DtmSysTagList>
      <DtmSysTag systemTag = "DTM-2"/>
      <DtmSysTag systemTag = "DTM-3"/>
      <DtmSysTag systemTag = "DTM-4"/>
    </DtmSysTagList>
  </DtmParentList>
</FDT>

```

Example IFdtTopology::GetChildNodes ():

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMSysTagListSchema.xml">
  <DtmChildList>
    <DtmSysTagList>
      <DtmSysTag systemTag = "DTM-2"/>
      <DtmSysTag systemTag = "DTM-3"/>
      <DtmSysTag systemTag = "DTM-4"/>
    </DtmSysTagList>
  </DtmChildList>
</FDT>

```

13.10 Audit Trail

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"/>
    <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>
</Schema>
```

```

</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>

```

13.11 Device Status

Used at: [IDtmOnlineDiagnosis::GetDeviceStatus\(\)](#)

The XML document contains the description of the current status of a device.

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-->
  <ElementType name="DtmDeviceStatus" content="mixed" model="closed">
    <attribute type="fdt:nodeld" required="no"/>
    <attribute type="fdt:statusFlag" required="yes"/>
    <element type="fdt:StatusInformation" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeld" 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">
    <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>
```

13.12 Functions of a DTM

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 <code>IDtmDocumentation::GetDocumentation()</code>).
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

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 document created 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"/>
<!--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:nodeId" 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:nodeId" 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:nodeId" 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:nodeId" 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">
    <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>
        <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>

```

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">
  <Functions label="Functions" fdt:name="" help="">
    <!--Definition of a group of function entries bwlow 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)-->
    <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="C:\MyDTMPath\Extra.hlp">
        <OpenWith programName="WinHelp.exe"/>
      </Document>
    </Functions>
  </Functions>
</FDT>
```

13.13 Functions of a DTM Channel Object

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
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 <code>IDtmDocumentation::GetDocumentation()</code>).
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

Attribute	Description
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

	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 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="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:nodeld" 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="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"/>
        <element type="Status" minOccurs="0" maxOccurs="1"/>
        <group order="many" minOccurs="0" maxOccurs="1">
            <element type="Function" minOccurs="0" maxOccurs="1"/>
            <element type="Functions" minOccurs="0" maxOccurs="1"/>
        </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"/>
        <element type="Status" minOccurs="0" maxOccurs="1"/>
        <group order="many" minOccurs="0" maxOccurs="1">
            <group order="one" minOccurs="0" maxOccurs="1">
                <element type="Function" minOccurs="0" maxOccurs="1"/>
                <element type="Document" minOccurs="0" maxOccurs="1"/>
                <element type="StandardFunction" minOccurs="0" maxOccurs="1"/>
            </group>
            <group order="one" minOccurs="0" maxOccurs="1">
                <element type="Functions" minOccurs="0" maxOccurs="1"/>
                <element type="StandardFunctions" minOccurs="0" maxOccurs="1"/>
            </group>
        </group>
    </ElementType>
    <ElementType name="FDT" content="eltOnly" model="closed">
        <attribute type="fdt:nodeld" 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>
```

13.14 Online Compare of DTM Data

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>
```


13.15 Failsafe

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:nodel" 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:nodel" 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>
```

13.16 Scan

Used at: [IFdtChannelSubTopology::OnScanResponse\(\)](#)

The XML document contains the result of the topology scan

Tag	Description
TopologyScan	Collection of bus specific information concerning the scan result
FDT	Root tag

```
<?xml version="1.0"?>
<Schema name="DTMTopologyScanSchema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:fdtprodevice="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:nodeld" required="no"/>
    <group order="one">
      <element type="fdthartdevice:HARTDevice" minOccurs="0" maxOccurs="*" />
      <element type="fdtprodevice: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>
```

13.17 Operation Phase

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"/>
  <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"/>
```

13.18 DTM Initialization

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:Languageld languageld="1131"/>
    </fdt:SupportedLanguages>
    <fdt:BusCategories>
      <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
    </fdt:BusCategories>
  </fdt:DtmDeviceType>
</FDT>
```

13.19 User Message

Used at: `IFdtDialog::UserDialog()`

The XML document contains the definition of a user message.

	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:nodel" required="no"/>
    <attribute type="fdt:string" required="yes"/>
  </ElementType>
  <ElementType name="FDTUserMessage" content="mixed" model="closed">
    <attribute type="fdt:nodel" 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>
    <attribute type="resultMessage" required="no"/>
    <attribute type="resultStatus" required="no"/>
  </ElementType>
</Schema>
```

```

</ElementType>
<ElementType name="FDT" content="mixed" model="closed">
  <attribute type="fdt:nodeld" 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>

```

13.20 DTM Information List

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:nodeId" required="no"/>
    <element type="dtmInfo:DtmInfo" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" 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>
```


14 Appendix – FDT XML Styles

14.1 Documentation

Used at: IDtmDocumentation::GetDocumentation()

The XML document contains the standard FDT Style for documentation.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl" 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/>
    <TABLE>
      <xsl:for-each select="FDT/DocumentVariables">
        <xsl:apply-templates select="DocumentVariable"/>
      </xsl:for-each>
    </TABLE>
    <HR/>
    <xsl:apply-templates select="FDT/DTMStyleForCompleteDocument"/>
    <xsl:apply-templates select="FDT/DTMSpecificXMLData"/>
  </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>
          </TABLE>
        </TD>
      </TR>
    </TABLE>
  </xsl:template>
</xsl:stylesheet>
```

```

        </TD>
    </TR>
    <TR>
        <TD>Classification:</TD>
        <TD>
            <xsl:value-of select="@fdt:classificationId"/>
        </TD>
    </TR>
    <TR>
        <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">
    <IMG>
        <xsl:attribute name="SRC">
            <xsl:value-of select="DocumentVariables/GraphicReference/@path"/>
        </xsl:attribute>
    </IMG>

```

```

                </xsl:attribute>
                <xsl:attribute name="width">
200
                </xsl:attribute>
                </IMG>
            </TD>
        </TR>
    </TABLE>
    <HR/>
</xsl:template>
<xsl:template match="DocumentVariable">
    <TR>
        <TD>
            <xsl:value-of select="@fdt:name"/>
        </TD>
    </TR>
    <TR>
        <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>
        <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/@lowerValue"/>
        </TD>
    </TR>
    <TR>
        <TD>Upper Value:</TD>
        <TD>
            <xsl:value-of select="/fdt:Range/@upperValue"/>
        </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:stylesheet>

```

15 Appendix – Profibus

PROFIBUS schemas are required to define the structure and semantics of the protocol-specific data transferred via XML documents on the FDT interfaces.

The schemas are based on definitions given in the PROFIBUS-Specification. Furthermore, they contain additional information about the device that is needed by systems to configure PROFIBUS links and to establish communication between the PROFIBUS master device and the PROFIBUS slave devices.

Configuration:

The configuration of the device itself is done with the aid of the DTM's GUI. Downloading the configuration into the slave device is performed via the PROFIBUS master device. To do that and in order to set up the bus communication the master needs information from the DTM as there is:

- GSD file
The GSD information is type-specific information and not instance-specific. It is not stored with single slave instances or in a global accessible file. It is provided by the DTM at [IDtmInformation](#). On method [GetInformation\(\)](#), a DTM of a PROFIBUS device provides the GSD information within its XML document.
The master device can use the general type-specific information from the slave's GSD information like bus timing parameters, supported baud rates etc.
- CFG-String (Cfg_Data)
The CFG-String provides the instance-specific information about the current configuration of the device. It defines the structure of the data frames that will be transmitted on the PROFIBUS. This structure depends on the modules that are actually configured or the channels that are activated.
The DTM provides the CFG-String within the attribute [busMasterConfigurationPart](#) that is part of the XML document available via [IDtmParameter::GetParameters\(\)](#). The structure of the [busMasterConfigurationPart](#) is defined according to the PROFIBUS-DP-Slave-Bus-Parameter-Set (see [chapter 15.5](#). and also PROFIBUS Specification).

The master device uses this information to set up communication with the slave device.
- Channels
In case of PROFIBUS, an FDT channel is a representative for a single data or a process value that can be accessed from a frame-application via the master device. The XML document available at [IFdtChannel](#) describes how to access a channel via a PROFIBUS DPV1 command or how to address a channel within a PROFIBUS DP frame for cyclic I/O. Besides address information according to the PROFIBUS Specification. The XML document contains an id and name for the channel and information like data type, related status signals etc.

In a DPV0 environment, depending on the situation, the underlying master device may have either Master Class 1 functionality or Master Class 2 functionality. A Class 1 master can write output data to a device and control data exchange, where a Class 2 master can only read the output data. Generally it is assumed that parametrization as described here is performed as a master Class 2 station.

Parameterization:

There are two options to write parameters set from the DTM's GUI to the PROFIBUS slave device in the field:

- User Parameters
User Parameters are part of the PROFIBUS-DP-Slave-Bus-Parameter-Set. They contain manufacturer-specific data to characterize the DP-Slave. The DTM writes the User Parameters to the [busMasterConfigurationPart](#). The User Parameters are stored with the master device during PROFIBUS master configuration and are automatically sent to the slave during set up of bus communication. (This is PROFIBUS-specific; for details, see PROFIBUS Specification.) When changing User Parameters on runtime,

the DTM must use a DP-V0 connection and the appropriate DP-V0 commands for parameter exchange as described in the XML schemas.

- Writing Parameters with DP-V1 services (MSAC2 services)

The DTM may use DP-V1 transport services to send its parameters to the slave device. For that, it has to use a DP-V1 connection and the corresponding communication commands. During set up of communication, DP-V1 services are not sent automatically. The frame-application or a DTM must invoke a download of parameters via DP-V1.

For details on the different behavior of slaves depending on the kind of parameterization, refer to the PROFIBUS Specification.

DP-V1 connections and communication commands can also be used to execute commands at the slave. For details on the use of DP-V1, see also PROFIBUS Specification.

15.1 Communication Schema

Used at: [IFdtCommunication::ConnectRequest\(\)](#)
[IFdtCommunication::OnConnectResponse\(\)](#)
[IFdtCommunication::DisconnectRequest\(\)](#)
[IFdtCommunication::OnDisconnectResponse\(\)](#)
[IFdtCommunication::TransactionRequest\(\)](#)
[IFdtCommunication::OnTransactionResponse\(\)](#)

15.1.1 DPV0 Communication

The XML document contains the address information and the communication data.

Attribute	Description
busAddress	Address information according to the Profibus specification (see also DTMParameterSchema , attribute busAddress)
connectStatus	Describes the connection status established by the communication component. The status "masterConnectedOnly" means that the communication component has established a connection to the Profibus master device and will accept an online access to the user parameters, independent whether the device is available or not. The Status "deviceAtLifeList" means that the communication component has established a connection to the Profibus master device and has checked that the device is in the life list of the master stack. In this state the master will accept an online access to the user parameters and will send the user parameter to the device, independent whether the device is in data-exchange or not. The status "deviceInDataExchange" means that the communication component has established a connection to the Profibus master device and has checked that the device is in data-exchange. In this state the master will accept an online access to the user parameters and will send the user parameter to the device, so that the new data will directly influence the process
errorCode	Status information according to the Profibus specification. For description of error code see: DIN 19245 Part 3, PROFIBUS (P. 40ff., 83ff., 39)
communicationReference	Mandatory identifier for a communication link to a device This identifier is allocated by the communication component during the connect. The address information has to be used for all following

Attribute	Description
	communication calls
delayTime	Delay time in [ms] between two communication calls
sequenceTime	Period of time in [ms] for the whole sequence

Tag	Description
Abort	Describes the abort
ConnectRequest	Describes the communication request to establish a connection to a Profibus master device
ConnectResponse	Describes the communication response to the connect request and provides the information how the following WriteUserParameter commands will be send to the device (see connectStatus)
DisconnectRequest	Describes the communication request to release a connection to a Profibus master device
DisconnectResponse	Describes the communication response
FDT	Root tag
ReadUserParameterRequest	Describes the communication request according to the Profibus DPV0 specification
ReadUserParameterResponse	Describes the communication response, read from the Profibus master device, according to the Profibus DPV0 specification
WriteUserParameterRequest	Describes the communication request according to the Profibus DPV0 specification (see setPrm). The user parameter will be send according to the established connection (see connectStatus)
WriteUserParameterResponse	Describes the communication response according to the Profibus DPV0 specification
ReadOutputDataRequest	Describes the communication request according to the Profibus DPV0 specification. Depending on whether the underlying FDTChannel is an Master Cl. 1 or a Master Cl. 2 this service will be local or result in an read access to the slave device. It is possible to specify which data will be read by providing an fdt:ChannelReference.
ReadOutputDataResponse	Describes the communication response according to the Profibus DPV0 specification
SequenceBegin	Describes the sequence begin
SequenceEnd	Describes the sequence end
SequenceStart	Describes the sequence start
WriteOutputDataRequest	Describes the communication request according to the Profibus DPV0 specification. The output data will be send according to the established connection. It is necessary to specify which data will be read by providing an fdt:ChannelReference. If the underlying FDTChannel is provided as Master Cl. 2, writing Output Data will be not possible. According to this, the response will provide an communication error.
WriteOutputDataResponse	Describes the communication response according to the Profibus DPV0 specification
ReadInputDataRequest	Describes the communication request according to the Profibus DPV0 specification. Depending on whether the underlying FDTChannel is an Master Cl. 1 or a Master Cl. 2 this service will be local or result in a read access to the slave device. It is possible to specify which data is read by providing an fdt:ChannelReference.

Tag	Description
ReadInputDataResponse	Describes the communication response according to the Profibus DPV0 specification
ReadDiagnosisDataRequest	Describes the communication request according to the Profibus DPV0 specification
ReadDiagnosisDataResponse	Describes the communication response according to the Profibus DPV0 specification

```

<Schema name="FDTProfibusDPV0CommunicationSchema" 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="ui1"/>
  <AttributeType name="errorCode" dt:type="bin.hex"/>
  <AttributeType name="communicationReference" dt:type="uuid"/>
  <AttributeType name="connectStatus" dt:type="enumeration" dt:values="masterConnectedOnly
deviceAtLifeList deviceInDataExchange"/>
  <AttributeType name="sequenceTime" dt:type="ui4"/>
  <AttributeType name="delayTime" dt:type="ui4"/>
  <!--Definition of Elements-->
  <ElementType name="ConnectRequest" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="busAddress" required="yes"/>
  </ElementType>
  <ElementType name="ConnectResponse" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="busAddress" required="yes"/>
    <attribute type="communicationReference" required="yes"/>
    <attribute type="connectStatus" required="yes"/>
  </ElementType>
  <ElementType name="DisconnectRequest" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="busAddress" required="yes"/>
    <attribute type="communicationReference" required="yes"/>
  </ElementType>
  <ElementType name="DisconnectResponse" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="busAddress" required="yes"/>
    <attribute type="communicationReference" required="yes"/>
  </ElementType>
  <ElementType name="ReadUserParameterRequest" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="communicationReference" required="yes"/>
  </ElementType>
  <ElementType name="ReadUserParameterResponse" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="communicationReference" required="yes"/>
    <attribute type="errorCode" required="yes"/>
    <element type="fdt:CommunicationData" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="WriteUserParameterRequest" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="communicationReference" required="yes"/>
    <element type="fdt:CommunicationData" minOccurs="1" maxOccurs="1"/>
  </ElementType>
  <ElementType name="WriteUserParameterResponse" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="communicationReference" required="yes"/>
    <attribute type="errorCode" required="yes"/>
  </ElementType>
  <ElementType name="ReadOutputDataRequest" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="communicationReference" required="yes"/>
    <element type="fdt:ChannelReference" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="ReadOutputDataResponse" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="communicationReference" required="yes"/>
    <attribute type="errorCode" required="yes"/>
    <element type="fdt:CommunicationData" minOccurs="0" maxOccurs="1"/>
  </ElementType>

```



```

</ElementType>
<ElementType name="WriteOutputDataRequest" content="eltOnly" model="closed">
  <attribute type="fdt:nodetd" required="no"/>
  <attribute type="communicationReference" required="yes"/>
  <element type="fdt:ChannelReference" minOccurs="1" maxOccurs="1"/>
  <element type="fdt:CommunicationData" minOccurs="1" maxOccurs="1"/>
</ElementType>
<ElementType name="WriteOutputDataResponse" content="empty" model="closed">
  <attribute type="fdt:nodetd" required="no"/>
  <attribute type="communicationReference" required="yes"/>
  <attribute type="errorCode" required="yes"/>
</ElementType>
<ElementType name="ReadInputDataRequest" content="eltOnly" model="closed">
  <attribute type="fdt:nodetd" required="no"/>
  <attribute type="communicationReference" required="yes"/>
  <element type="fdt:ChannelReference" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="ReadInputDataResponse" content="eltOnly" model="closed">
  <attribute type="fdt:nodetd" required="no"/>
  <attribute type="communicationReference" required="yes"/>
  <attribute type="errorCode" required="yes"/>
  <element type="fdt:CommunicationData" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="ReadDiagnosisDataRequest" content="empty" model="closed">
  <attribute type="fdt:nodetd" required="no"/>
  <attribute type="communicationReference" required="yes"/>
</ElementType>
<ElementType name="ReadDiagnosisDataResponse" content="eltOnly" model="closed">
  <attribute type="fdt:nodetd" required="no"/>
  <attribute type="communicationReference" required="yes"/>
  <attribute type="errorCode" required="yes"/>
  <element type="fdt:CommunicationData" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="SequenceBegin" content="empty" model="closed">
  <attribute type="sequenceTime" required="no"/>
  <attribute type="delayTime" required="no"/>
  <attribute type="communicationReference" required="yes"/>
</ElementType>
<ElementType name="SequenceEnd" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
</ElementType>
<ElementType name="SequenceStart" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
</ElementType>
<ElementType name="Abort" content="empty" model="closed"/>
<ElementType name="FDT" content="eltOnly" model="closed">
  <attribute type="fdt:nodetd" required="no"/>
  <group order="one">
    <element type="ConnectRequest"/>
    <element type="ConnectResponse"/>
    <element type="DisconnectRequest"/>
    <element type="DisconnectResponse"/>
    <element type="ReadUserParameterRequest"/>
    <element type="ReadUserParameterResponse"/>
    <element type="WriteUserParameterRequest"/>
    <element type="WriteUserParameterResponse"/>
    <element type="ReadOutputDataRequest"/>
    <element type="ReadOutputDataResponse"/>
    <element type="WriteOutputDataRequest"/>
    <element type="WriteOutputDataResponse"/>
    <element type="ReadInputDataRequest"/>
    <element type="ReadInputDataResponse"/>
    <element type="ReadDiagnosisDataRequest"/>
    <element type="ReadDiagnosisDataResponse"/>
    <element type="SequenceBegin"/>
    <element type="SequenceEnd"/>
    <element type="SequenceStart"/>
    <element type="Abort"/>
    <element type="fdt:CommunicationError"/>
  </group>
</ElementType>
</Schema>

```

15.1.2 DPV1 Communication

The XML document contains the address information and the communication data.

Attribute	Description
api	Address information according to the Profibus specification
busAddress	Address information according to the Profibus specification (see also DTMParameterSchema , attribute busAddress)
errorCode	Status information according to the Profibus specification. For description of error code see: PROFIBUS Nutzerorganisation e.V., PROFIBUS Guideline, Order-Nr. 2.0082, Technical Guideline, PROFIBUS – DP Extensions to EN 50170 (DPV1), Version 2.0, April 1998. (P.. 120-121); . For description of abort information see: DIN 19245 Part 3, PROFIBUS, (P..54ff., 11)
index	Address information according to the Profibus specification
communicationReference	Mandatory identifier for a communication link to a device This identifier is allocated by the communication component during the connect. The address information has to be used for all following communication calls
delayTime	Delay time in [ms] between two communication calls
sequenceTime	Period of time in [ms] for the whole sequence
slot	Address information according to the Profibus specification

Tag	Description
Abort	Describes the abort
ConnectRequest	Describes the communication request
ConnectResponse	Describes the communication response
DisconnectRequest	Describes the communication request
DisconnectResponse	Describes the communication response
FDT	Root tag
ReadRequest	Describes the communication request
ReadResponse	Describes the communication response
SequenceBegin	Describes the sequence begin
SequenceEnd	Describes the sequence end
SequenceStart	Describes the sequence start
WriteRequest	Describes the communication request
WriteResponse	Describes the communication response

```
<Schema name="FDTProfibusDPV1CommunicationSchema" 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="api" dt:type="ui1"/>
  <AttributeType name="busAddress" dt:type="ui1"/>
  <AttributeType name="errorCode" dt:type="bin.hex"/>
  <AttributeType name="index" dt:type="ui1"/>
  <AttributeType name="communicationReference" dt:type="uuid"/>
  <AttributeType name="slot" dt:type="ui1"/>
  <AttributeType name="sequenceTime" dt:type="ui4"/>
  <AttributeType name="delayTime" dt:type="ui4"/>
  <!--Definition of Elements-->
```

```

<ElementType name="ConnectRequest" content="empty" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <attribute type="api" required="yes"/>
  <attribute type="busAddress" required="yes"/>
</ElementType>
<ElementType name="ConnectResponse" content="empty" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <attribute type="api" required="yes"/>
  <attribute type="busAddress" required="yes"/>
  <attribute type="communicationReference" required="yes"/>
  <attribute type="errorCode" required="yes"/>
</ElementType>
<ElementType name="DisconnectRequest" content="empty" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <attribute type="api" required="yes"/>
  <attribute type="busAddress" required="yes"/>
  <attribute type="communicationReference" required="yes"/>
</ElementType>
<ElementType name="DisconnectResponse" content="empty" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <attribute type="api" required="yes"/>
  <attribute type="busAddress" required="yes"/>
  <attribute type="communicationReference" required="yes"/>
  <attribute type="errorCode" required="yes"/>
</ElementType>
<ElementType name="ReadRequest" content="empty" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <attribute type="slot" required="yes"/>
  <attribute type="index" required="yes"/>
  <attribute type="communicationReference" required="yes"/>
</ElementType>
<ElementType name="ReadResponse" content="eltOnly" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <attribute type="slot" required="yes"/>
  <attribute type="index" required="yes"/>
  <attribute type="communicationReference" required="yes"/>
  <attribute type="errorCode" required="yes"/>
  <element type="fdt:CommunicationData" minOccurs="1" maxOccurs="1"/>
</ElementType>
<ElementType name="WriteRequest" content="eltOnly" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <attribute type="slot" required="yes"/>
  <attribute type="index" required="yes"/>
  <attribute type="communicationReference" required="yes"/>
  <element type="fdt:CommunicationData" minOccurs="1" maxOccurs="1"/>
</ElementType>
<ElementType name="WriteResponse" content="empty" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <attribute type="slot" required="yes"/>
  <attribute type="index" required="yes"/>
  <attribute type="communicationReference" required="yes"/>
  <attribute type="errorCode" required="yes"/>
</ElementType>
<ElementType name="SequenceBegin" content="empty" model="closed">
  <attribute type="sequenceTime" required="no"/>
  <attribute type="delayTime" required="no"/>
  <attribute type="communicationReference" required="yes"/>
</ElementType>
<ElementType name="SequenceEnd" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
</ElementType>
<ElementType name="SequenceStart" content="empty" model="closed">
  <attribute type="communicationReference" required="yes"/>
</ElementType>
<ElementType name="Abort" content="empty" model="closed"/>
<ElementType name="FDT" content="eltOnly" model="closed">
  <attribute type="fdt:nodeld" required="no"/>
  <group order="one">
    <element type="ConnectRequest"/>
    <element type="ConnectResponse"/>
    <element type="DisconnectRequest"/>
    <element type="DisconnectResponse"/>
    <element type="ReadRequest"/>

```

```

        <element type="ReadResponse"/>
        <element type="WriteRequest"/>
        <element type="WriteResponse"/>
        <element type="SequenceBegin"/>
        <element type="SequenceEnd"/>
        <element type="SequenceStart"/>
        <element type="Abort"/>
        <element type="fdt:CommunicationError"/>
    </group>
</ElementType>
</Schema>

```

Example:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTProfibusDPV1CommunicationSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
    <ReadResponse slot="1" index="1" communicationReference="13966300-d860-4d18-8bc3-f11f8c9d7597"
    errorCode="0000000000">
        <fdt:CommunicationData byteArray="FF01"/>
    </ReadResponse>
</FDT>

```

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTProfibusDPV1CommunicationSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
    <ReadResponse slot="1" index="1" communicationReference="6B29FC40-CA47-1067-B31D-
00DD010662DA" errorCode="0">
        <fdt:CommunicationData byteArray="FF01"/>
    </ReadResponse>
</FDT>

```

15.2 Channel Schema

Used at: [IFdtChannel::GetChannelParameters\(\)](#)

The XML document describes a how to access a channel via a Profibus DPV1 command or how to adress a channel within a Profibus DP frame for cyclic I/O.

	Description
api	Address information according to the Profibus specification for channels accessible via Profibus DPV1
bitLength	Additional data type information especially for fieldbus specific data types like 12 bit integer
bitPosition	Address information according to the Profibus specification for channels accessible via Profibus DP
frameApplicationTag	Frame application specific tag used for identification and navigation. The DTM should display this tag at channel specific user interfaces
gatewayBusCategory	Unique identifier for a supported bus type like Profibus or HART according to the FDT specific CATID
invalidBit	Bit position of the invalid status channel accessible via Profibus DP
index	Address information according to the Profibus specification for channels accessible via Profibus DPV1
logic	Additional data type information: positive 0=FALSE 1=TRUE
number	Address information for diagnosis according to the Profibus specification for channels accessible via Profibus DP
protectedByChannelAssignment	TRUE if the channels is set to read only by the frame-application. Usually set to TRUE if a channel assignment exists
simulationBit	Bit position of the simulation status channel accessible via Profibus DP
substituteValueBit	Bit position of the substitute status channel accessible via Profibus DP
statusChannel	TRUE if the channel is for status information only
slotNumber	Address information according to the Profibus specification for channels accessible via Profibus DPV1

Tag	Description
DpAddress	Address information according to the Profibus specification for channels accessible via Profibus DP
DpV1Address	Address information according to the Profibus specification for channels accessible via Profibus DPV1
FDT	Root tag
FDTChannel	Description of the channel
FDTChannelType	Description of the channel component in case of channels with gateway functionality
StatusInformation	Description of additional status information for channels accessible via Profibus DP

```

<Schema name="FDTProfibusChannelParameterSchema" 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="api" dt:type="ui1"/>
  <AttributeType name="bitLength" dt:type="ui4"/>
  <AttributeType name="bitPosition" dt:type="ui4"/>
  <AttributeType name="frameApplicationTag" dt:type="string"/>
  <AttributeType name="gatewayBusCategory" dt:type="uuid"/>
  <AttributeType name="invalidBit" dt:type="ui4"/>
  <AttributeType name="index" dt:type="ui1"/>
  <AttributeType name="logic" dt:type="enumeration" dt:values="positive negative"/>
  <AttributeType name="number" dt:type="ui4"/>
  <AttributeType name="protectedByChannelAssignment" dt:type="boolean"/>
  <AttributeType name="simulationBit" dt:type="ui4"/>
  <AttributeType name="statusChannel" dt:type="boolean"/>
  <AttributeType name="substituteValueBit" dt:type="ui4"/>
  <AttributeType name="slotNumber" dt:type="ui1"/>
  <!--Definition of Elements-->
  <ElementType name="DpAddress" content="empty" model="closed">
    <attribute type="fdt:nodel" required="no"/>
    <attribute type="bitPosition" required="yes"/>
    <attribute type="bitLength" required="yes"/>
  </ElementType>
  <ElementType name="DpV1Address" content="empty" model="closed">
    <attribute type="fdt:nodel" required="no"/>
    <attribute type="api" required="yes"/>
    <attribute type="slotNumber" required="yes"/>
    <attribute type="index" required="yes"/>
  </ElementType>
  <ElementType name="StatusInformation" content="empty" model="closed">
    <attribute type="fdt:nodel" required="no"/>
    <attribute type="logic" required="yes"/>
    <attribute type="invalidBit" required="no"/>
    <attribute type="simulationBit" required="no"/>
    <attribute type="substituteValueBit" required="no"/>
  </ElementType>
  <ElementType name="FDTChannel" content="eltOnly" model="closed" order="seq">
    <attribute type="fdt:nodel" required="no"/>
    <attribute type="fdt:tag" required="yes"/>
    <attribute type="fdt:id" required="yes"/>
    <attribute type="protectedByChannelAssignment" required="yes"/>
    <attribute type="number" required="yes"/>
    <attribute type="fdt:dataType" required="yes"/>
    <attribute type="fdt:signalType" required="yes"/>
    <attribute type="frameApplicationTag" required="no"/>
    <element type="fdt:BitEnumeratorEntries" minOccurs="0" maxOccurs="1"/>
    <element type="fdt:EnumeratorEntries" minOccurs="0" maxOccurs="1"/>
    <element type="fdt:Unit" minOccurs="0" maxOccurs="1"/>
    <element type="DpAddress" minOccurs="0" maxOccurs="1"/>
    <element type="DpV1Address" minOccurs="0" maxOccurs="1"/>
    <element type="StatusInformation" minOccurs="0" maxOccurs="1"/>
    <element type="fdt:Alarms" minOccurs="0" maxOccurs="1"/>
    <element type="fdt:Ranges" minOccurs="0" maxOccurs="1"/>
    <element type="fdt:SubstituteValue" minOccurs="0" maxOccurs="1"/>
  </ElementType>
  <ElementType name="FDTChannelType" content="eltOnly" model="closed">
    <attribute type="fdt:nodel" required="no"/>
    <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1"/>
    <attribute type="gatewayBusCategory" required="no"/>
    <attribute type="statusChannel" required="no"/>
  </ElementType>
  <ElementType name="FDT" content="eltOnly" model="closed">
    <attribute type="fdt:nodel" required="no"/>
    <element type="FDTChannelType" minOccurs="1" maxOccurs="1"/>
    <element type="FDTChannel" minOccurs="1" maxOccurs="1"/>
  </ElementType>
</Schema>

```

Example:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTProfibusChannelParameterSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
  <FDTChannelType>
    <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
  </FDTChannelType>
  <FDTChannel fdt:tag="myTag" fdt:id="PV" protectedByChannelAssignment="0" number="123"
fdt:dataType="float" fdt:signalType="output">
    <DpAddress bitPosition="17" bitLength="32"/>
    <DpV1Address api="1" slotNumber="1" index="1"/>
    <StatusInformation logic="positive"/>
    <fdt:Alarms>
      <fdt:Alarm alarmType="lowAlarm">
        <fdt:StaticValue staticValue="25"/>
      </fdt:Alarm>
      <fdt:Alarm alarmType="highAlarm">
        <fdt:StaticValue staticValue="100"/>
      </fdt:Alarm>
    </fdt:Alarms>
    <fdt:Ranges>
      <fdt:Range>
        <fdt:LowerRange>
          <fdt:ChannelReference idref="PV_LOWER_RANGE_VALUE"/>
        </fdt:LowerRange>
        <fdt:UpperRange>
          <fdt:ChannelReference idref="PV_UPPER_RANGE_VALUE"/>
        </fdt:UpperRange>
        <fdt:Unit>
          <fdt:ChannelReference idref="PV_RANGE_VALUES_UNITS_CODE"/>
        </fdt:Unit>
      </fdt:Range>
    </fdt:Ranges>
  </FDTChannel>
</FDT>

```

15.3 Topology Scan Schema

Used at: IDtmEvents::OnScanResponse()

The XML document describes one entry in the list of scanned PROFIBUS-Devices.

Tag	Description
ProfibusDevice	Specifies a Profibus device

```
schema:FDTDataTypesSchema.xml" xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <!--Definition of Attributes-->
  <AttributeType name="busAddress" dt:type="ui1"/>
  <!--Definition of Elements-->
  <ElementType name="ProfibusDevice" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="busAddress" required="yes"/>
    <attribute type="fdt:deviceTypeId" required="yes"/>
    <attribute type="fdt:subDeviceType" required="no"/>
  </ElementType>
</Schema>
```


15.4 Master-Bus Parameter Set (DP)

The following parameter set represent the content of the attribute [busMasterConfigurationPart](#) within the [DTMParameterSchema](#) for Profibus master device. This attribute has to be set for each Profibus master device according to the Profibus specification. For further details please refer to the sequence chart "Configuration of a Fieldbus Master" and the Profibus specification.

Name	Type	Comment
Bus_Para_Len	Unsigned16	Length of the bus parameter set inclusive the length parameter (range 34 to $2^{16}-1$)
FDL_Add	Unsigned8	Mandatory part of the Bus Parameter Set according to PROFIBUS-DP Specification; not used in this context as station address is transferred in a separate variable; FDL-Add may differ from real station address used by EE and DTMs
Baud_rate	Unsigned8	Code number for the baud rate
T _{SL}	Unsigned16	Slot Time
min T _{SDR}	Unsigned16	Min. station delay response
max T _{SDR}	Unsigned16	Max. station delay response
T _{QUI}	Unsigned8	Quit-time
T _{SET}	Unsigned8	Setup-time
T _{TR}	Unsigned32	Target rotation time
G	Unsigned8	GAP Update Factor
HSA	Unsigned8	Highest station address
max_retry_limit	Unsigned8	Max. retry limit
Bp_Flag	Unsigned8	Flags for the user interface, e.g. error action flag
Min_Slave_Interval	Unsigned16	Smallest allowed time period between two slave poll cycles
Poll_Timeout	Unsigned16	Master-master timeout
Data_Control_Time	Unsigned16	Guaranteed time period between two Data_transfer_list updates
Octet 1 (reserved)	Octet-String	
....		
Octet 6 (reserved)		
Master_User_Data_Len	Unsigned16	Length of Master_User_Data inclusive lengthparameter
Master_Class2_Name	Visible-String (32)	Name of DP class 2 master the parameter set was created with
Master_User_Data	Octet-String	Manufacturer specific

Table 1. (Master-) Bus Parameter Set

15.5 Slave Bus Parameter Set (DP)

The following parameter set represent the content of the attribute [busMasterConfigurationPart](#) within the [DTMParameterSchema](#) for Profibus slave devices. This attribute has to be set for each Profibus slave device according to the Profibus specification. For further details please refer to the sequence chart "Configuration of a Fieldbus Master" and the Profibus specification.

Name	Type	Comment
Slave_Para_Len	Unsigned16	Length of the slave parameter set inclusive the length parameter
Sl_Flag	Unsigned8	Slave specific flags like New_Prm, Active, Fail_Safe,
Slave_Type	Unsigned8	Manufacturer specific slave type denotation (0 by default for DP-Slaves)
Octet 1 (reserved)	Octet-String	
....		
Octet 12 (reserved)		
Prm_Data_Len	Unsigned16	Length of Prm_Data inclusive the length parameter (range 9 to 246)
Prm_Data	Octet-String	
Cfg_Data_Len	Unsigned16	Length of Cfg_Data inclusive the length parameter (range 3 to 246)
Cfg_Data	Octet-String	
Add_Tab_Len	Unsigned16	Length of Add_Tab inclusive the length parameter (range 2 to $2^{16}-31$)
Add_Tab	Octet-String	Address assignment table
Slave_User_Data_Len	Unsigned16	Length of Slave_User_Data inclusive the length parameter (range 2 to $2^{16}-31$)
Slave_User_Data	Octet-String	Manufacturer specific data to characterize the DP-Slave for the master

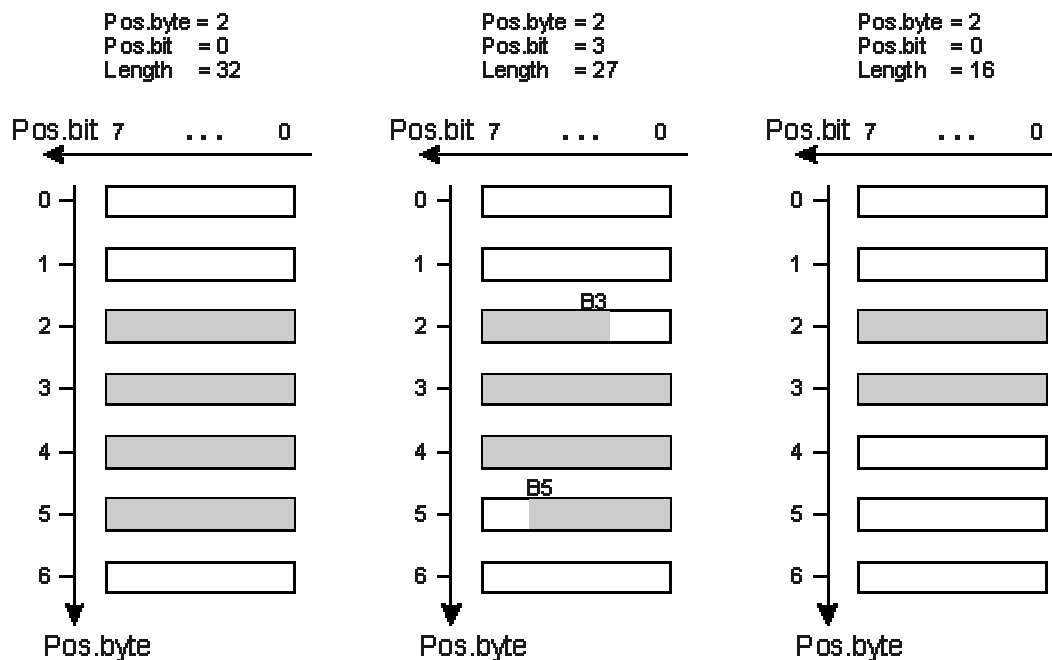
Table 2. Parameter Set of DP-Slaves

15.6 Module and Channel Data

A slave's current module configuration including the belonging channels has to be available by FdtChannel objects. That is required as within the environment process variables have to be assigned to single channels and the slave configuration has to be displayed in the environment's system overview without using the DTM's user interface.

The addressing of channels within the PROFIBUS data frame is bit oriented and data type independent. A bit position and a bit length determine each channel. The data type of a parameter determines how to convert the bitfield.

Channels within PROFIBUS data frames:



Representation in FDT Notation:

bitPosition="16"
bitLength="32"

bitPosition="19"
bitLength="27"

bitPosition="16"
bitLength="16"

The following example shows the expected structure description of a modular slave in a DTM parameter set using FDT-Parameters with the object types defined in the FDT-Specification.

Representation of the data frame using GSD information:

```
BeginSlave;;
RIOLB8101.GSD;6;
BeginModules;;
1X03 FrequencyCount;0x51;
Empty ;0x00;
Empty ;0x00;
Empty;0x00;
Empty;0x00;
Empty;0x00;
Empty;0x00;
2XXX ValveBlock.;0x30;
EndModules;;
EndSlave;;
```

Description of the position of single signal channels within the data frame:

# Frame description Remote I/O											
Vendor:	"RIO manufacturer"										
Device type:	"LB 8101"	Device identifier:				0x8101	# hex				
Frame identifier:	"B1t1"										
# Input data											
# Channel	# Channel type	# Data	# Data	# Data	# Invalid Bit	# Invalid Bit	# Subst. Value Bit	# Subst. Value Bit	# Sim. Bit	# Sim. Bit	# Status-Channel
		# PoS. byte	# PoS. bit	# Length	# PoS. byte	# Pos. bit	# Pos. byte	# Pos. bit	# PoS. byte	# Pos. bit	
Module:	"EP01" "Digital Input (Counter) LB/FB 1X03 (4 Byte)"										
"Count 1_0"	READ INT	0	0	32	-/-	-/-	-/-	-/-	-/-	-/-	0
Module:	"EP08" "Valve block LB/FB 2XXX / 1 DO / 2 DI (1 Byte Input u. Output)"										
"DI_8_LF OUT0"	READ BOOL	4	1	1	-/-	-/-	-/-	-/-/-	-/-1		
"DI_8_LF1"	READ BOOL	4	3	1	-/-	-/-	-/-	-/-/-	-/-1		
"DI_8_LF2"	READ BOOL	4	5	1	-/-	-/-	-/-	-/-	-/-	-/-	1
# Output data											
# Channel	# Channel type	# Data	# Data	# Data	# Invalid Bit	# Invalid Bit	# Subst. Value Bit	# Subst. Value Bit	# Sim. Bit	# Sim. Bit	# Status-Channel
		# PoS. byte	# PoS. bit	# Length	# PoS. byte"#"o". b"t"#"PoS. byte		# PoS. byte	# Pos. bit			
Modul:	"EP08" "Valve block LB/FB 2XXX / 1 DO / 2 DI (1 Byte Input u. Output)"										
"DO_8_0"	WRITE BOOL	0	0	1	4	1	-/-	-/-	-/-	-/-	0

Corresponding XML documents:**DTM Parameters:**

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistant" fdt:dataSetState="default">
  <fdt:DtmDeviceType>
    <fdt:VersionInformation name="LB 8101" vendor="RIO Manufacturer" version="1.0" date="2000-
08"05"/>
    <fdt:SupportedLanguages>
      <fdt:LanguageId languageid="1"/>
    </fdt:SupportedLanguages>
    <fdt:BusCategories>
      <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E098727"B9"/>
      <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E098727"B9"/>
    </fdt:BusCategories>
  </fdt:DtmDeviceType>
  <DtmDevice fdt:tag="00PGH10EC"01">
    <InternalTopology>

```

```

    <InternalChannel>
      <Module moduleid="1" slot="1">
        <fdt:VersionInformation name="LB/FB 1"03" vendor="RIO Manufacturer" versio="10"
date="2000-08"05" descriptor="Digital Input (Counter 4 Byte)">
          <fdt:ChannelReferences>
            <fdt:ChannelReference idref="Count_1_0"/>
          </fdt:ChannelReferences>
        </Module>
      </InternalChannel>
    <InternalChannel>
      <Module moduleid="8" slot="8">
        <fdt:VersionInformation name="1DO / 2DI" vendor="Vendor name" version=""0"
date="2000-08"05" descriptor="Valve Block LB/FB 2XXX 1Byte Input Output">
          <fdt:ChannelReferences>
            <fdt:ChannelReference idref="DI_8LFO"0"/>
            <fdt:ChannelReference idref="DI_1_1"/>
            <fdt:ChannelReference idref="DI_8_F1"/>
            <fdt:ChannelReference idref="DI_2_2"/>
            <fdt:ChannelReference idref="DI_8_F2"/>
            <fdt:ChannelReference idref="DO_0_0"/>
          </fdt:ChannelReferences>
        </Module>
      </InternalChannel>
    </InternalTopology>
  </DtmDevice>
</FDT>

```

FDT Channel "Count_1_0":

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTProfibusChannelParameterSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
  <FDTChannelType>
    <fdt:VersionInformation name="Digital Input (Counter 4 Byte)" vendor="RIO Manufacturer"
version="1.0" date="2000-08"05"/>
  </FDTChannelType>
  <FDTChannel fdt:tag="00PGH10EB001" fdt:id="Count_0_1" protectedByChannelAssignment="0"
number="0" fdt:dataType="int" fdt:signalType="input">
    <DpAddress bitPosition="0" bitLength="32"/>
  </FDTChannel>
</FDT>

```

FDT Channel "DI_8_LFOUT0":

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTProfibusChannelParameterSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
  <FDTChannelType statusChannel="1">
    <fdt:VersionInformation name="Digital Input Valve Block" vendor="RIO Manufacturer" version="1.0"
date="2000-08-05"/>
  </FDTChannelType>
  <FDTChannel fdt:tag="00PGH10FB001" fdt:id="DI_8LFOUT0" protectedByChannelAssignment="0"
number="10" fdt:dataType="binary" fdt:signalType="input">
    <DpAddress bitPosition="33" bitLength="1"/>
  </FDTChannel>
</FDT>

```

FDT Channel "DI_8_1":

```

<?xml version="1.0"?>

```

```

<FDT xmlns="x-schema:FDTProfibusChannelParameterSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
  <FDTChannelType>
    <fdt:VersionInformation name="Digital Input Valve Block" vendor="RIO Manufacturer" version="1.0"
date="2000-08-05"/>
  </FDTChannelType>
  <FDTChannel fdt:tag="00PGH10EB002" fdt:id="DI_8_1" protectedByChannelAssignment="0" number="1"
fdt:dataType="binary" fdt:signalType="input">
    <DpAddress bitPosition="34" bitLength="1"/>
    <StatusInformation logic="positive" invalidBit="35"/>
  </FDTChannel>
</FDT>

```

FDT Channel "DI_8_LF1":

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTProfibusChannelParameterSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
  <FDTChannelType statusChannel="1">
    <fdt:VersionInformation name="Digital Input Valve Block" vendor="RIO Manufacturer" version="1.0"
date="2000-08-05"/>
  </FDTChannelType>
  <FDTChannel fdt:tag="00PGH10FB002" fdt:id="DI_8_LF1" protectedByChannelAssignment="0"
number="11" fdt:dataType="binary" fdt:signalType="input">
    <DpAddress bitPosition="35" bitLength="1"/>
  </FDTChannel>
</FDT>

```

FDT Channel "DI_8_2":

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTProfibusChannelParameterSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
  <FDTChannelType>
    <fdt:VersionInformation name="Digital Input Valve Block" vendor="RIO Manufacturer" version="1.0"
date="2000-08-05"/>
  </FDTChannelType>
  <FDTChannel fdt:tag="00PGH10EB003" fdt:id="DI_8_2" protectedByChannelAssignment="0" number="2"
fdt:dataType="binary" fdt:signalType="input">
    <DpAddress bitPosition="36" bitLength="1"/>
    <StatusInformation logic="positive" invalidBit="37"/>
  </FDTChannel>
</FDT>

```

FDT Channel "DI_8_LF2":

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTProfibusChannelParameterSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
  <FDTChannelType statusChannel="1">
    <fdt:VersionInformation name="Digital Input Valve Block" vendor="RIO Manufacturer" version="1.0"
date="2000-08-05"/>
  </FDTChannelType>
  <FDTChannel fdt:tag="00PGH10FB003" fdt:id="DI_8_LF2" protectedByChannelAssignment="0"
number="12" fdt:dataType="binary" fdt:signalType="input">
    <DpAddress bitPosition="37" bitLength="1"/>
  </FDTChannel>
</FDT>

```

FDT Channel "DO_8_0":

```

<?xml version="1.0"?>

```

```

<FDT xmlns="x-schema:FDTProfibusChannelParameterSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
  <FDTChannelType>
    <fdt:VersionInformation name="Digital Output Valve Block" vendor="RIO Manufacturer" version="1.0"
date="2000-08-05"/>
  </FDTChannelType>
  <FDTChannel fdt:tag="00PGH10EB004" fdt:id="DO_8_0" protectedByChannelAssignment="0"
number="3" fdt:dataType="binary" fdt:signalType="output">
    <DpAddress bitPosition="0" bitLength="1"/>
    <StatusInformation logic="positive" invalidBit="32"/>
  </FDTChannel>
</FDT>

```

15.7 ProfiSafe

15.7.1 Motivation

The purpose of the ProfiSafe-UseCase paper is to compile a collection of typical existing failsafe automation examples from the discrete and continuous manufacturing areas and to derive a common systematic approach. This proposed solution ought to be based on

- Profibus DPV1 and ProfiSafe communication principles,
- new possibilities of combined failsafe controllers (standard and failsafe programs within one PLC)
- and current activities of other Profibus working groups like FDT/DTM and Device-FB

Within the context of the ProfiSafe-UseCase paper this document covers the partial aspect "FDT programmers interface for i-parameter verification".

15.7.2 General Parameter Handling

In order to communicate in a safe way, each ProfiSafe device (F-Slave) requires so-called F-Parameters for the adjustment of the operational mode of its ProfiSafe driver. Those parameters comprise watch dog time, safety integrity level, container size, etc. In case of simple F-slaves no further parameters are required. F-Parameters will therefore be defined within GSD Revision 4.0.

In contrast, a complex F-slave requires additional individual and safety relevant parameters (i-Parameter) that, due to its size (>240 Bytes), often cannot be transported by the initial parameterization telegram (Prm-Telegram). As an amendment, ProfiSafe guidelines are suggesting an additional method via proxy function blocks in a safety PLC, thus providing additional functionality like program-controlled reparametrization, etc.

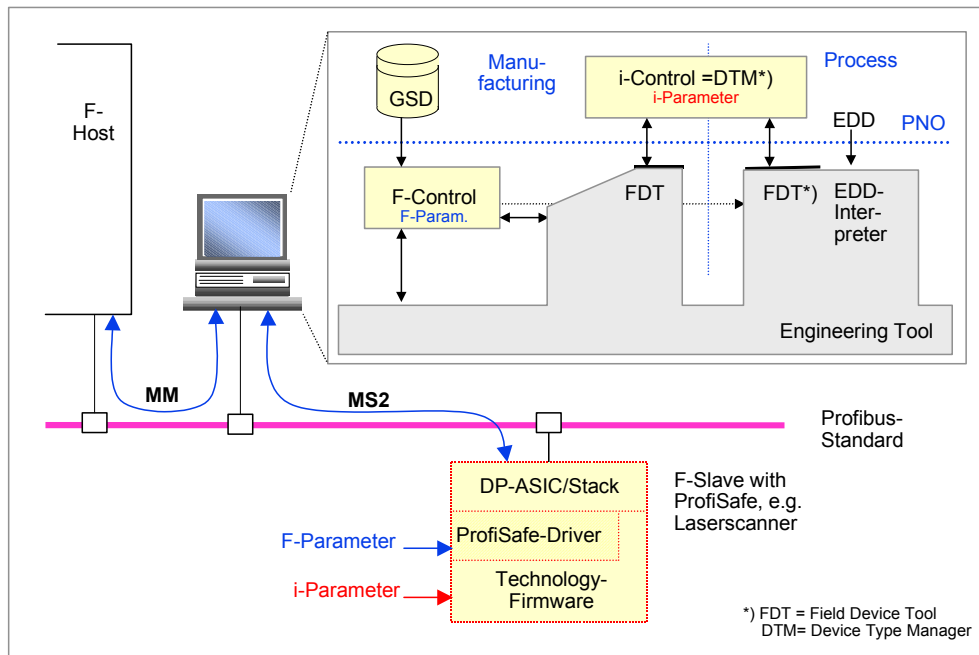


Figure 1-1 F-Parameter and i-Parameter

I-Parameters, being F-device specific by nature, ought to be handled by Device-Type-Manager programs (DTM) coming from the F-device manufacturers. The engineering tool operates as a frame-application for such a DTM of a F-slave. It is routing the communication requests of a DTM to its device (MS2) and provides data persistence. Otherwise it is covering all traditional tasks like network configuration, parameterization, commissioning and diagnosis. It is communicating with the PLC via the Master-Master-Protocol (MM). In this respect a PLC on the Profibus is a device of its own class (Master Class 1). The engineering tool itself is defined to be a "Master Class 2" device.

15.7.3 ProfiSafe i-Parameter

The structure of the ProfiSafe i-parameter set is defined within the ProfiSafe guidelines. This set is wrapped up in XML-format for transportation across the FDT interface.

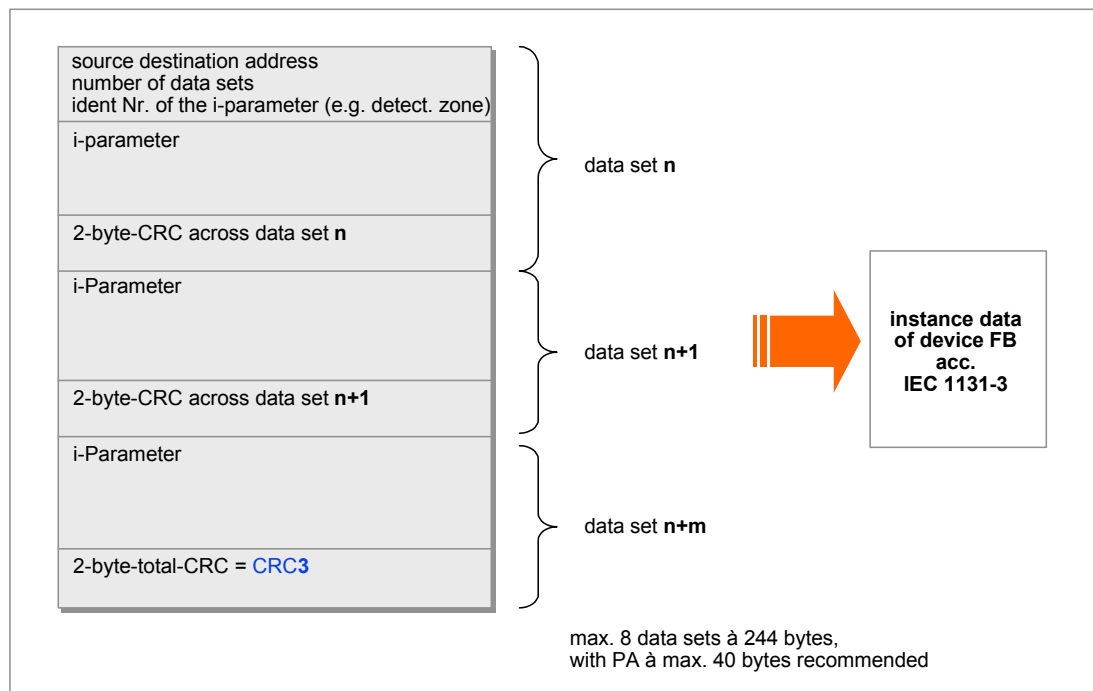


Figure 3-2 Data Structure of ProfiSafe i-Parameters

The data structures and the data formats of the inner i-Parameters are proprietary. The whole set is divided into parts that fit into a Profibus data set ready for transportation. Each data set comprises a 2-bytes CRC for data consistency check. The polynomial is fixed and defined within the ProfiSafe guidelines. The CRC of the last set checks the total block.

15.8 GSD Information

The GSD information is not stored with single slave instances or in a global accessible file. It is provided by the DTM at [IDtmInformation](#). On method [GetInformation\(\)](#), a DTM of a PROFIBUS device provides the GSD information within its XML document.

16 Appendix – HART

16.1 Communication Schema

Used at: [IFdtCommunication::ConnectRequest\(\)](#)
[IFdtCommunication::OnConnectResponse\(\)](#)
[IFdtCommunication::DisconnectRequest\(\)](#)
[IFdtCommunication::OnDisconnectResponse\(\)](#)
[IFdtCommunication::TransactionRequest\(\)](#)
[IFdtCommunication::OnTransactionResponse\(\)](#)

The XML document contains the address information and the communication data.

Attribute	Description
address1	Address information according to the HART specification
address2	Address information according to the HART specification
address3	Address information according to the HART specification
commandNumber	Address information according to the HART specification
communicationReference	Mandatory identifier for a communication link to a device This identifier is allocated by the communication component during the connect. The address information has to be used for all following communication calls
delayTime	Delay time in [ms] between two communication calls
deviceStatus	Status information according to the HART specification
deviceTypeld	Address information according to the HART specification
longFrameRequired	Address information according to the HART specification
manufacturerId	Address information according to the HART specification (Table: VIII, MANUFACTURER IDENTIFICATION CODES)
preambleCount	At the connect request the attribute is optional and contains a hint for the communication component about the number of preambles, required by the device type. At the connect response the attribute is mandatory contains the information about the currently used preambleCount.
primaryMaster	At the connect request the attribute is optional and contains a hint for a communication component that a DTM requires communication as primary or secondary master. At the connect response the attribute is mandatory contains the information about the current state of the master
sequenceTime	Period of time in [ms] for the whole sequence
shortAddress	Address information according to the HART specification. This value is accessible via the attribute slaveAddress defined within the DTMParameterSchema . SlaveAddress is part of the BusInformation structure. These values must be set by the responsible component as described in chapter Nested Communication
value	Variable for status information

Tag	
Abort	Describes the abort
CommandResponse	Status information according to the HART specification
CommunicationStatus	Status information according to the HART specification
ConnectRequest	Describes the communication request
ConnectResponse	Describes the communication response
DataExchangeRequest	Describes the communication request

Tag	Description
DataExchangeResponse	Describes the communication response
DisconnectRequest	Describes the communication request
DisconnectResponse	Describes the communication response
FDT	Root tag
LongAddress	Address information according to the HART specification (only supported by HART devices based on HART revision > 5, see related documentation) In the HART protocol Manufacturer ID and Device type ID are contained in the longaddress. If the channel delivers different values in fdthart:manufacturerId / fdthart:deviceTypeId and in the corresponding bytes in fdthart:LongAddress, the following rule applies: <ul style="list-style-type: none"> * The fdthart:LongAddress has to be used for communication and * The fdthart:manufacturerId and fdthart:deviceTypeId may be used only as information about the manufacturer and the type of device.
SequenceBegin	Describes the sequence begin
SequenceEnd	Describes the sequence end
SequenceStart	Describes the sequence start
ShortAddress	Address information according to the HART specification
Status	Status information according to the HART specification

Remark:

The attribute 'fdt:tag', used within this schema, is accessible via the attribute 'fdt:tag' defined within the [DTMParameterSchema](#). 'fdt:tag' is part of the DtmDevice structure and contains the HART-specific value called TAG, which is used e.g. within command #11, 'READ UNIQUE IDENTIFIER ASSOCIATED WITH TAG'. These value must be set by the responsible component as described in chapter Nested Communication.

```
<Schema name="FDTHARTCommunicationSchema" 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="address1" dt:type="ui1"/>
  <AttributeType name="address2" dt:type="ui1"/>
  <AttributeType name="address3" dt:type="ui1"/>
  <AttributeType name="commandNumber" dt:type="ui1"/>
  <AttributeType name="communicationReference" dt:type="uuid"/>
  <AttributeType name="deviceStatus" dt:type="ui1"/>
  <AttributeType name="deviceTypeId" dt:type="ui1"/>
  <AttributeType name="longFrameRequired" dt:type="boolean"/>
  <AttributeType name="manufacturerId" dt:type="ui1"/>
  <AttributeType name="preambleCount" dt:type="ui1"/>
  <AttributeType name="primaryMaster" dt:type="boolean"/>
  <AttributeType name="shortAddress" dt:type="ui1"/>
  <AttributeType name="value" dt:type="ui1"/>
  <AttributeType name="sequenceTime" dt:type="ui4"/>
  <AttributeType name="delayTime" dt:type="ui4"/>
  <!--Definition of Elements-->
  <ElementType name="CommunicationStatus" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="value" required="yes"/>
  </ElementType>
  <ElementType name="CommandResponse" content="empty" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="value" required="yes"/>
  </ElementType>
  <ElementType name="Status" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <attribute type="deviceStatus" required="yes"/>
    <group order="one" minOccurs="1" maxOccurs="1">
      <element type="CommunicationStatus"/>
      <element type="CommandResponse"/>
    </group>
  </ElementType>
</Schema>
```

```

        </group>
    </ElementType>
    <ElementType name="LongAddress" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="manufacturerId" required="yes"/>
        <attribute type="deviceTypeId" required="yes"/>
        <attribute type="address1" required="yes"/>
        <attribute type="address2" required="yes"/>
        <attribute type="address3" required="yes"/>
    </ElementType>
    <ElementType name="ShortAddress" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="shortAddress" required="yes"/>
    </ElementType>
    <ElementType name="ConnectRequest" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:tag" required="yes"/>
        <attribute type="preambleCount" required="no"/>
        <attribute type="primaryMaster" required="no"/>
        <attribute type="longFrameRequired" required="no"/>
        <element type="LongAddress" minOccurs="0" maxOccurs="1"/>
        <element type="ShortAddress" minOccurs="1" maxOccurs="1"/>
    </ElementType>
    <ElementType name="ConnectResponse" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:tag" required="yes"/>
        <attribute type="preambleCount" required="yes"/>
        <attribute type="primaryMaster" required="yes"/>
        <attribute type="communicationReference" required="yes"/>
        <element type="LongAddress" minOccurs="0" maxOccurs="1"/>
        <element type="ShortAddress" minOccurs="1" maxOccurs="1"/>
    </ElementType>
    <ElementType name="DisconnectRequest" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="communicationReference" required="yes"/>
    </ElementType>
    <ElementType name="DisconnectResponse" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="communicationReference" required="yes"/>
    </ElementType>
    <ElementType name="DataExchangeRequest" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="commandNumber" required="yes"/>
        <attribute type="communicationReference" required="yes"/>
        <element type="fdt:CommunicationData" minOccurs="0" maxOccurs="1"/>
    </ElementType>
    <ElementType name="DataExchangeResponse" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="commandNumber" required="yes"/>
        <attribute type="communicationReference" required="yes"/>
        <element type="fdt:CommunicationData" minOccurs="0" maxOccurs="1"/>
        <element type="Status" minOccurs="1" maxOccurs="1"/>
    </ElementType>
    <ElementType name="SequenceBegin" content="empty" model="closed">
        <attribute type="sequenceTime" required="no"/>
        <attribute type="delayTime" required="no"/>
        <attribute type="communicationReference" required="yes"/>
    </ElementType>
    <ElementType name="SequenceEnd" content="empty" model="closed">
        <attribute type="communicationReference" required="yes"/>
    </ElementType>
    <ElementType name="SequenceStart" content="empty" model="closed">
        <attribute type="communicationReference" required="yes"/>
    </ElementType>
    <ElementType name="Abort" content="empty" model="closed"/>
    <ElementType name="FDT" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <group order="one" minOccurs="1" maxOccurs="1">
            <element type="ConnectRequest"/>
            <element type="ConnectResponse"/>
            <element type="DisconnectRequest"/>
            <element type="DisconnectResponse"/>
            <element type="DataExchangeRequest"/>

```

```

        <element type="DataExchangeResponse"/>
        <element type="SequenceBegin"/>
        <element type="SequenceEnd"/>
        <element type="SequenceStart"/>
        <element type="Abort"/>
        <element type="fdt:CommunicationError"/>
    </group>
</ElementType>
</Schema>

```

Example:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTHARTCommunicationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <DataExchangeRequest fdt:nodeId="myId" commandNumber="42" communicationReference="6B29FC40-
CA47-1067-B31D-00DD010662DA"/>
</FDT>

```

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTHARTCommunicationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
  <DataExchangeResponse commandNumber="1" communicationReference="6B29FC40-CA47-1067-B31D-
00DD010662DA">
    <fdt:CommunicationData byteArray="ff02"/>
    <Status deviceStatus="0">
      <CommandResponse value="1"/>
    </Status>
  </DataExchangeResponse>
</FDT>

```

16.2 Channel Schema

It is up to a DTM whether it provides any channels. If a DTM allows a frame-application, other DTMs, or a controller the direct access to its process values via HART protocol it should provide channel objects as described in this chapter. Only the complete description of all channels belonging to a HART command allow a proper access for external applications.

The description of channels, especially of the process values, allows the frame-application to support the device in a more sufficient way.

Used at: [IFdtChannel::GetChannelParameters\(\)](#)
[IFdtChannel::SetChannelParameters\(\)](#)

The XML document describes a how to access a channel via a HART command.

Attribute	Description
byteLength	Additional data type information especially for fieldbus specific data types like 3 byte values
commandNumber	Number of the HART command containing the channel value
frameApplicationTag	Frame application specific tag used for identification and navigation. The DTM should display this tag at channel specific user interfaces
gatewayBusCategory	Unique identifier for a supported bus type like Profibus or HART according to the FDT specific CATID
protectedByChannelAssignment	TRUE if the channels is set to read only by the frame-application. Usually set to TRUE if a channel assignment exists
value	Current value of a channel for read or write

Tag	Description
FDT	Root tag
FDTChannel	Description of the channel
FDTChannelType	Description of the channel component in case of channels with gateway functionality
ReadCommand	Description of the command to read the channel from a device
Reply	Description of the reply structure of a HART command according to the HART specification
Request	Description of the request structure of a HART command according to the HART specification
ResponseCodes	Collection of HART specific response codes according to the HART specification (known as COMMAND-SPECIFIC RESPONSE CODES).
WriteCommand	Description of the command to write the channel to a device

```
<Schema name="FDTHARTChannelParameterSchema" 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="byteLength" dt:type="ui1"/>
```

```

<AttributeType name="commandNumber" dt:type="ui4"/>
<AttributeType name="frameApplicationTag" dt:type="string"/>
<AttributeType name="gatewayBusCategory" dt:type="uuid"/>
<AttributeType name="protectedByChannelAssignment" dt:type="boolean"/>
<AttributeType name="value" dt:type="string"/>
<!--Definition of Elements-->
<ElementType name="Request" content="eltOnly" model="closed" order="many">
  <attribute type="fdt:nodel" required="no"/>
  <element type="fdt:ChannelReference" minOccurs="0" maxOccurs="*" />
</ElementType>
<ElementType name="ResponseCodes" content="eltOnly" model="closed">
  <attribute type="fdt:nodel" required="no"/>
  <element type="fdt:EnumeratorEntry" minOccurs="1" maxOccurs="*" />
</ElementType>
<ElementType name="Reply" content="eltOnly" model="closed">
  <attribute type="fdt:nodel" required="no"/>
  <element type="fdt:ChannelReference" minOccurs="0" maxOccurs="*" />
  <element type="ResponseCodes" minOccurs="0" maxOccurs="1" />
</ElementType>
<ElementType name="ReadCommand" content="eltOnly" model="closed">
  <attribute type="fdt:nodel" required="no"/>
  <attribute type="commandNumber" required="yes"/>
  <element type="Request" minOccurs="0" maxOccurs="1" />
  <element type="Reply" minOccurs="0" maxOccurs="1" />
  <element type="ResponseCodes" minOccurs="0" maxOccurs="1" />
</ElementType>
<ElementType name="WriteCommand" content="eltOnly" model="closed">
  <attribute type="fdt:nodel" required="no"/>
  <attribute type="commandNumber" required="yes"/>
  <element type="Request" minOccurs="0" maxOccurs="1" />
  <element type="Reply" minOccurs="0" maxOccurs="1" />
  <element type="ResponseCodes" minOccurs="0" maxOccurs="1" />
</ElementType>
<ElementType name="FDTChannel" content="eltOnly" model="closed" order="seq">
  <attribute type="fdt:nodel" required="no"/>
  <attribute type="fdt:tag" required="yes"/>
  <attribute type="fdt:id" required="yes"/>
  <attribute type="protectedByChannelAssignment" required="yes"/>
  <attribute type="fdt:dataType" required="yes"/>
  <attribute type="byteLength" required="yes"/>
  <attribute type="fdt:signalType" required="yes"/>
  <attribute type="frameApplicationTag" required="no"/>
  <element type="fdt:BitEnumeratorEntries" minOccurs="0" maxOccurs="1" />
  <element type="fdt:EnumeratorEntries" minOccurs="0" maxOccurs="1" />
  <element type="fdt:Unit" minOccurs="0" maxOccurs="1" />
  <element type="ReadCommand" minOccurs="0" maxOccurs="1" />
  <element type="WriteCommand" minOccurs="0" maxOccurs="1" />
  <element type="fdt:Alarms" minOccurs="0" maxOccurs="1" />
  <element type="fdt:Ranges" minOccurs="0" maxOccurs="1" />
  <element type="fdt:SubstituteValue" minOccurs="0" maxOccurs="1" />
</ElementType>
<ElementType name="FDTChannelType" content="eltOnly" model="closed">
  <attribute type="fdt:nodel" 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:nodel" required="no"/>
  <element type="FDTChannelType" minOccurs="1" maxOccurs="1" />
  <element type="FDTChannel" minOccurs="1" maxOccurs="1" />
</ElementType>
</Schema>

```

Example:

```

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTChannelParameterSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
  <FDTChannelType>

```



```

    <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
  </FDTChannelType>
  <FDTChannel fdt:tag="myTag" fdt:id="PV_UNIT" protectedByChannelAssignment="0" fdt:dataType="byte"
byteLength="1" fdt:signalType="input">
    <fdt:EnumeratorEntries>
      <fdt:EnumeratorEntry index="7" name="bar"/>
      <fdt:EnumeratorEntry index="8" name="mbar"/>
    </fdt:EnumeratorEntries>
    <ReadCommand commandNumber="1">
      <Reply>
        <fdt:ChannelReference idref="PV_UNIT"/>
        <fdt:ChannelReference idref="PV"/>
      </Reply>
      <ResponseCodes>
        <fdt:EnumeratorEntry index="8" name="Warning: Update Failure"/>
      </ResponseCodes>
    </ReadCommand>
    <WriteCommand commandNumber="44">
      <Request>
        <fdt:ChannelReference idref="PV_UNIT"/>
      </Request>
      <Reply>
        <fdt:ChannelReference idref="PV_UNIT"/>
      </Reply>
    </WriteCommand>
  </FDTChannel>
</FDT>

<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTHARTChannelParameterSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
  <FDTChannelType>
    <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
  </FDTChannelType>
  <FDTChannel fdt:tag="myTag" fdt:id="PV" protectedByChannelAssignment="0" fdt:dataType="float"
byteLength="4" fdt:signalType="output">
    <ReadCommand commandNumber="1">
      <Reply>
        <fdt:ChannelReference idref="PV_UNIT"/>
        <fdt:ChannelReference idref="PV"/>
      </Reply>
      <ResponseCodes>
        <fdt:EnumeratorEntry index="8" name="Warning: Update Failure"/>
      </ResponseCodes>
    </ReadCommand>
    <fdt:Alarms>
      <fdt:Alarm alarmType="lowAlarm">
        <fdt:StaticValue staticValue="25"/>
      </fdt:Alarm>
      <fdt:Alarm alarmType="highAlarm">
        <fdt:StaticValue staticValue="100"/>
      </fdt:Alarm>
    </fdt:Alarms>
    <fdt:Ranges>
      <fdt:Range>
        <fdt:LowerRange>
          <fdt:ChannelReference idref="PV_LOWER_RANGE_VALUE"/>
        </fdt:LowerRange>
        <fdt:UpperRange>
          <fdt:ChannelReference idref="PV_UPPER_RANGE_VALUE"/>
        </fdt:UpperRange>
        <fdt:Unit>
          <fdt:ChannelReference idref="PV_RANGE_VALUES_UNITS_CODE"/>
        </fdt:Unit>
      </fdt:Range>
    </fdt:Ranges>
  </FDTChannel>
</FDT>

```


16.3 Topology Scan Schema

used at: IDtm::OnScanResponse()

The XML document describes one entry in the list of scanned HART-Devices.

Tag	Description
HARTDevice	Definiton of a HART device concerning the scan response

```
<?xml version="1.0"?>
<Schema name="DTMHARTDeviceSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml" xmlns:fdthart="x-schema:FDTHARTCommunicationSchema.xml"
xmlns:dtminfo="x-schema:DTMInformationSchema.xml">
  <!--Definition of Elements-->
  <ElementType name="HARTDevice" content="eltOnly" model="closed">
    <attribute type="fdt:nodeId" required="no"/>
    <element type="fdthart:LongAddress" minOccurs="0" maxOccurs="1"/>
    <attribute type="fdthart:manufacturerId" required="no"/>
    <attribute type="fdthart:deviceTypeId" required="no"/>
    <attribute type="fdt:subDeviceType" required="no"/>
    <attribute type="fdt:tag" required="yes"/>
    <attribute type="fdthart:shortAddress" required="no"/>
  </ElementType>
</Schema>
```

17 Appendix – Update Hints

This chapter contains information about changes between version 1.1 and version 1.2

	Topic	Description
3.2.2	DTM State Machine	Shortcut from state 'new enabled' to 'zombie'
3.3.1.3, 3.3.5.1	IDtm::GetFunctions(), IDtmDocumentation::GetDocumentation()	Support of function orientated documentation, refer to DTMFunctionsSchema, element 'Document' for additional information
3.3.1.7	IDtm::PrepareToRelease()	Hint: A DTM has to send IDtmEvents::OnPreparedToRelease() to inform the frame-application that the DTM can be released
3.3.13.1	IFdtEvents::OnChildParameterChanged()	Clarification concerning the frame-application specific behavior
3.4.1.2	IDtmActiveXControl::PrepareToRelease()	Clarification concerning asynchronous behavior
3.5.2.3	IFdtChannelActiveXInformation::GetChannelFunctions()	New method to guarantee proper functionality
3.5.3	IFdtCommunication	Bugfix concerning sequence handling
3.6.1.2	IFdtChannelActiveXControl::PrepareToRelease()	Clarification concerning asynchronous behavior
3.7.1.5	IDtmEvents::OnChannelFunctionChanged()	New method to guarantee proper functionality
3.7.2.3	IDtmAuditTrailEvents::OnStartTransaction()	Bugfix concerning documentation of method behavior
5.4	Download of master configuration	Clarification concerning download of fieldbus master configuration
6.1.2	Component categories	Revised table of examples
10	IDL	Revised interface definition language file. Due to guarantee proper functionalities, changes concerning interface definitions where made. Take into account that some interface related UUIDs where changed.
-/-	XML-Files	Revised XML-Files: <ul style="list-style-type: none"> • DTMChannelFunctionsInstance.xml • DTMChannelFunctionsSchema.xml • DTMFunctionsInstance1.xml • DTMFunctionsInstance2.xml • DTMFunctionsSchema.xml • DTMInfoListInstance.xml • DTMInformationInstance.xml • DTMTopologyScanSchema.xml • FDTDataTypesSchema.xml • FDTHARTCommunicationSchema.xml • FDTProfibusDPV0CommunicationSchema.xml • FDTProfibusDPV1CommunicationSchema.xml

© Copyright by:

PROFIBUS Nutzerorganisation e.V.
Haid-und-Neu-Str. 7
D-76131 Karlsruhe

Phone: ++49 721 / 96 58 590

Fax: ++49 721 / 96 58 589

pi@profibus.com

www.profibus.com